

Hackeando el aprendizaje

OverTheWire como puerta de entrada a Linux

1. Introducción

- ¿Qué es OverTheWire y por qué es útil?

OverTheWire es una plataforma en línea de juegos de guerra (wargames) de ciberseguridad que ofrece una serie de desafíos para aprender y practicar habilidades de hacking ético. Diseñada para ayudar a los usuarios a mejorar sus habilidades en Linux y ciberseguridad mediante ejercicios prácticos en la terminal. Es un tipo de CTF o (Capture the flag) .

- Objetivo de la guía.

El objetivo de esta guía es conocer los diferentes comandos que se usan habitualmente en Linux aplicado a Ciberseguridad a medida que subimos de nivel.

- Cómo usarla

Si en algún momento necesitas recordar el funcionamiento de un comando o su aplicación en ciberseguridad, esta guía te servirá como referencia rápida.

2. Conceptos Básicos

- Uso de SSH: SSH o (Secure Shell) es un protocolo que se utiliza para establecer conexiones de forma segura a servidores remotos o a máquinas. Este protocolo cifra los datos entre el cliente y el servidor. Logrando así una forma segura de gestionar sistemas remotos.
- Directorios: Linux organiza su sistema de la siguiente manera: / es la raíz, /home contiene los directorios personales de usuario, /etc contiene los archivos de configuración, . representa el directorio actual y .. representa el directorio anterior.
- Comandos esenciales: ls que lista los archivos en el directorio actual , cd que nos permite cambiar de directorio, pwd que muestra la ruta del directorio actual, cat que muestra el contenido de un archivo, file que indica el tipo de archivo y find que busca archivos y directorios del sistema.
- Bash: Nos permite automatizar procesos.
- Permisos: r(read) es de lectura, w(write) es escritura y x(execute) es de ejecución.

3. Solución de Niveles

Nivel 0: Conectarse al servidor por SSH

- *Objetivo del nivel*

El objetivo de este nivel es aprender a conectarse al servidor de OverTheWire mediante el protocolo SSH. Para ello, nos proporcionan un host `bandit.labs.overthewire.org`, un puerto 2220, un usuario `bandit0` y una contraseña, que es la misma contraseña del usuario.

The goal of this level is for you to log into the game using SSH. The host to which you need to connect is bandit.labs.overthewire.org, on port 2220. The username is `bandit0` and the password is `bandit0`. Once logged in, go to the Level 1 page to find out how to beat Level 1.

- **Comandos utilizados**

El comando que requerimos para este nivel es SSH.

```
ssh bandit0@bandit.labs.overthewire.org -p 2220
```

- **Explicación paso a paso**

Normalmente, el protocolo SSH usa el puerto 22 para establecer conexiones. Sin embargo, en OverTheWire Bandit, el servidor está configurado para usar el puerto 2220, por lo que debemos especificarlo al conectarnos. La estructura general para realizar conexiones mediante SSH, es "ssh usuario@nombre_del_host -p puerto ". Usamos el parámetro " -p " para especificar el puerto al que queremos realizar la conexión. en caso que no sea el puerto 22 .

Ahora nos piden la contraseña que nos suministraron, es decir, bandit0.

Logramos entrar a bandit0. Ahora seguimos con el siguiente nivel.

```
--[ More information ]--
```

```
For more information regarding individual wargames, visit  
http://www.overthewire.org/wargames/
```

```
For support, questions or comments, contact us on discord or IRC.
```

```
Enjoy your stay!
```

```
bandit0@bandit:~$
```

Nivel 1: Encontrando readme

- *Objetivo del nivel*

Se nos pide localizar un archivo llamado " readme " en el directorio home que contiene la contraseña o la flag para acceder al siguiente nivel.

```
The password for the next level is stored in a file called readme located in the home directory. Use this password to log into bandit1 using SSH. Whenever you find a password for a level, use SSH (on port 2220) to log into that level and continue the game.
```

```
Commands you may need to solve this level
```

```
ls, cd, cat, file, du, find
```

- ** _Comandos utilizados

Los comandos que utilizaremos para este nivel son:

1. **ls** : Nos ayuda a listar el contenido, los archivos y directorios del sistema.
2. **cat**: Nos muestra el contenido de un archivo.

