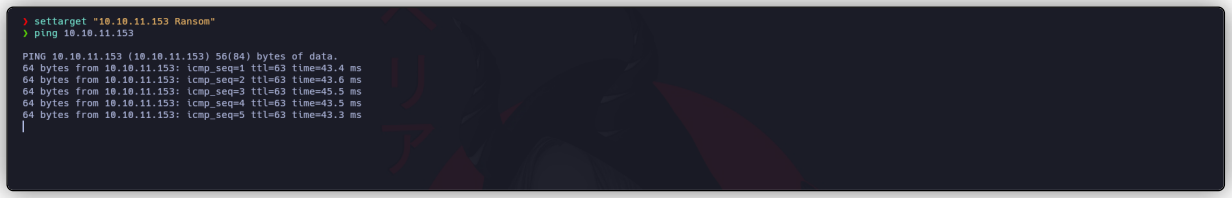
247- RANSOM

1. RANSOM

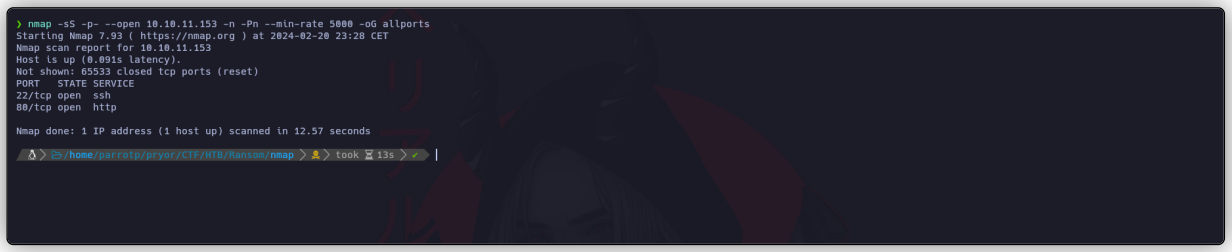
https://app.hackthebox.com/machines/Ransom

