COZYHOSTING

1. COZYHOSTING

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

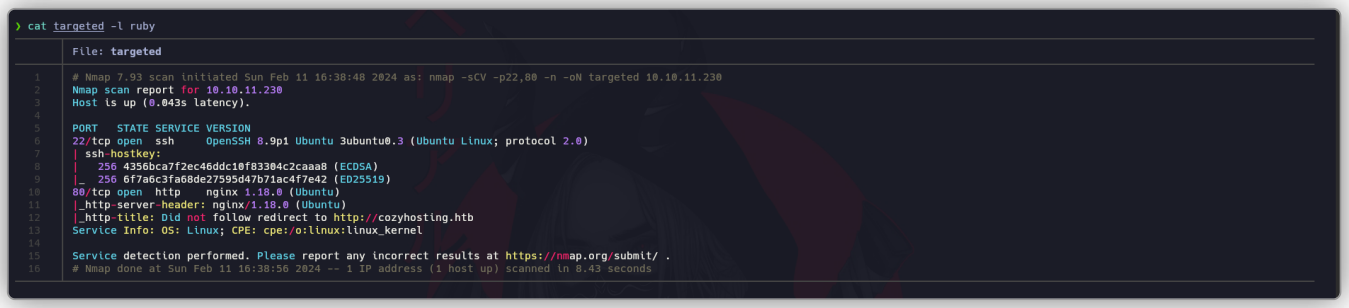
1.1. Preliminar

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*.

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`. Evidencia en archivo *targeted*.



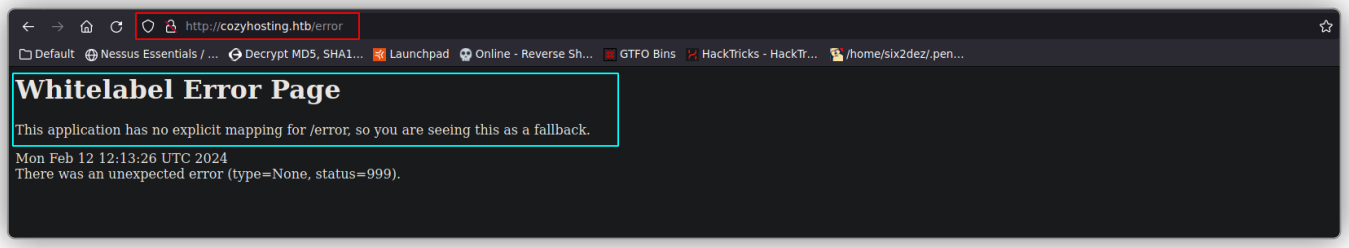
1.3. Tecnologías web

Whatweb: nos reporta lo siguiente. Vemos que no se ha podido resolver *cozyhosting.htb*, por tanto añadimos este dominio a nuestro /etc/hosts. Seguidamente, lanzamos de nuevo el escaneo con Whatweb.

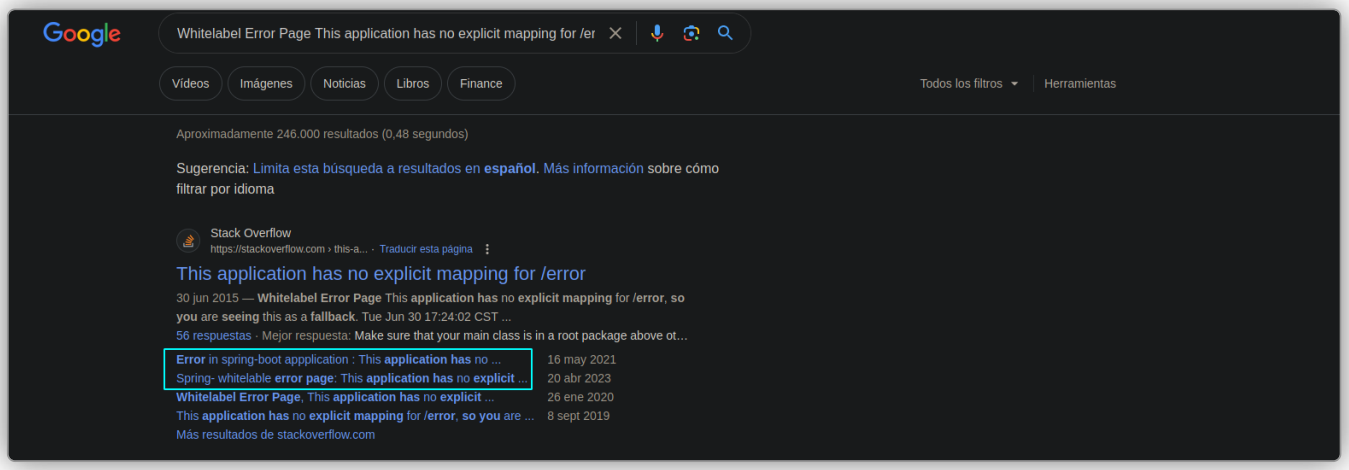
1.4. Fuzzing web

Gobuster: nos reporta los siguientes directorios.

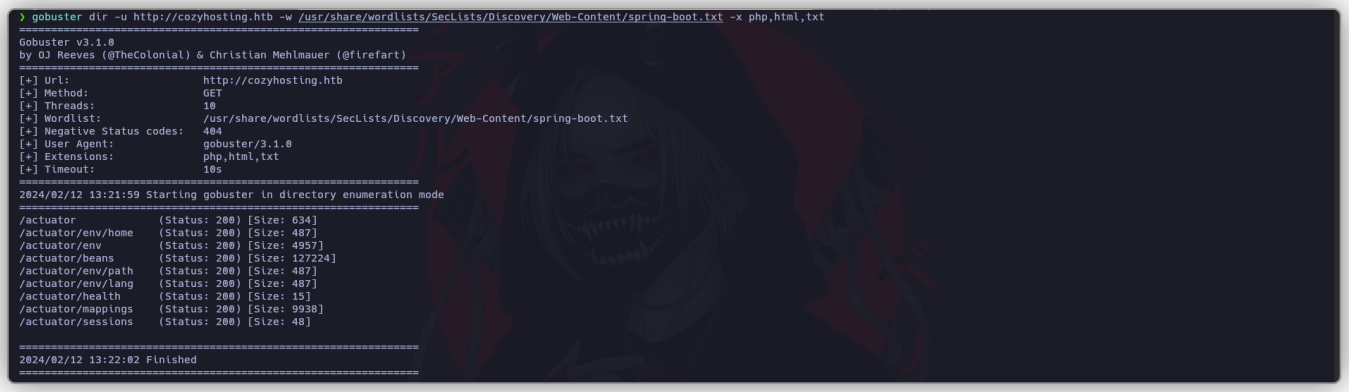
Tratamos de entrar a /admin, pero somos redireccionados a la página principal. Seguidamente, entramos a /error y encontramos lo siguiente.