- ***Explicación paso a paso**

Usamos " **ls** " para listar el contenido del directorio en donde nos encontramos.

```
bandit0@bandit:~$ ls  
readme
```

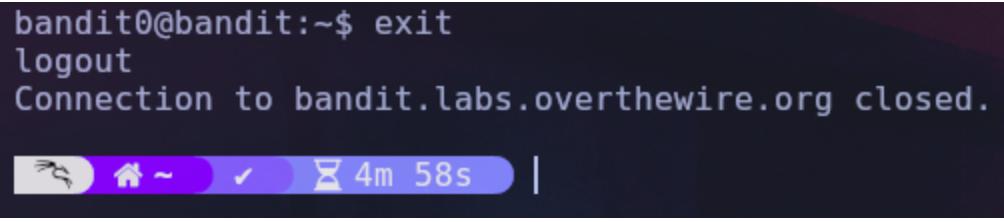
Ahora que ya ubicamos el archivo " **readme** " , usamos un " **cat** " para ver el contenido de ese archivo. Y ya tendríamos la flag.

```
bandit0@bandit:~$ cat readme
Congratulations on your first steps into the bandit game!!
Please make sure you have read the rules at https://overthewire.org/rules/
If you are following a course, workshop, walkthrough or other educational activity,
please inform the instructor about the rules as well and encourage them to
contribute to the OverTheWire community so we can keep these games free!

The password you are looking for is: ZjLjTmM6FvvyRnrb2rfNW0ZOTa6ip5If
```

Ahora que ya tenemos la flag para conectarnos a bandit1, podemos usar el comando "exit" para salir de "bandit0" y ir a nuestro directorio actual de trabajo.

```
bandit0@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
```



Procedemos a volvemos a conectar mediante ssh pero esta vez mediante el usuario "bandit1", y como contraseña usamos esa flag.

```
› ssh bandit1@bandit.labs.overthewire.org -p 2220
```

Nivel 2: Accediendo a -

- *Objetivo del nivel*

Aquí el reto es acceder a un archivo llamado " - ", lo cual puede ser complicado porque el guion " - " es interpretado como un argumento en comandos Linux. Esta ubicado en el directorio home.

- **_Comandos utilizados
 1. **cat**: Nos muestra el contenido de un archivo.
 2. **pwd**: Muestra en la terminal el directorio actual de trabajo.

- ***Explicación paso a paso**

Este es el archivo:

```
bandit1@bandit:~$ ls
-
```

Normalmente pensariamos en hacer un " cat - ", pero esto no funciona debido a que la terminal lo interpreta como un argumento y se quedaría en modo de espera o de escucha. Esperando que le pasemos un parámetro.

```
bandit1@bandit:~$ cat -
```

Cuando tenemos este tipo de archivos podemos leerlos de dos maneras:

1. Usando la ruta en la que se encuentra el archivo, si usamos pwd podemos ver el directorio en el que nos encontramos, a la vez que vemos en donde se encuentra el archivo.

```
bandit1@bandit:~$ pwd  
/home/bandit1  
bandit1@bandit:~$ ls  
-
```

El cat nos permite leer un archivo mediante la ruta en la que se encuentra ese mismo archivo. De la siguiente manera:

```
bandit1@bandit:~$ cat /home/bandit1/-  
263JGJPfgU6LtdEvgfWU1XP5yac29mFx
```

2. También podemos hacer lo siguiente: Partiendo del directorio actual de trabajo, mostrar el contenido de un archivo " - " que reside en el directorio actual de trabajo usando " ./ "

```
.
```

```
bandit1@bandit:~$ cat ./-  
263JGJPfgU6LtdEvgfWU1XP5yac29mFx
```

```
> ssh bandit2@bandit.labs.overthewire.org -p 2220
```



```
This is an OverTheWire game server.  
More information on http://www.overthewire.org/wargames
```

```
bandit2@bandit.labs.overthewire.org's password:
```

**Nivel 3: Spaces in this filename

- *Objetivo del nivel*

Nos piden mostrar el contenido de un archivo llamado " spaces in this filename " , ubicado en el directorio home. El archivo tiene espacios, entonces no podemos simplemente usar " cat spaces in this filename " porque la terminal lo interpretara como varios argumentos separados.