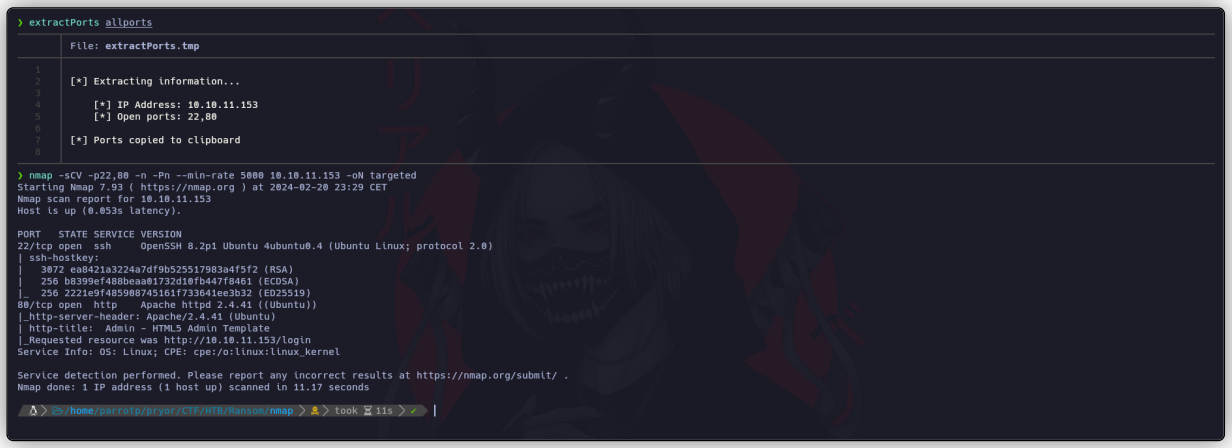
1.1. Preliminar

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

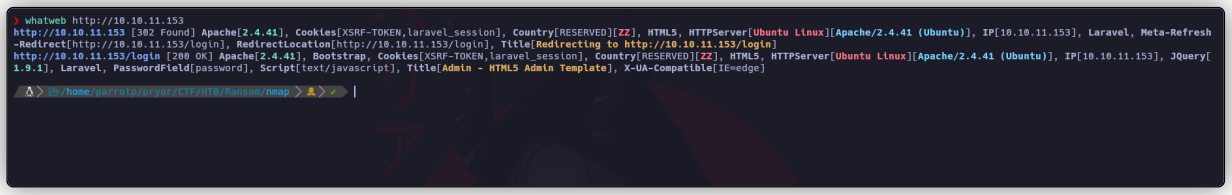
- 

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`.
  - 

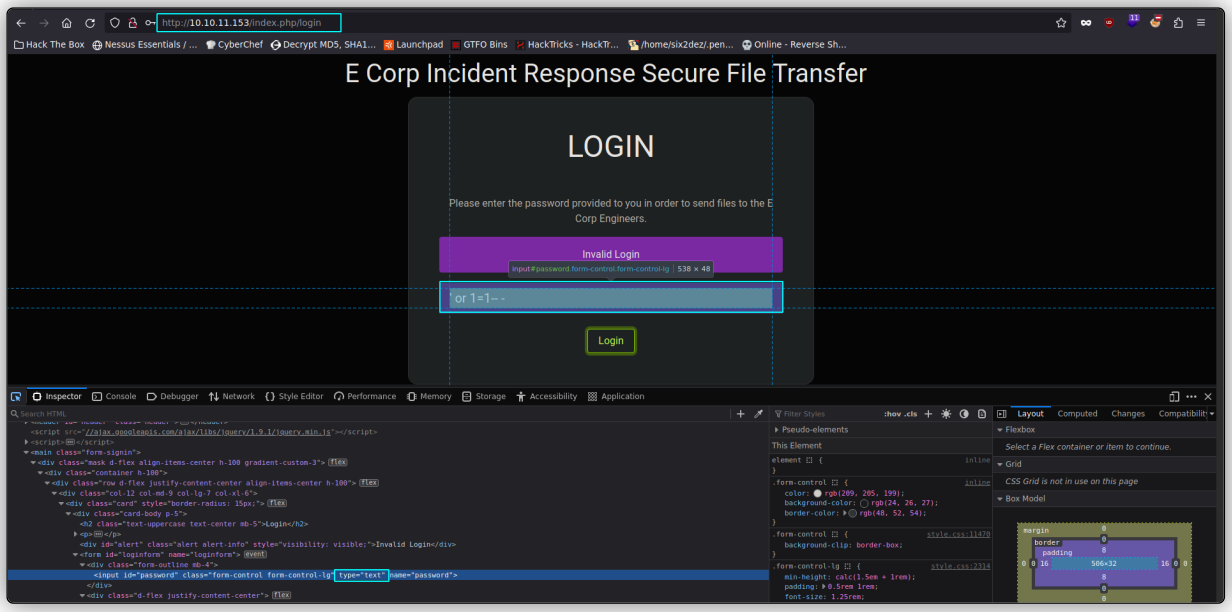
1.3. Tecnologías web

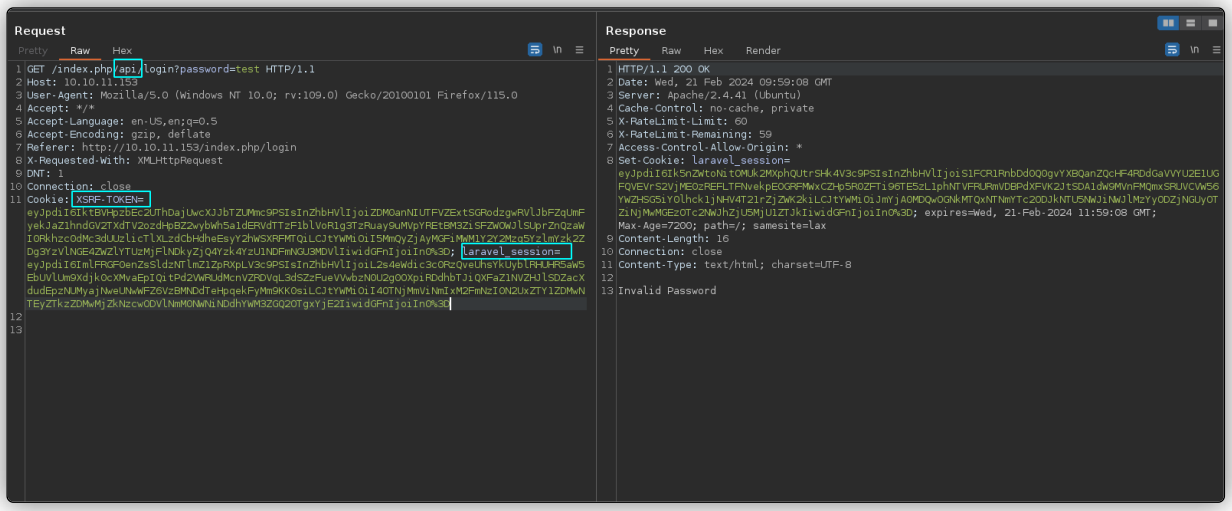
- **Whatweb**: nos reporta lo siguiente. La versión de *JQuery 1.9.1* es bastante También vemos que se está usando el framework de **Laravel**.
  - 

> - **Laravel** es un framework de desarrollo de aplicaciones web de código abierto, basado en *PHP*. Su primera versión fue lanzada en 2011. Desde entonces, se ha convertido en uno de los frameworks de PHP más populares y ampliamente utilizados en la comunidad de desarrollo web.

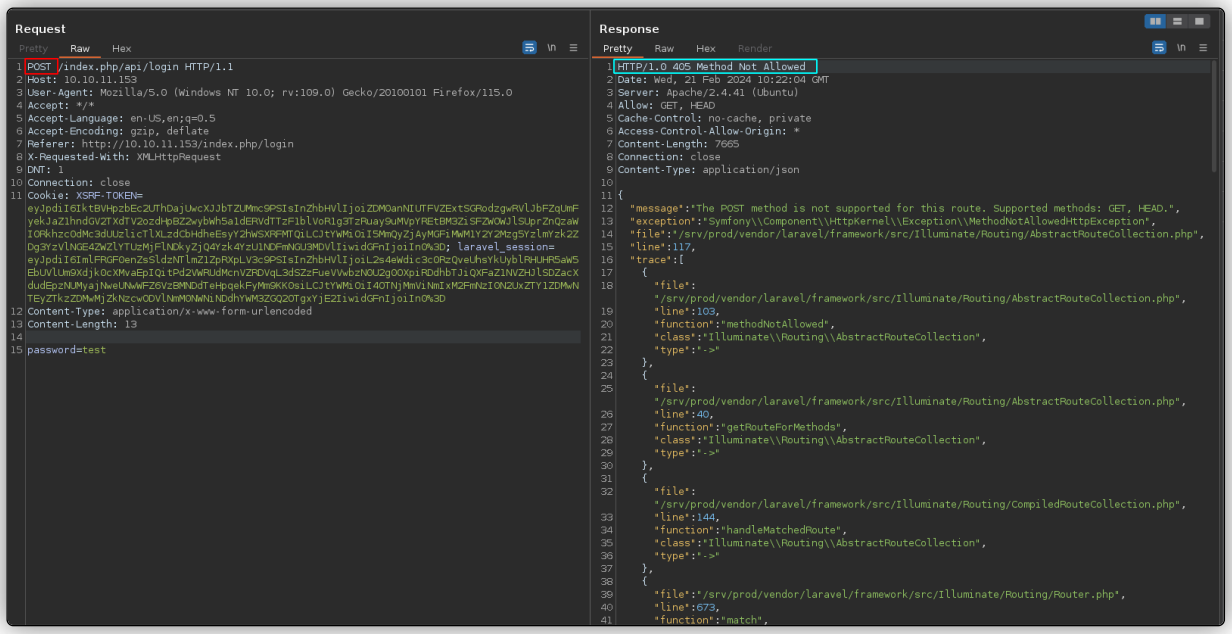
> Este framework sigue el patrón de arquitectura *MVC (Modelo-Vista-Controlador)*, lo que significa que separa la lógica de la aplicación en estos tres componentes principales.