Buscamos información sobre este error, y vemos que es un error típico de Spring Boot. Spring Boot es un marco de trabajo (framework) de código abierto para el desarrollo de aplicaciones Java.



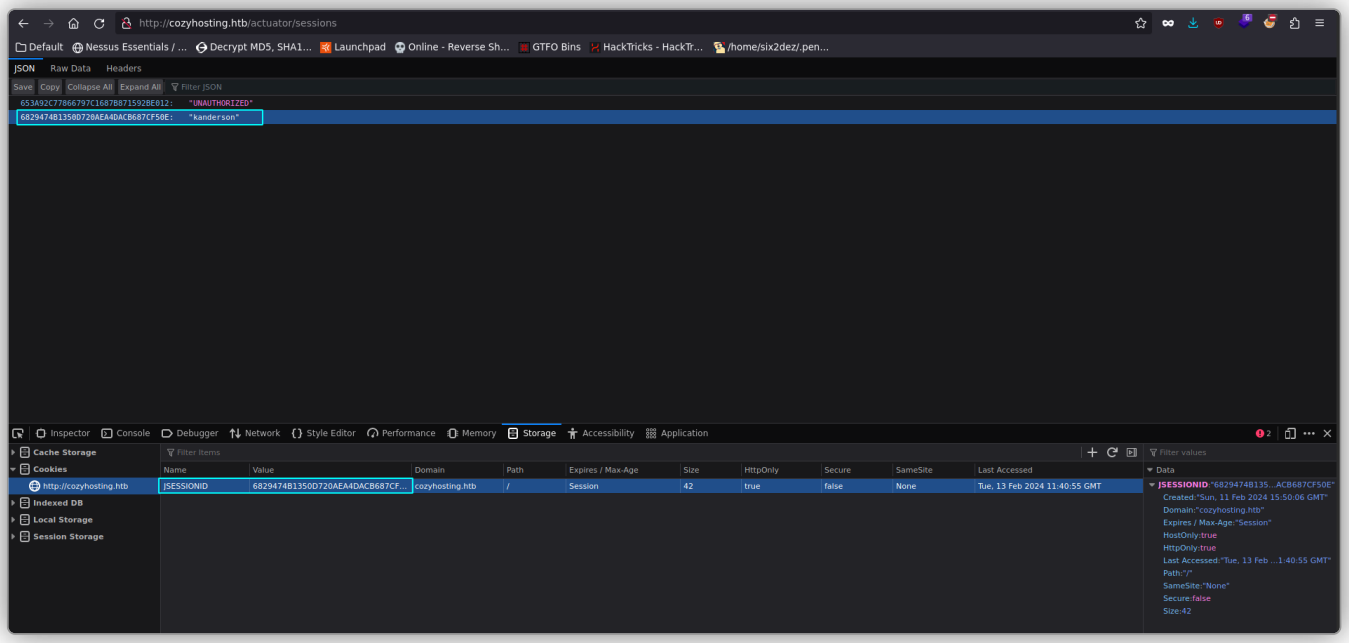
Por tanto, vamos a usar un diccionario de Seclists específico para este framework, el cual se encuentra en /usr/share/wordlists/SecLists/Discovery/Web-Content/spring-boot.txt. Encontramos ahora las siguientes rutas.



1.5. Leaked cookie session

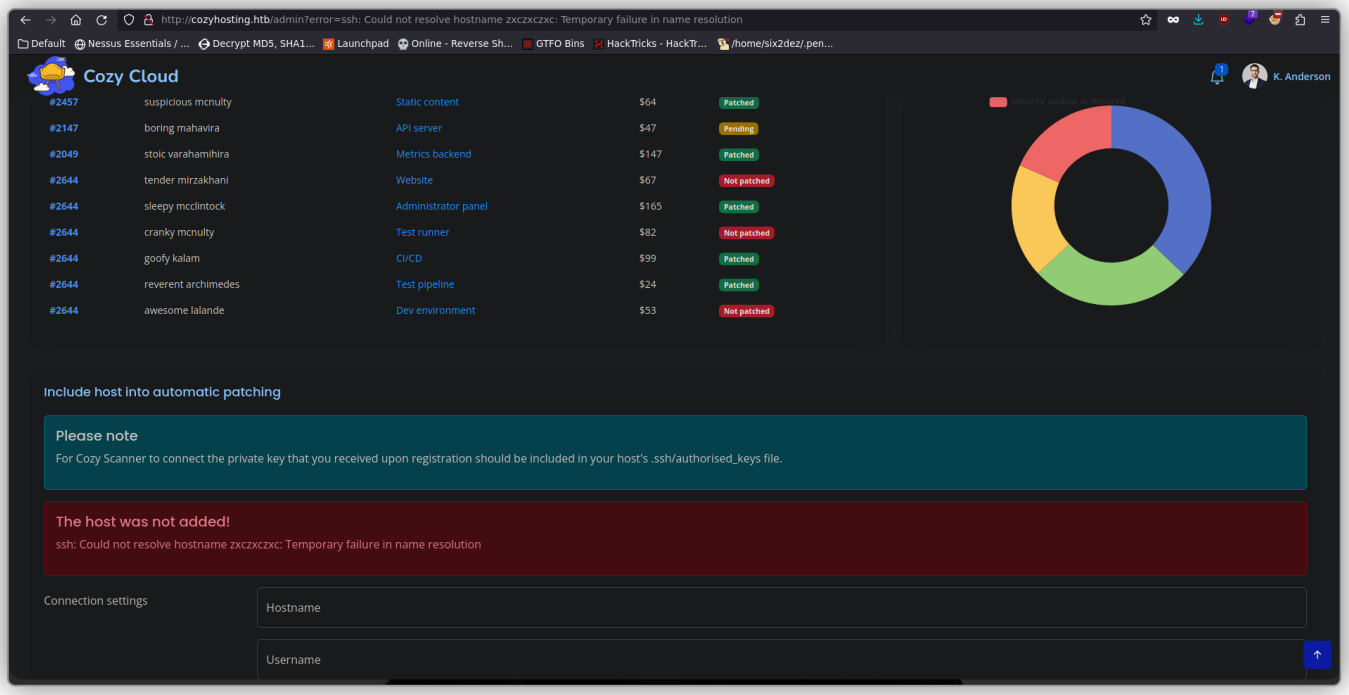
Visitamos estos directorios uno por uno, y parece que encontramos algo interesante en el directorio /actuator/sessions: un nombre de usuario y una posible cookie de sesión. Introducimos ésta en nuestro navegador y accedemos nuevamente a la