- **_Comandos utilizados

1. cat: Nos muestra el contenido de un archivo.

- *Explicación paso a paso

Como ya mencionamos anteriormente, si hacemos un " cat spaces in this filename " , la terminal lo interpretara como varios argumentos.

```
bandit2@bandit:~$ cat spaces in this filename
cat: spaces: No such file or directory
cat: in: No such file or directory
cat: this: No such file or directory
cat: filename: No such file or directory
bandit2@bandit:~$
```

Para acceder a la flag , tenemos varias maneras:

1. Para acceder a archivos que tengan espacios , hacemos un cat "nombre del archivo " , es decir, usamos paréntesis.

```
bandit2@bandit:~$ cat "spaces in this filename"
MNk8KNH3Usilio41PRUEoDFPqfxLPlSmx
```

2. O también podemos dividir por

```
bandit2@bandit:~$ cat spaces\ in\ this\ filename
MNk8KNH3Usilio41PRUEoDFPqfxLPlSmx
```

3. O mucho mas cómodo, decirle a la terminal que queremos mostrar el contenido de un archivo que empiece por " s " y que tiene una cadena que desconocemos.

```
bandit2@bandit:~$ cat s*
MNk8KNH3Usilio41PRUEoDFPqfxLPlSmx
```

**Nivel 4: Archivos ocultos

- *Objetivo del nivel*

La siguiente flag esta almacenada en un archivo oculto, dentro del directorio " inhere " .

Debido a esto, tendremos que utilizar otros comandos y trucos.

- **_Comandos utilizados

1. ls: Nos ayuda a listar el contenido, los archivos y directorios del sistema.
2. cd: Nos permite navegar entre directorios.
3. file: Nos permite saber el tipo de archivo al que nos estamos enfrentando.
4. cat: Nos muestra el contenido de un archivo.
5. find: Nos ayuda a desplegar y filtrar todos los archivos y directorios.

6. xargs: Toma la salida de otro comando o del comando anterior y la usa como argumento para otro comando. Es decir, analiza la salida de uno de los comandos y aplica los resultados en el segundo.
7. grep: Busca cadenas de texto dentro de archivos.

- ***Explicación paso a paso**

Listamos el contenido del directorio actual de trabajo

```
bandit3@bandit:~$ ls  
inhere
```

Se nos dice que " inhere " es un directorio, entonces, usamos el comando " cd " para entrar a ese directorio.

```
bandit3@bandit:~$ cd inhere/
```

Una vez dentro del directorio " inhere " , y si listamos su contenido con " ls " podemos ver que no hay nada en la salida. Esto porque el archivo esta oculto.

```
bandit3@bandit:~/inhere$ ls  
bandit3@bandit:~/inhere$ |
```

Entonces, podemos usar " ls -la " , el parámetro "-la" nos permite listar archivos ocultos.

```
bandit3@bandit:~/inhere$ ls -la  
total 12  
drwxr-xr-x 2 root      root      4096 Apr 10 14:23 .  
drwxr-xr-x 3 root      root      4096 Apr 10 14:23 ..  
-rw-r----- 1 bandit4 bandit3    33 Apr 10 14:23 ...Hiding-From-You
```

En la imagen presentada anteriormente, podemos ver dos directorios ocultos, y un archivo llamado "...Hiding-From-You" .

Podemos hacer un " file " para saber mas cosas de ese archivo

```
bandit3@bandit:~/inhere$ file ...Hiding-From-You  
...Hiding-From-You: ASCII text
```

Es un archivo que contiene ASCII text. Hacemos un " cat " y ya tendríamos la flag:

```
bandit3@bandit:~/inhere$ cat ...Hiding-From-You  
2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ
```

2. Una segunda forma en la que podemos hallar la flag, es mediante el comando " find " . Entonces si filtramos por archivos, nos va a arrojar archivos, y de forma paralela con el comando "grep" le pedimos que filtre por " Hiding " y luego con el comando " xargs " hacemos un cat.

```
bandit3@bandit:~/inhere$ find . -type f | grep "Hiding" | xargs cat  
2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ
```

Funciona de la siguiente manera: " find . -type f " , usamos este comando para buscar archivos dentro del directorio " inhere " . Vamos a desglosarlo

- `find` permite localizar archivos y directorios del sistema
- El punto `" . "` indica que la búsqueda debe iniciarse en el directorio actual de trabajo, si por ejemplo quisiéramos buscar en otro directorio, en vez del punto tendríamos que colocar la ruta específica del archivo.
- `" - type f "` especificamos que solo queremos buscar archivos, `" f "` hace referencia a `" file "`, si el `" f "` el comando mostraría los directorios.