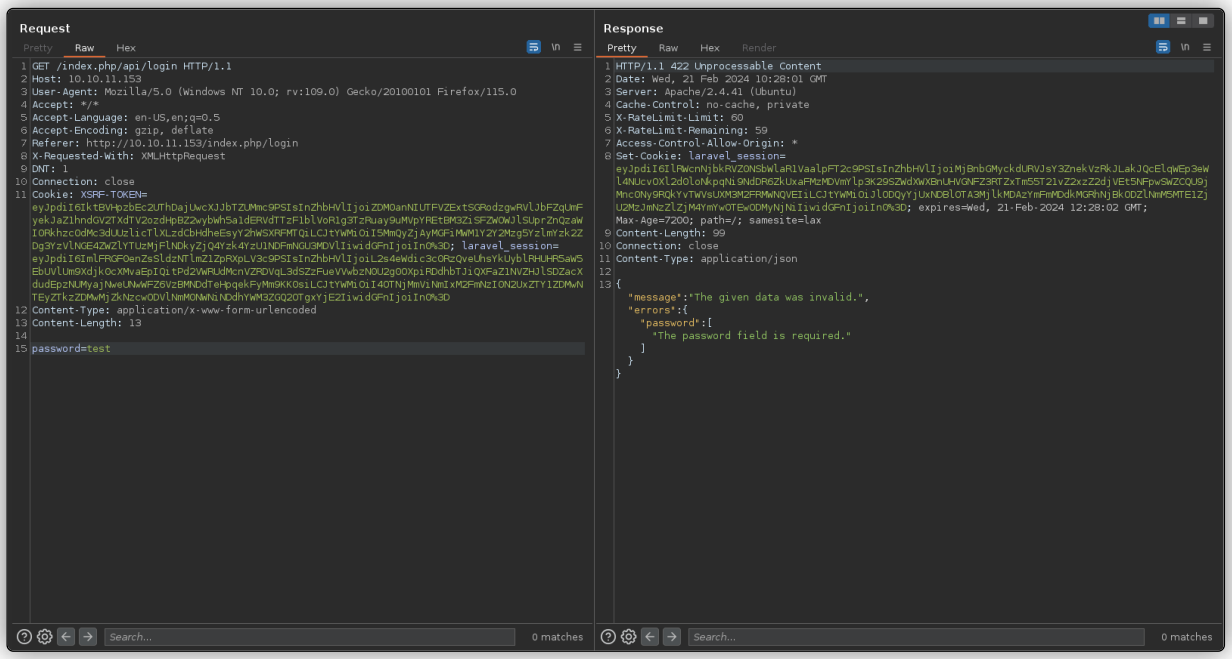
1.4. Type Juggling

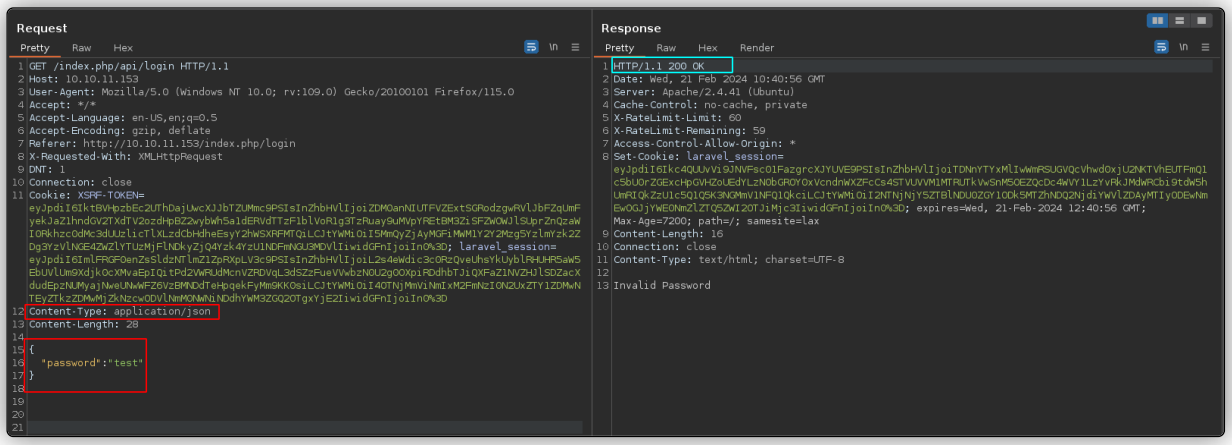
- Ingresamos a la página web. Tenemos un panel de login. Cambiamos el campo `type="text"` en las herramientas de desarrollador para que, de este modo, podamos ver lo que escribimos. Dicho esto, probamos diferentes credenciales, incluso inyecciones SQL, pero no conseguimos acceso.
  - 

- Por tanto, interceptamos esta petición con **Burp Suite**. Vemos que se nos están configurando dos cookies de sesión: *XSRF-TOKEN*, y *laravel_session*. También vemos que hay una **API** por detrás: /index.php/api.
  - 

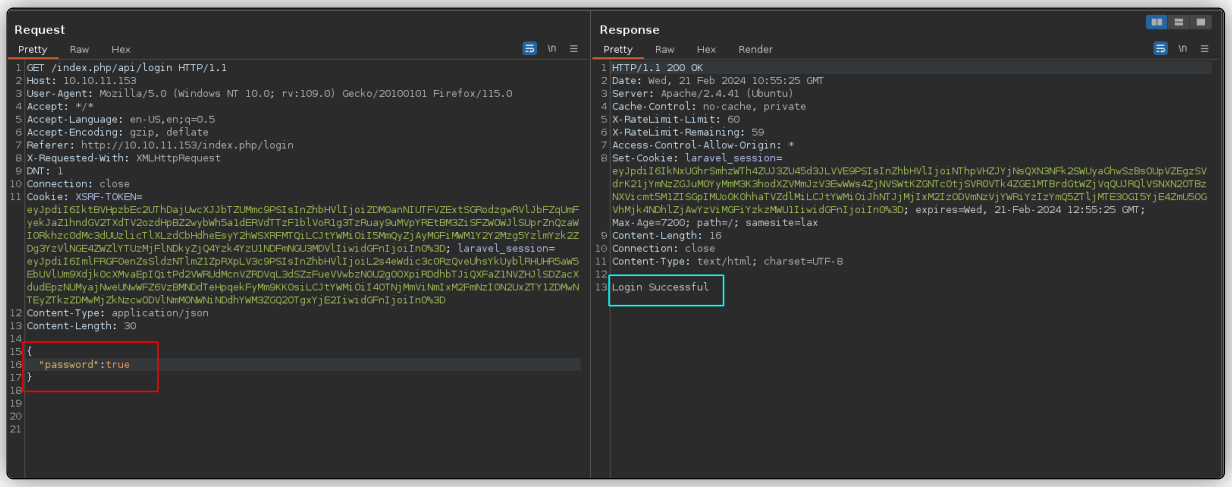
- Una de las cosas que podemos probar es enviar la petición cambiando el método de **GET** a **POST** y ver cómo reacciona la página. Vemos que el servidor, en este punto, no admite este método *(405: Method Not Allowed)*.