página de /login.



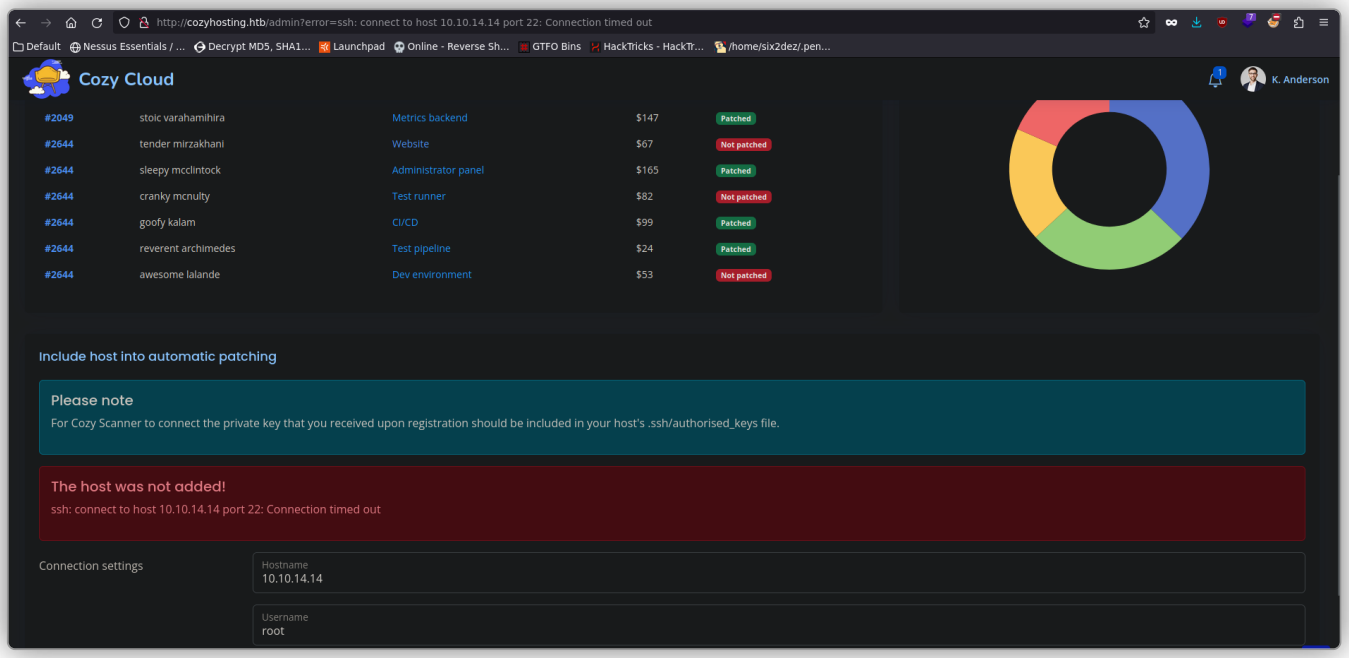
1.6. Command Injection coding blank spaces

Conseguimos acceso al panel /admin. Dentro de este panel, tenemos una opción de añadir un host para la aplicación. Hacemos una prueba, tratando de añadir un dominio que no exista. La aplicación no puede resolverlo.

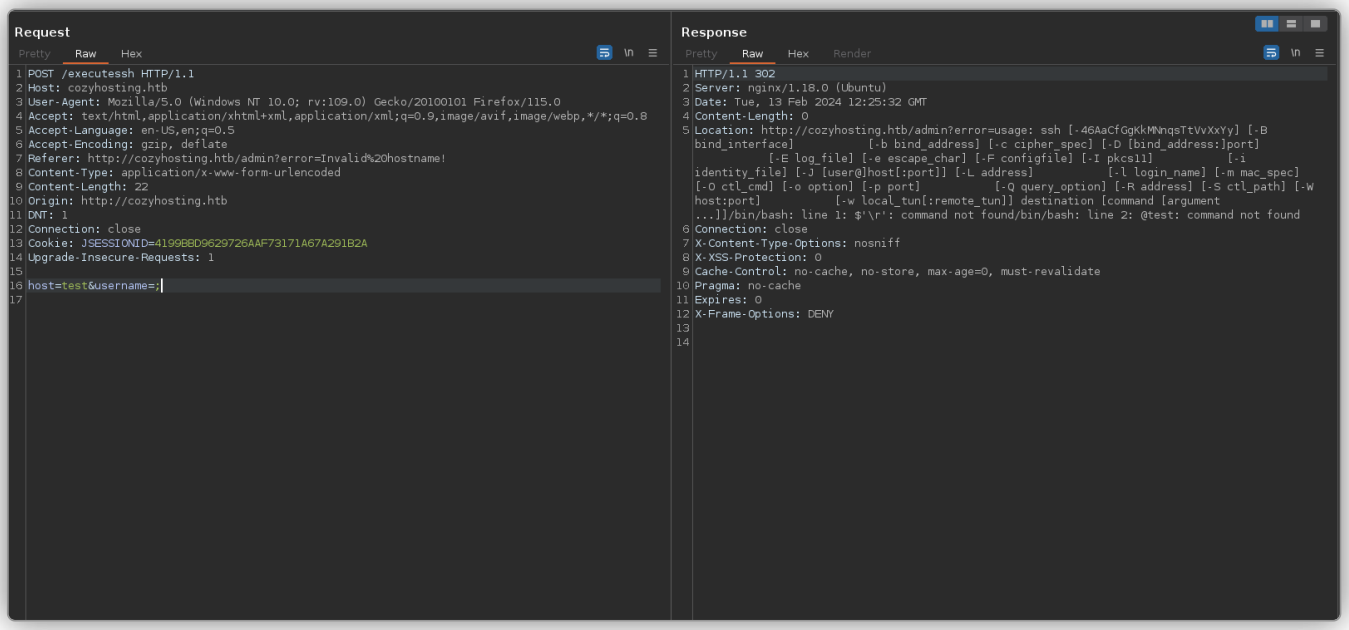


Cuando ponemos nuestra dirección, obtenemos este error. Pareciera que por detrás

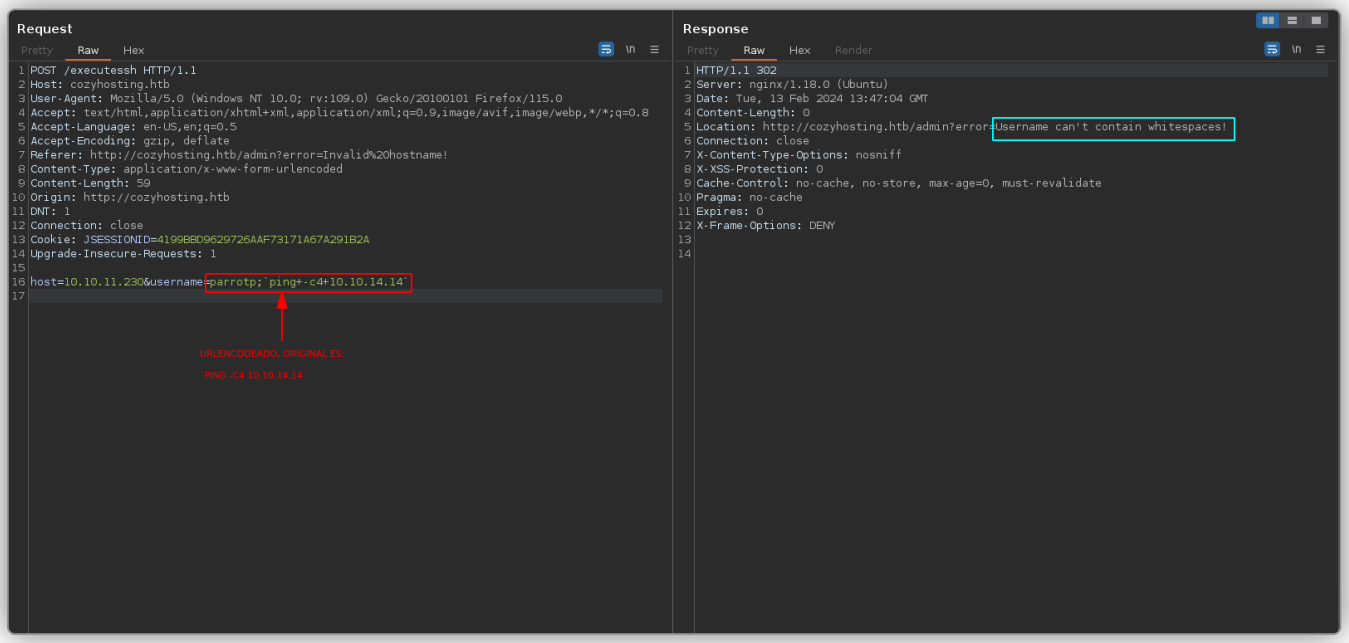
se tratara de conectar por SSH a nuestro sistema, lo que implicaría que, en cierto modo, se están concatenando comandos del sistema.



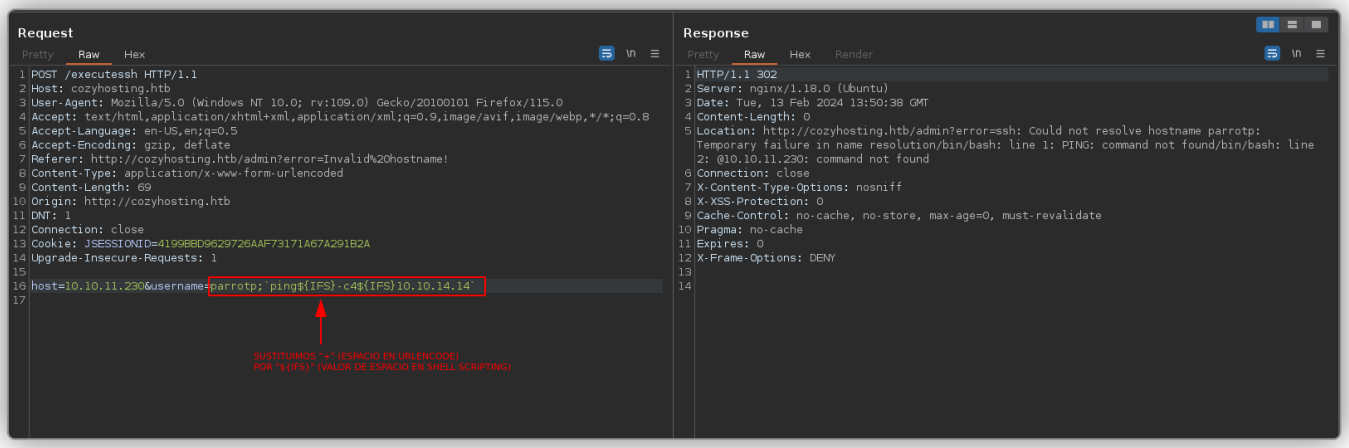
Por tanto, abrimos Burp Suite e interceptamos una petición para hacer pruebas. Dentro del campo *username*, introducimos `;`. Obtenemos el siguiente resultado: el panel de ayuda de SSH, por lo que sabemos que, por detrás, el sistema estaría ejecutando el comando `ssh (username)@(host)`.