- 

- Podríamos intentar, manteniendo igual el resto de la estructura de la petición, cambiar simplemente el método a **GET**. Es interesante ahora observar que el servidor responde con que no puede procesar el contenido *(422: Unprocessable Content)*.
  - 

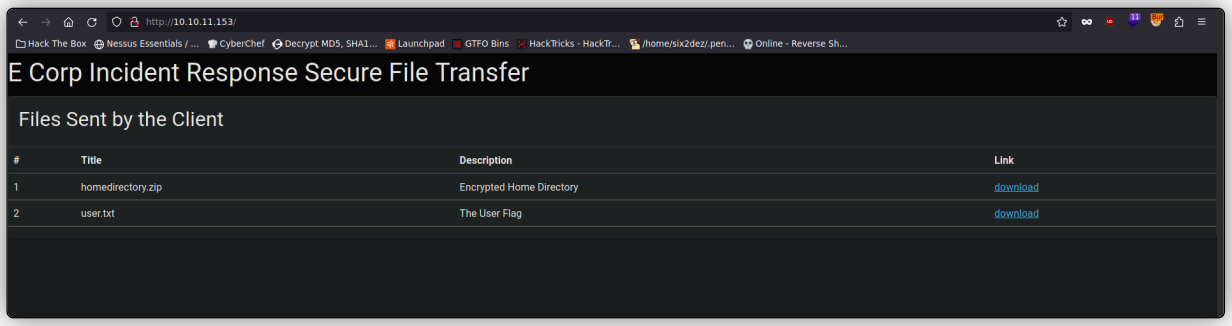
- Esta respuesta se debe a que el servidor espera los datos en formato **JSON**: *(Content-Type: application/json)*. Sabiendo esto, cambiamos el Content-Type en nuestra petición y estructuramos los datos en **JSON**. Esta vez, el servidor acepta la petición.
  - 

- El riesgo aquí está en cómo se están representando los datos, ya que dependiendo de cómo se esté aplicando la validación del campo de contraseña por detrás, a nivel de código, podríamos efectuar un Type Juggling, que es vulnerabilidad típica de PHP. Para ello, necesitamos forzar el cambio de tipo de dato. En este caso, cambiamos el tipo del campo *password* a `true`, es decir, lo convertimos a un valor booleano. De este modo, conseguimos el acceso.

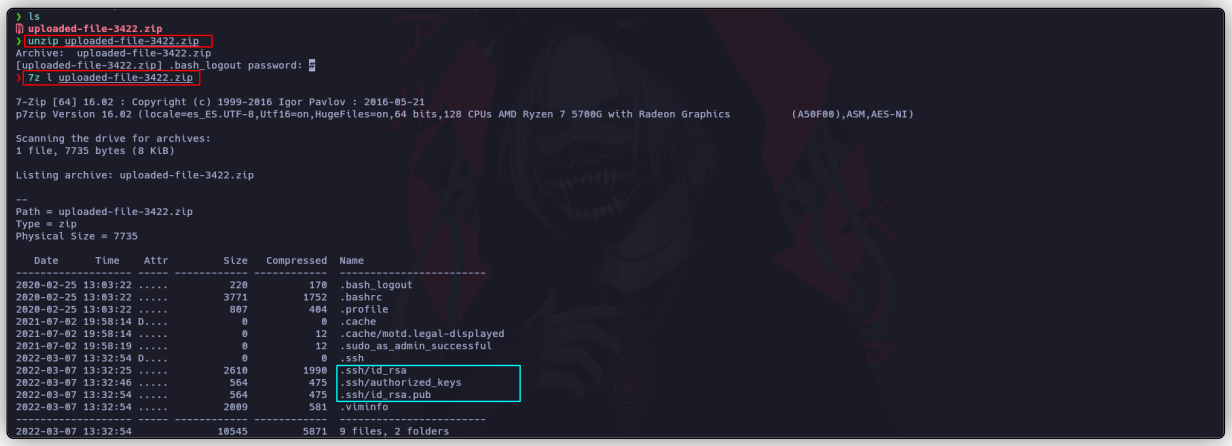


1.5. Cracking zip pass with Zip2john

- Una vez hemos conseguido el acceso vemos lo siguiente. Descargamos el comprimido *homedirectory.zip*.

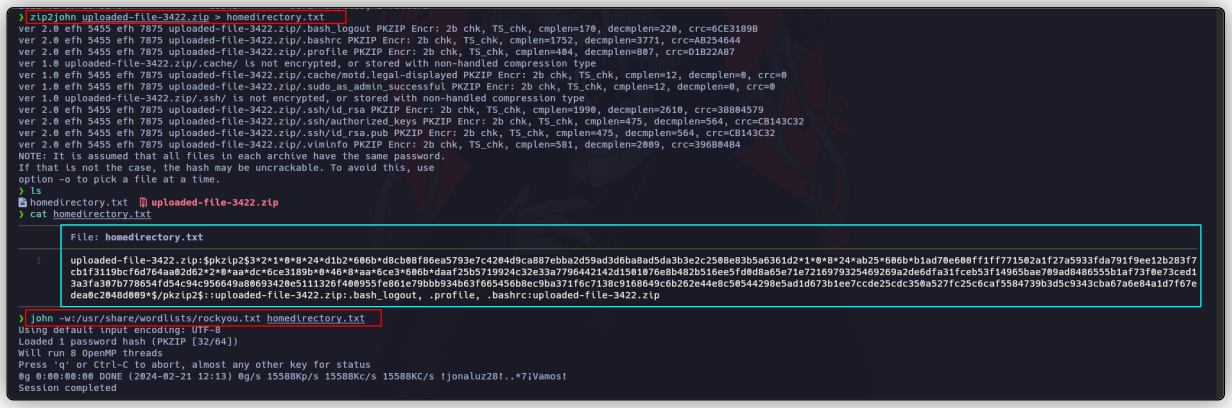


- Hacemos un `unzip` para descomprimir el archivo pero nos pide contraseña. Aún así, podemos listar el contenido de este comprimido con `7z l`. Vemos que tenemos claves SSH dentro. Por tanto, si conseguimos descomprimirlo, podríamos obtener acceso a la máquina por SSH.



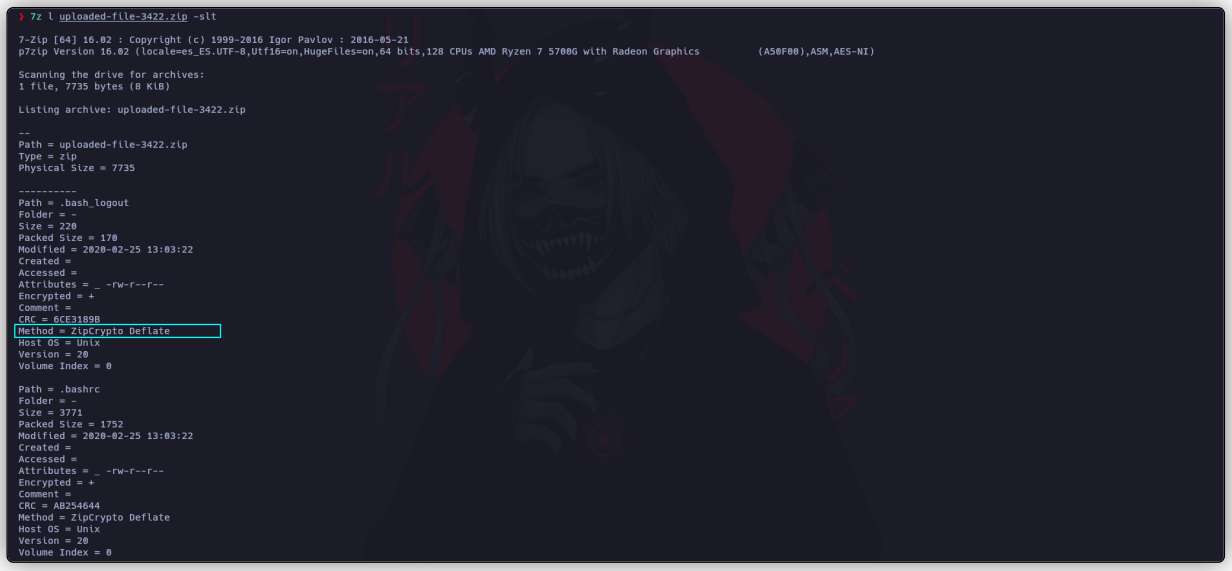
- Usamos `zip2john uploaded-file-3422.zip > homedirectory.txt` para realizar un ataque de fuerza bruta y obtener un hash de la contraseña. Seguidamente, ejecutando `john -`

`w:/usr/share/wordlists/rockyou.txt homedirectory.txt`, tratamos de romper este hash, pero no lo conseguimos.



1.6. Plaintext attack with Bkcrack

- Intentamos listar el contenido del comprimido con un poco más de información técnica. Esto lo podemos hacer con: `7z l uploaded-file-3422.zip -slt`. Lo que podemos ver a continuación es que nos tenemos un archivo comprimido encriptado *Method = ZipCrypto Deflate*.



- En este punto, podríamos tratar de efectuar un **Plaintext attack**. Para ello, necesitaremos disponer de un archivo que, medianamente, en texto claro, sepamos cómo están compuestas sus líneas. En este caso, como sabemos que este comprimido tiene el archivo *.bash_logout*, y éste, generalmente, es similar en todos los sistemas (es decir, no sufre alteraciones), podríamos usarlo para compararlo con el nuestro (el cual está en texto claro). Por tanto, la idea aquí es comparar el *.bash_logout* del comprimido (cifrado) con el *.bash_logout* de nuestro mismo sistema (texto claro). Copiamos ahora un *.bash_logout* a nuestro directorio actual de trabajo. Hacemos: `wc -c .bash_logout` para ver los bytes de este archivo. Vemos que tiene *220 bytes*, la misma cantidad que tiene el del archivo comprimido.

```
❯ locate .bash_logout
/snap/core20/2105/etc/skel/.bash_logout
/snap/core20/2182/etc/skel/.bash_logout
/snap/core22/1033/etc/skel/.bash_logout
/usr/share/doc/adduser/examples/adduser.local.conf.examples/skel/dot.bash_logout
❯ cat /snap/core20/2182/etc/skel/.bash_logout
```

```
File: /snap/core20/2182/etc/skel/.bash_logout

1  # ~/.bash_logout: executed by bash(1) when login shell exits.
2
3  # when leaving the console clear the screen to increase privacy
4
5  if [ "$SHLVL" = 1 ]; then
6      [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
7  fi
```

```
❯ cp /snap/core20/2182/etc/skel/.bash_logout .
❯ ls
homedirectory.txt  uploaded-file-3422.zip
❯ ls -la
drwxr-xr-x root   root    102 B  Wed Feb 21 12:49:34 2024 .
drwxr-xr-x root   root     38 B  Tue Feb 20 23:28:05 2024 ..
.rw-r--r-- root   root    220 B  Wed Feb 21 12:49:34 2024 .bash_logout
.rw-r--r-- root   root    688 B  Wed Feb 21 12:07:49 2024 homedirectory.txt
.rw-r--r-- parrotp parrotp 7.6 KB Wed Feb 21 12:02:24 2024 uploaded-file-3422.zip
❯ wc -c .bash_logout
220 .bash_logout
❯ 7z l uploaded-file-3422.zip

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=es_ES.UTF-8,Utf16=on,HugeFiles=on,64 bits,128 CPUs AMD Ryzen 7 5700G with Radeon Graphics          (A50F00),ASM,AES-NI)

Scanning the drive for archives:
1 file, 7735 bytes (8 KiB)

Listing archive: uploaded-file-3422.zip

--
Path = uploaded-file-3422.zip
Type = zip
Physical Size = 7735

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2020-02-25 13:03:22 .....          220          170  .bash_logout
2020-02-25 13:03:22 .....         3771         1752  .bashrc
2020-02-25 13:03:22 .....          807          404  .profile
2021-07-02 19:58:14 D....            0            0  .cache
2021-07-02 19:58:14 .....            0           12  .cache/motd.legal-displayed
```