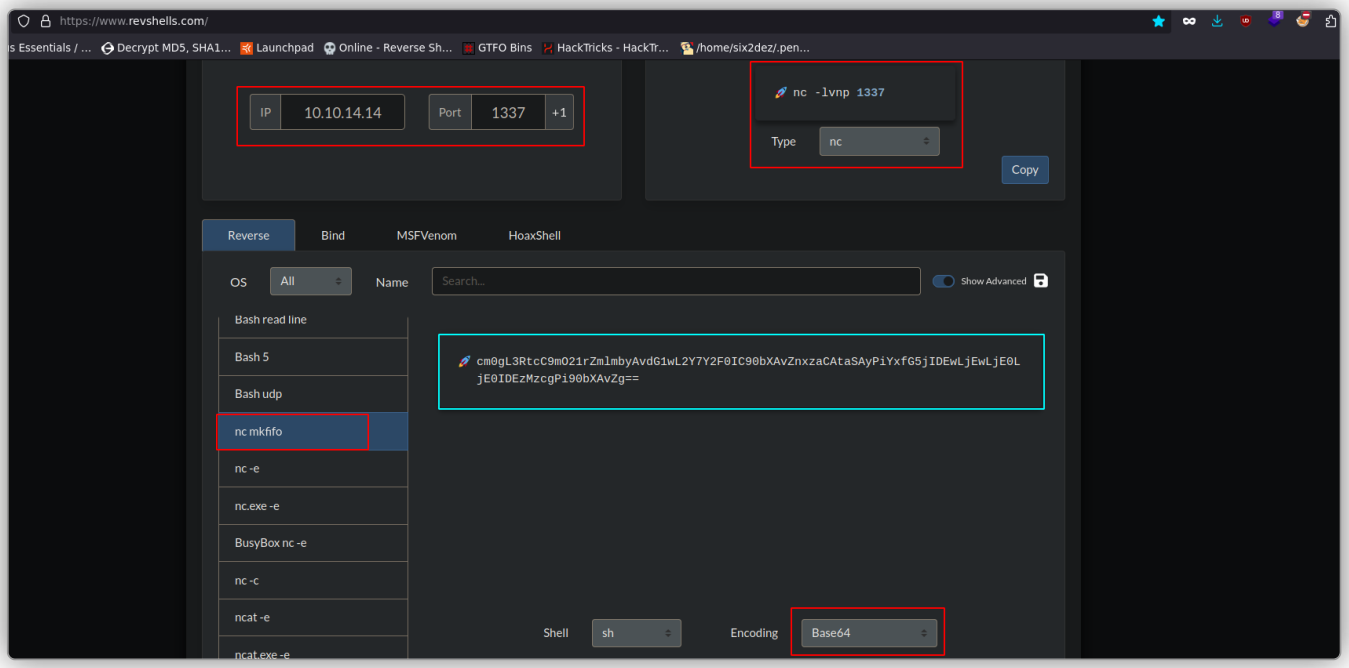
Sabiendo esto, podríamos tratar de inyectar comandos arbitrarios, como por ejemplo, el ping que aparece en la siguiente imagen. No obstante, para ejecutar los comandos, urlencodearemos los caracteres especiales. Por otro lado, en nuestra terminal, ejecutamos `tcpdump -i tun0 icmp` para comprobar si recibimos la traza ICMP. Al enviar este comando, obtenemos un error relacionado con los espacios en blanco.



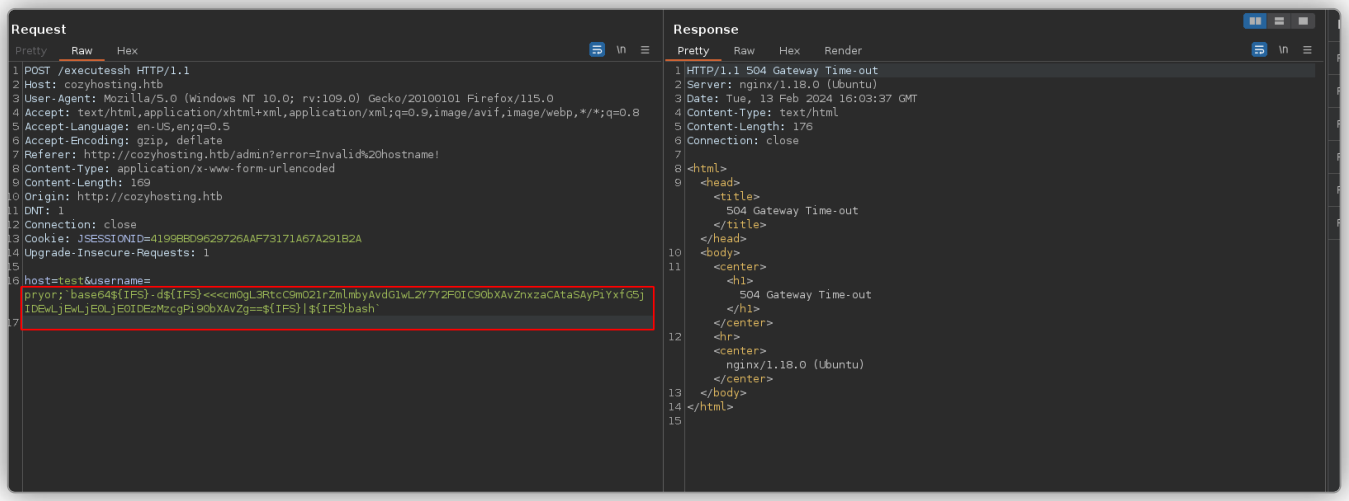
Así que usaremos un truco para codificar los espacios en blanco con `${IFS}`. Nuestra inyección de comandos, por tanto, quedaría tal y como aparece en la siguiente imagen. Asimismo, vemos cómo recibimos las trazas ICMP por nuestra interfaz que pusimos en escucha.