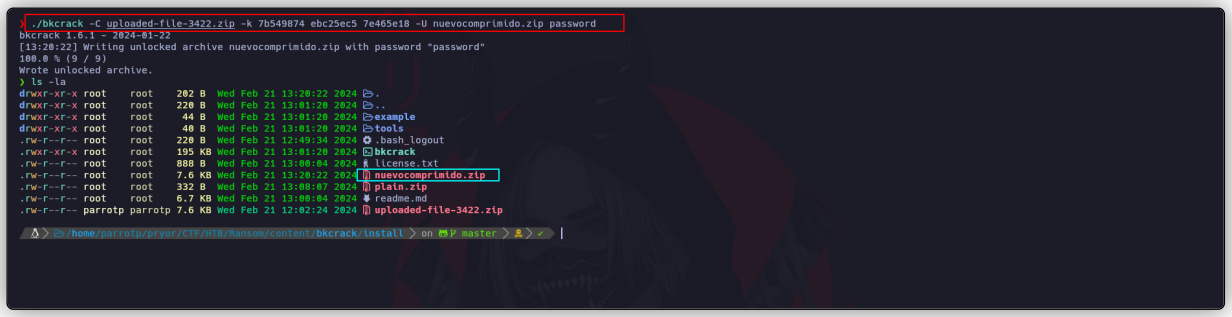
- Seguidamente, nos clonamos la herramienta **Bkcrack** en nuestro directorio de trabajo, y la compilamos para poder usarla. Una vez tengamos la herramienta instalada, tendremos que crearnos primero un archivo **ZIP** que contenga nuestro *.bash_logout*, para ello: `zip plain.zip .bash_logout`. Podemos descargar esta herramienta del siguiente enlace.
  - https://github.com/kimci86/bkcrack

```
❯ ls -la
drwxr-xr-x root    root    146 B  Wed Feb 21 13:03:45 2024 .
drwxr-xr-x root    root    220 B  Wed Feb 21 13:01:20 2024 ..
drwxr-xr-x root    root     44 B  Wed Feb 21 13:01:20 2024 example
drwxr-xr-x root    root     40 B  Wed Feb 21 13:01:20 2024 tools
.rw-r--r-- root    root    220 B  Wed Feb 21 12:49:34 2024 .bash_logout
.rwxr-xr-x root    root    195 KB Wed Feb 21 13:01:20 2024 bkcrack
.rw-r--r-- root    root    888 B  Wed Feb 21 13:00:04 2024 license.txt
.rw-r--r-- root    root    6.7 KB Wed Feb 21 13:00:04 2024 readme.md
.rw-r--r-- parrotp parrotp 7.6 KB Wed Feb 21 12:02:24 2024 uploaded-file-3422.zip
❯ zip plain.zip .bash_logout
  adding: .bash_logout (deflated 28%)
❯ ls -la
drwxr-xr-x root    root    164 B  Wed Feb 21 13:08:07 2024 .
drwxr-xr-x root    root    220 B  Wed Feb 21 13:01:20 2024 ..
drwxr-xr-x root    root     44 B  Wed Feb 21 13:01:20 2024 example
drwxr-xr-x root    root     40 B  Wed Feb 21 13:01:20 2024 tools
.rw-r--r-- root    root    220 B  Wed Feb 21 12:49:34 2024 .bash_logout
.rwxr-xr-x root    root    195 KB Wed Feb 21 13:01:20 2024 bkcrack
.rw-r--r-- root    root    888 B  Wed Feb 21 13:00:04 2024 license.txt
.rw-r--r-- root    root    332 B  Wed Feb 21 13:08:07 2024 plain.zip
.rw-r--r-- root    root    6.7 KB Wed Feb 21 13:00:04 2024 readme.md
.rw-r--r-- parrotp parrotp 7.6 KB Wed Feb 21 12:02:24 2024 uploaded-file-3422.zip
❯ 7z l plain.zip

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=es_ES.UTF-8,Utf16=on,HugeFiles=on,64 bits,128 CPUs AMD Ryzen 7 5700G with Radeon Graphics          (A50F00),ASM,AES-NI)

Scanning the drive for archives:
1 file, 332 bytes (1 KiB)

Listing archive: plain.zip

--
Path = plain.zip
Type = zip
Physical Size = 332

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2024-02-21 12:49:34 .....          220          158  .bash_logout

2024-02-21 12:49:34                220          158  1 files

 /home/parrotp/pryor/CTF/HTB/Ransom/content/bkcrack/install  on  master
```

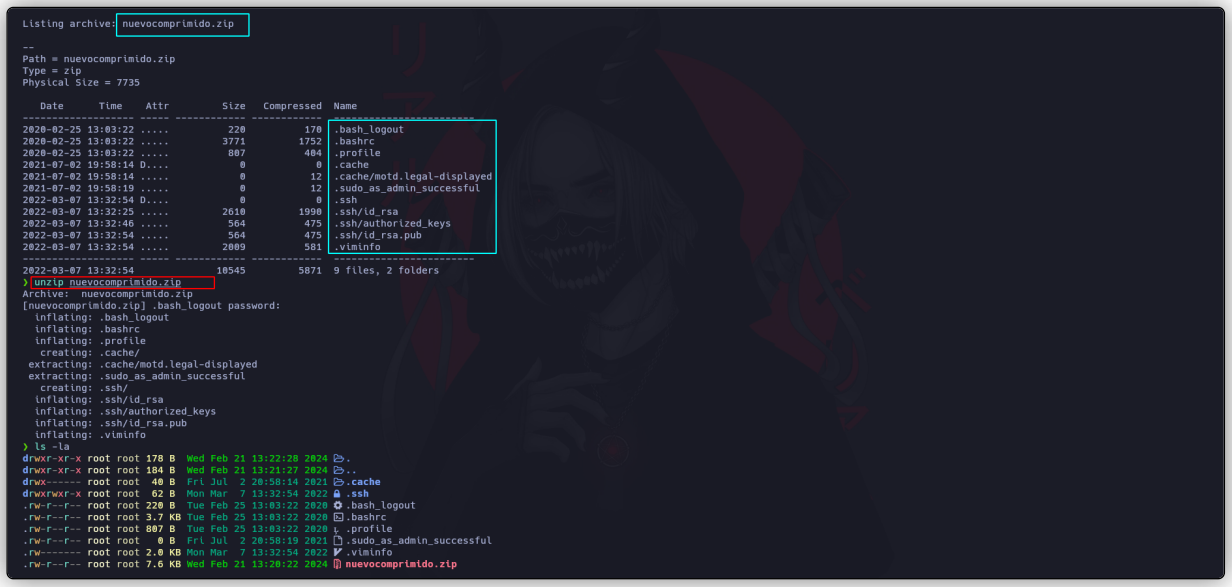
- Para ejecutar la herramienta tendremos que indicar: el ZIP encriptado, el archivo dentro que esté cifrado, nuestro ZIP y el archivo respectivo en texto claro: `./bkcrack -C uploaded-file-3422.zip -c ".bash_logout" -P plain.zip -p ".bash_logout"`. Esto ejecutará un ataque que tratará de obtener un par de claves. Al cabo de unos minutos, obtenemos las claves. Las copiamos.

```
❯ ./bkcrack -C uploaded-file-3422.zip -c ".bash_logout" -P plain.zip -p ".bash_logout"
bkcrack 1.6.1 - 2024-01-23
[13:12:46] Z reduction using 151 bytes of known plaintext
100.0 % (151 / 151)
[13:12:47] Attack on 56903 Z values at index 6
Keys: 7b549874 ebc25ec5 7e465e18
75.6 % (43026 / 56903)
Found a solution. Stopping.
You may resume the attack with the option: --continue-attack 43026
[13:13:17] Keys
7b549874 ebc25ec5 7e465e18

 /home/p/pryor/CTF/HTB/Ransom/content/bkcrack/install  on  master  took 31s
```