Podemos usar ahora www.revshells.com para generar un payload para obtener una shell reversa, la cual codificaremos en base64.



Copiamos este payload y lo pegamos en nuestra inyección de **Burp Suite**, precedido por `base64 -d` (para que se decodifique el payload y se pueda interpretar) y añadiendo los espacios `${IFS}` que fuesen necesarios. Adicionalmente, para este caso, estamos usando `<<<`, también llamado **redirección de aquí documentos**, la cual proporciona datos de entrada a un comando: `(comando) <<< (datos de entrada)`. Por último, se emplea `| bash` para ejecutar el comando. Podemos ver el payload completo en la siguiente imagen.

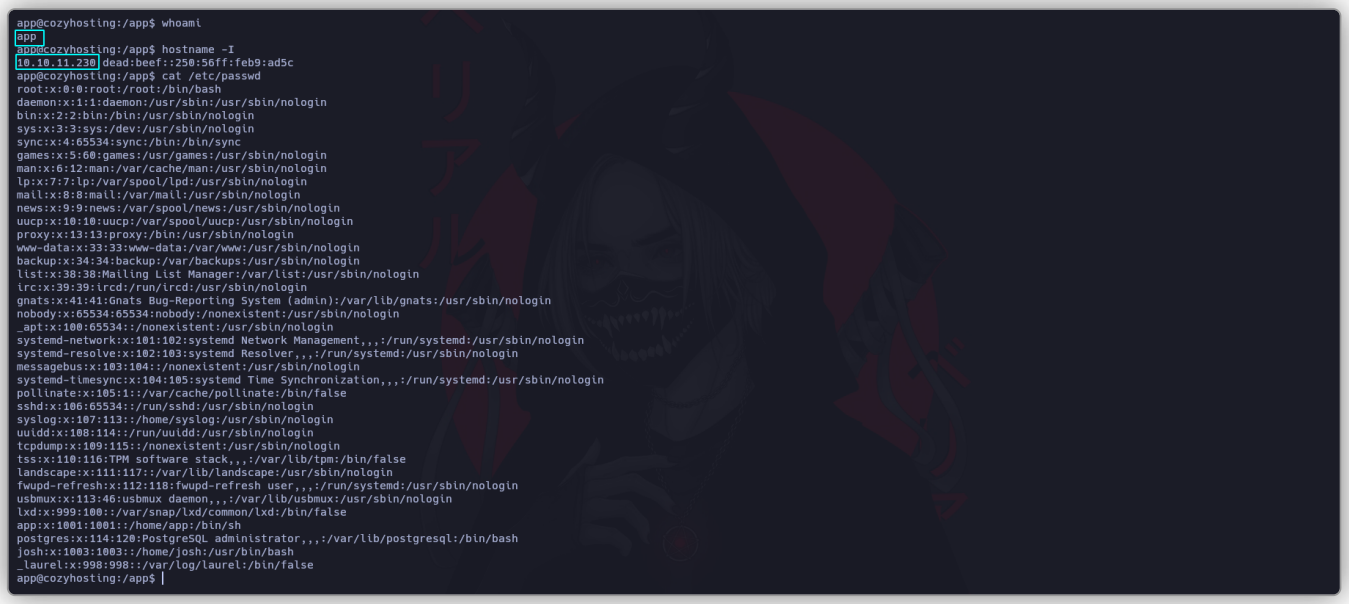


> `{IFS}`: generalmente se refiere, dentro de scripts de shell de Unix, al separador de campos interno **(Internal Field Separator)**. Este es un carácter especial que se utiliza para dividir cadenas en campos. Por defecto, el valor de `{IFS}` es el espacio en

> blanco, tabulador y nueva línea.
> `$`: en el contexto de inyección de comandos, se usa para representar una **expansión de parámetros** en la sintaxis de los scripts de shell. Por ejemplo, en un ataque de inyección de comandos en un script de shell, podrías intentar manipular la expansión de parámetros para introducir un espacio en blanco. En este caso, podrías usar la sintaxis `${IFS}` para representar el valor por defecto de `{IFS}`.

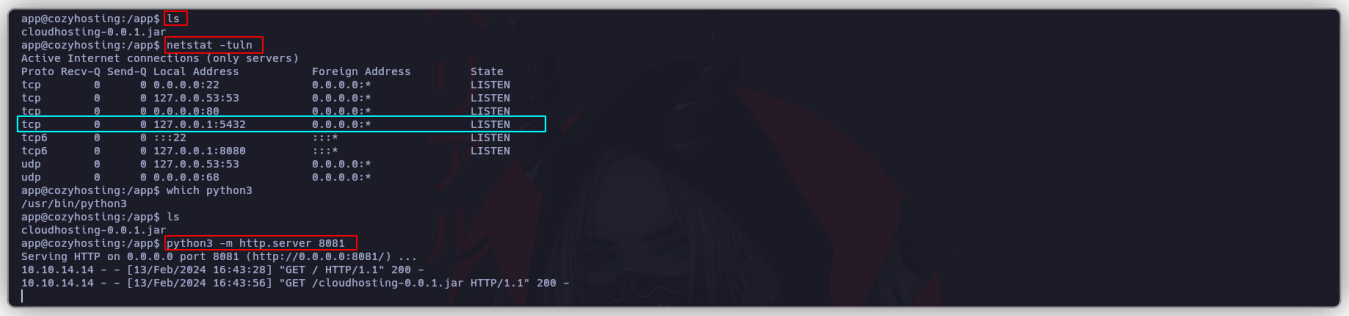
1.7. Reverse engineering JAR file

Tras ponernos en escucha por el *puerto 1337* y enviar nuestro payload, recibimos la shell reversa. Realizamos el **tratamiento de la TTY**. Estamos ahora como el usuario *app*. Hacemos un `cat /etc/passwd` para ver posibles usuarios a los que podamos hipotéticamente migrar la sesión. Intentamos ver qué permisos tenemos con `sudo -l`, pero necesitamos contraseña. Tratamos de listar archivos con **privilegios SUID** asignados, pero en principio, tampoco vemos nada.