- Luego con **Bkcrack**: `./bkcrack -C uploaded-file-3422.zip -k 7b549874 ebc25ec5 7e465e18 -U nuevocomprimido.zip password`. Es decir, especificamos las claves con `-k` y creamos un nuevo comprimido con `-U`. Adicionalmente, asignaremos una contraseña que queramos.



- Este nuevo archivo comprimido tiene el mismo contenido que el encriptado. La diferencia es que lo podemos descomprimir ahora: `unzip nuevocomprimido.zip`. Tenemos acceso de este modo a las claves SSH.



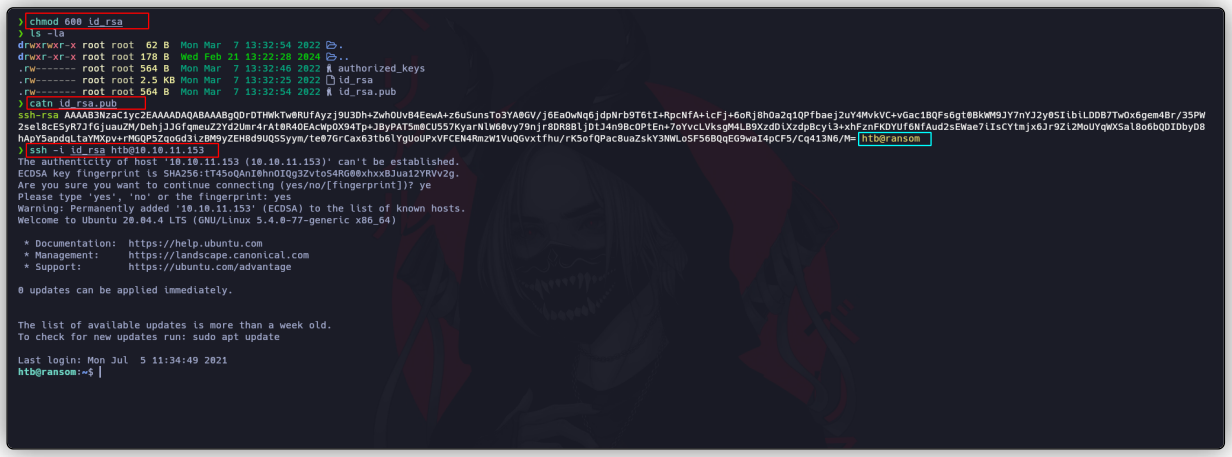
> - La **encriptación de archivos en un archivo ZIP** es una función que algunos programas de compresión de archivos ofrecen para proteger los datos comprimidos con una contraseña. Esto significa que los archivos dentro del archivo ZIP están encriptados y solo pueden ser descomprimidos con la contraseña correcta.

> - Un **Plaintext attack (KPA)** es un tipo de ataque en criptografía donde el atacante tiene acceso tanto al **texto plano (llamado criba)** como a su versión **encriptada (texto cifrado)**. Estos pueden ser utilizados para revelar claves secretas y libros de códigos.

> - **Bkcrack** es una herramienta de software utilizada para realizar ataques criptográficos específicos en archivos comprimidos en formato *ZIP*. Su propósito principal es recuperar las claves internas de cifrado del algoritmo de cifrado usado en ZIP, conocido como *ZIP Legacy Encryption*. Esto permite desencriptar archivos ZIP sin necesidad de conocer la contraseña original.

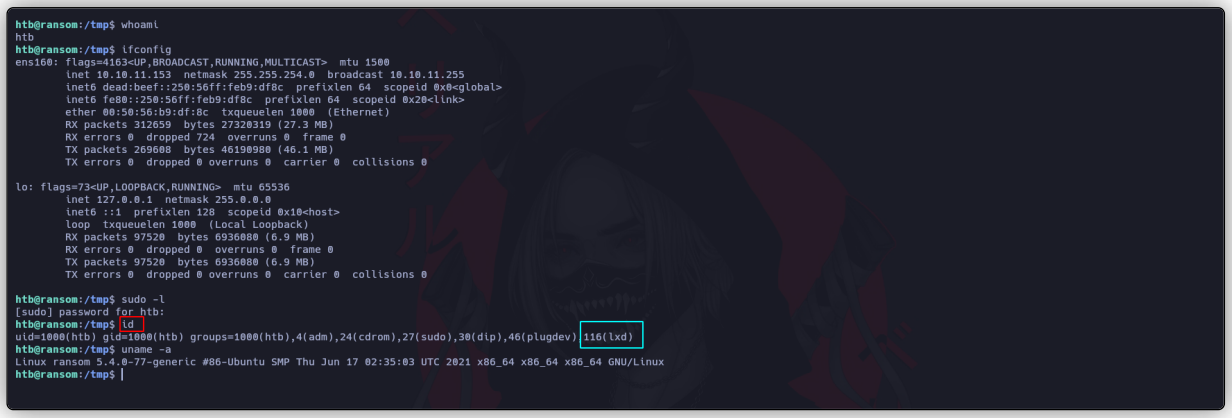
1.7. SSH access

- Tenemos ahora la clave id_rsa para conectarnos por SSH a la máquina víctima. No obstante, no tenemos un usuario para ello. Probamos con root, pero nos pide contraseña. En cualquier caso, como siempre, asignamos los permisos necesarios a la clave con `chmod 600 id_rsa`. Vimos un posible usuario en la clave pública id_rsa.pub: el usuario *htb*. Así que nos conectamos con `ssh -i id_rsa htb@10.10.11.153`. Conseguimos acceso.

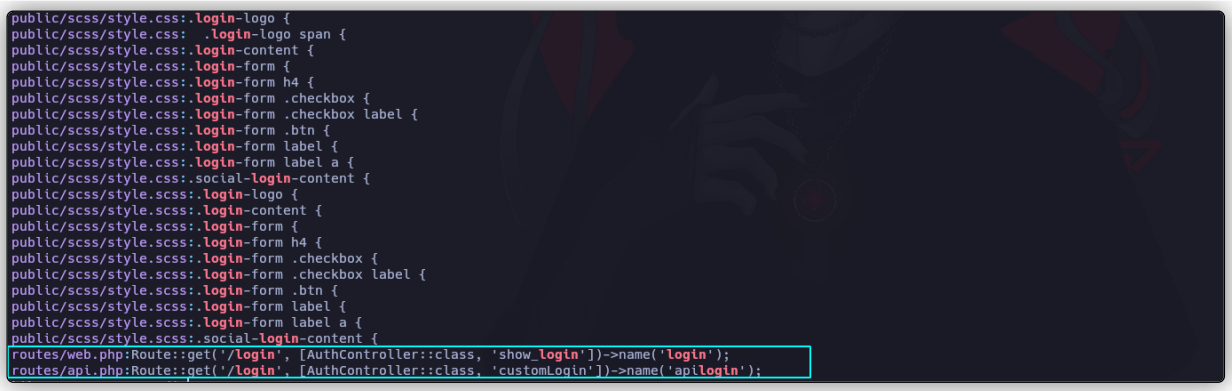


1.8. Privesc via hardcoded root credentials

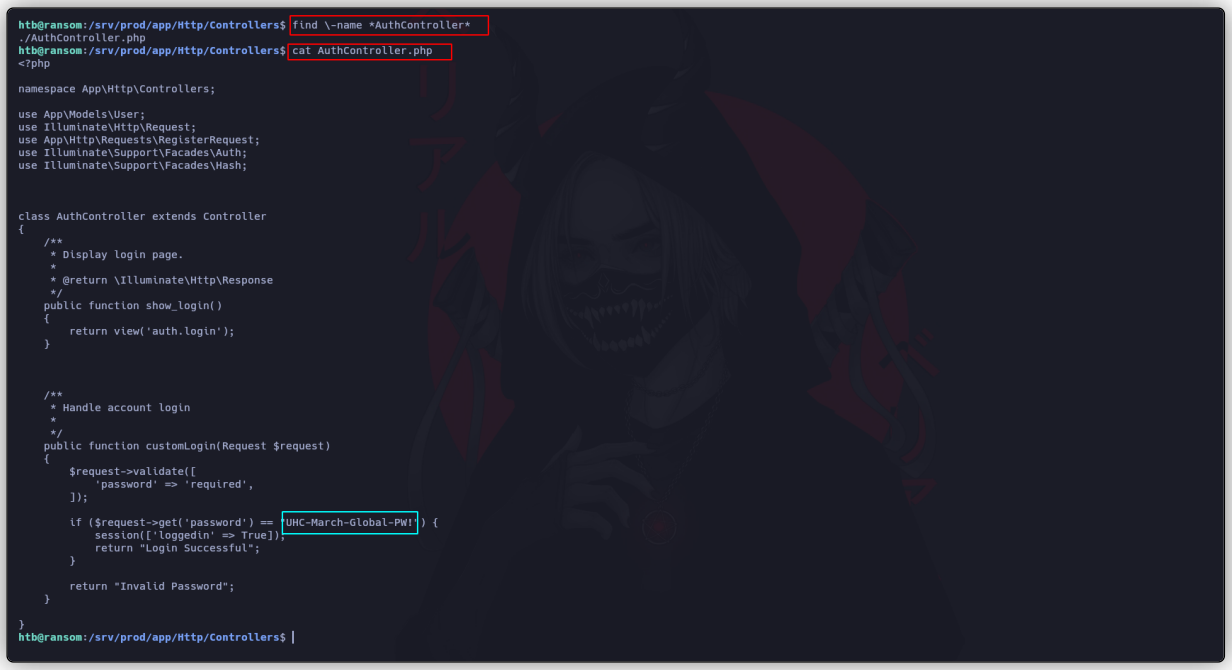
- Estamos como usuario *htb*. Al hacer `id`, vemos que estamos en el grupo lxd. Pertenecer a este grupo presenta una vía potencial de escalar privilegios.



- No obstante, vamos a buscar el archivo de login de la web donde se entablaba la autenticación. Buscamos por /etc/apache2/sites-enabled/000-default.conf . Aquí se hace referencia a otro directorio: /srv/prod/public, que es donde se encontraba el ZIP encriptado. De momento, no encontramos el archivo. Ejecutamos ahora: `grep -r "login"`. De esta búsqueda, nos interesan las dos últimas líneas, concretamente, eso de *AuthController*.



- Realizamos otra búsqueda: `find \-name *AuthController*`. Vemos una ruta: /app/Http/Controllers/AuthController.php. Al leer este archivo, vemos la contraseña del usuario **root**, la cual está **harcodeada**. Asimismo, vemos en este archivo que se está usando `==` para validar la misma. De ahí que pudiéramos efectuar anteriormente el **Type juggling**. En cualquier caso, teniendo ya esta contraseña, migramos la sesión a **root**. Obtenemos la última bandera.



> - **Hardcodeado** se refiere a la práctica de incluir valores directamente dentro del código fuente en lugar de obtener esos valores de manera dinámica o mediante la interacción con el usuario u otros sistemas. El problema con el hardcodeado es que puede hacer que el código sea menos flexible y más difícil de mantener. Si necesitas cambiar ese valor en el futuro, tendrías que modificar directamente el código fuente en lugar de simplemente cambiar un parámetro o configuración externa. Además, puede dificultar la reutilización del código en diferentes contextos. Por estas razones, en general, es preferible evitar el hardcodeado siempre que sea posible y utilizar variables o configuraciones externas para valores que puedan necesitar cambios.