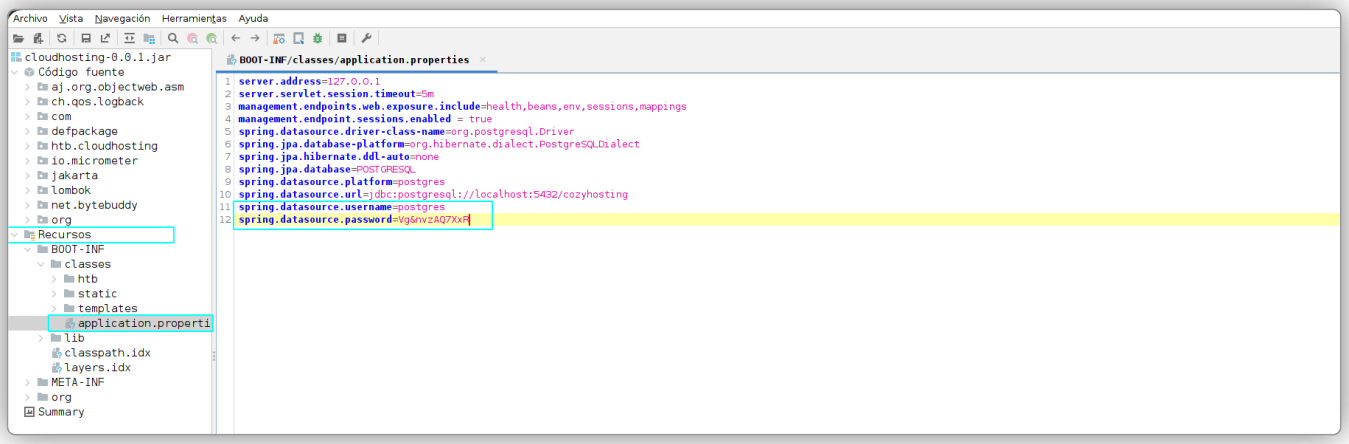


Enumeramos **los puertos internos** con `netstat -tuln`, y observamos que tenemos el *puerto 5432*, el cual es el puerto por defecto para el servicio de bases de datos de **PostgreSQL**. Por tanto, podríamos examinar esta aplicación para ver si podemos obtener algunas credenciales. Por otro lado, abrimos un servidor con Python desde la máquina víctima para poder descargar, desde nuestra máquina de atacante, este

archivo que encontramos llamado *cloudhosting-0.0.1.jar* con `wget` `http://cozyhosting.htb:8081/cloudhosting-0.0.1.jar`.



Teniendo *cloudhosting-0.0.1.jar* en nuestro sistema, vamos a proceder a analizarlo mediante **Jadx-gui**. Para ello, usamos `jadx-gui` en la terminal y cargamos nuestro archivo. Investigando un poco la aplicación, encontramos unas credenciales de un usuario para la base de datos **PostgreSQL**.



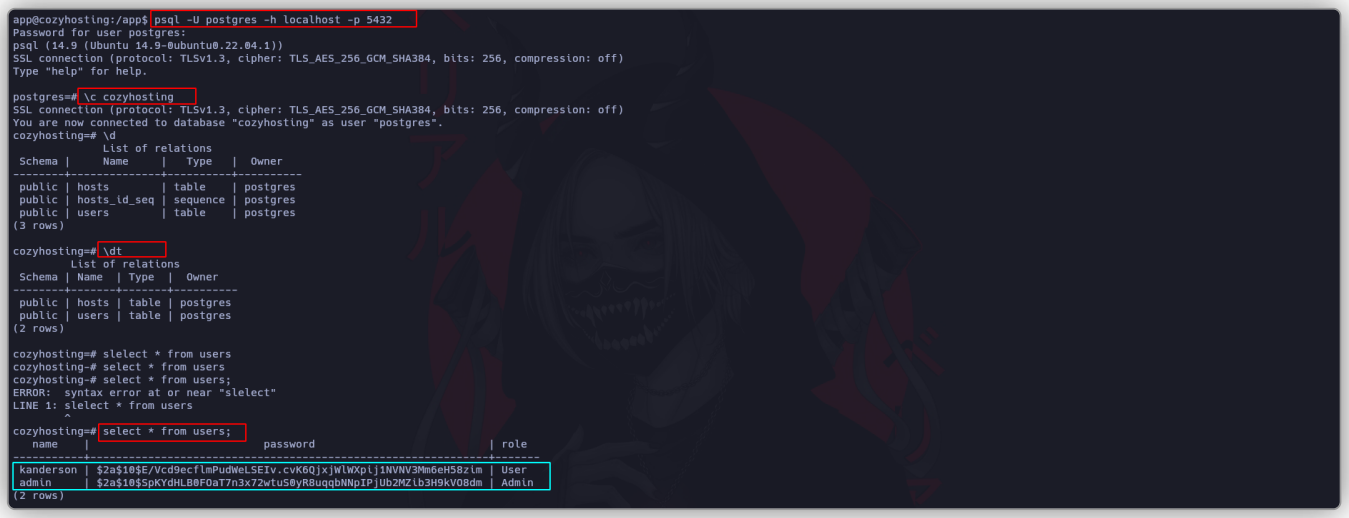
> Un archivo **JAR (Java ARchive)** es un archivo que se utiliza para almacenar y distribuir una o varias aplicaciones o bibliotecas Java. Esencialmente, un archivo JAR es una forma de empaquetar archivos Java y recursos relacionados en un solo archivo comprimido, similar a un archivo ZIP.

> **Jadx-gui** es una interfaz gráfica de usuario (GUI) para **jadx**, una herramienta de código abierto para abrir y descompilar archivos *JAR* y archivos *APK*, ya que ambos son archivos **Java**. Entonces, aunque jadx es conocido por su uso en el contexto de

> aplicaciones de Android, jadx-gui puede ser utilizado para descompilar archivos JAR estándar también. Jadx-gui proporciona una forma más amigable y fácil de usar jadx, permitiendo a los usuarios navegar por el código fuente de las aplicaciones de Android de manera más intuitiva. Con jadx-gui, los usuarios pueden cargar archivos APK, explorar su estructura, ver el código fuente descompilado y realizar búsquedas dentro del código.

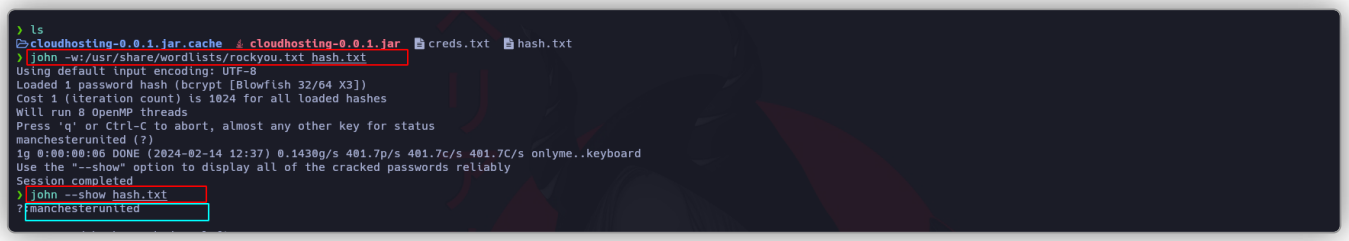
1.8. Credentials in PostgreSQL

Entramos a la base de datos con `psql -U postgres -h localhost -p 5432` y proporcionando la contraseña encontrada. Investigamos un poco las diferentes tablas, hasta que conseguimos dumpear las **contraseñas hasheadas** para los usuarios *kanderson* y *admin*, tal y como se muestra en la siguiente imagen.

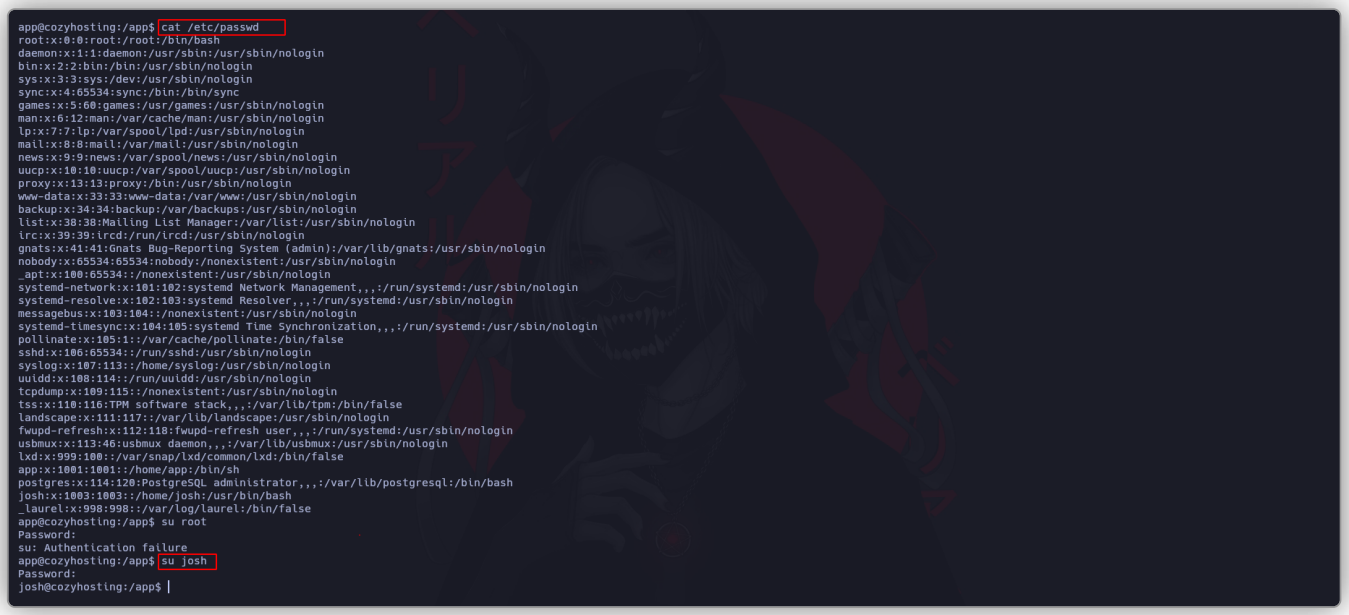


1.9. Password cracking with John

Usamos **John the Ripper** para crackear la contraseña: `john -w:/usr/share/wordlists/rockyou.txt hash.txt`. Al cabo de unos minutos, lo conseguimos. La contraseña para el usuario *admin* es: *manchesterunited*,



Tratamos de usar estas credenciales para conectarnos por SSH con diferentes usuarios, tales como admin, root, etc. pero no lo conseguimos. También vemos los diferentes usuarios del /etc/passwd y los probamos, hasta que, finalmente, descubrimos que la contraseña pertenece al usuario *josh*.



1.10. Privesc via sudo SSH

Lo primero que hacemos al tener la sesión como *josh* es comprobar los privilegios de sudo con `sudo -l`. Podemos ejecutar como root el binario de SSH. Por tanto, vamos a GTFObins y vemos que tenemos una vía potencial de escalar privilegios ejecutando este comando: `sudo ssh -o ProxyCommand=';sh 0<&2 1>&2' x`. Obtenemos nuestra

sesión como root.