Luego, filtramos solo los archivos que empiecen por `" Hiding "` que es la primera cadena de texto del archivo en donde se encuentra la flag. Toma ese archivo filtrado para después hacerle un `" cat "` con `" xargs "`.

**Nivel 5: Encuentra el texto oculto

- ***Objetivo del nivel***

Se nos pide encontrar el único archivo que es legible para los humanos dentro del directorio `inhere`. Sucede que algunos archivos pueden ser binarios o pueden tener contenido ilegible.

- **Comandos utilizados**

1. `ls`: Nos ayuda a listar el contenido, los archivos y directorios del sistema.
2. `file`: Nos permite saber el tipo de archivo al que nos estamos enfrentando.
3. `xargs`: Toma la salida de otro comando o del comando anterior y la usa como argumento para otro comando. Es decir, analiza la salida de uno de los comandos y aplica los resultados en el segundo.
4. `cat`: Nos muestra el contenido de un archivo.
5. `grep`: Busca cadenas de texto dentro de archivos.
6. `find`: Nos ayuda a desplegar/filtrar todos los archivos y directorios.

- ***Explicación paso a paso**

Hacemos un `ls`, nos metemos al directorio `inhere`, volvemos a hacer `" ls "` y podemos ver los siguientes archivos.

```
bandit4@bandit:~/inhere$ ls
-file00  -file01  -file02  -file03  -file04  -file05  -file06  -file07  -file08  -file09
```

Cuando nos enfrentemos a varios archivos ocultos podemos listarlos y ver el tipo de contenido en un solo comando, podemos hacer un `" xargs file "` para saber el tipo de archivo.

```
bandit4@bandit:~/inhere$ find . -type f | grep "\-file" | xargs file
./-file05: data
./-file03: data
./-file06: data
./-file02: data
./-file01: data
./-file09: data
./-file00: PGP Secret Sub-key -
./-file04: data
./-file08: data
./-file07: ASCII text
```

Dentro del "grep" se pone un "\ " porque sin el, el comando grep esperara un argumento después del " - ". Podemos ver que el unico archivo que es legible es numero siete, el de tipo ASCII text.

Dado que cada archivo tiene un guion, no podemos listar su contenido solo con un cat.

```
bandit4@bandit:~/inhere$ cat -file07
cat: invalid option -- 'f'
Try 'cat --help' for more information.
bandit4@bandit:~/inhere$ |
```

Hacemos lo mismo que antes, usamos "./"

```
bandit4@bandit:~/inhere$ cat ./-file07
4oQYVPkxZ00E005pTW81FB8j8lxXGUQw
bandit4@bandit:~/inhere$
```

**Nivel 6: Filtrando por tamaño y permisos

- *Objetivo del nivel*

Debemos encontrar dentro del directorio " inhere " un archivo con las siguientes propiedades: que sea humanamente legible, sea de tamaño 1033 bytes y que no sea ejecutable.

- **_Comandos utilizados

1. find
2. xargs
3. cat

- ***Explicación paso a paso**

Podemos usar el comando " find " . Con los siguientes parámetros: " -readable " significa que estamos buscando archivos con capacidad de lectura, el " ! -executable " nos ayuda a buscar archivos que no sea ejecutables, negamos el parámetro " executable " con un " ! " . Para después filtrar por tamaño " -size 1033c " , " c " es bytes para la

terminal.

```
bandit5@bandit:~/inhere$ find . -type f -readable ! -executable -size 1033c  
./maybehere07/.file2  
bandit5@bandit:~/inhere$ find . -type f -readable ! -executable -size 1033c | xargs cat  
HWasnPhtq9AVKe0dmk45nxy20cvUa6EG
```

**Nivel 7: Encuentra el archivo perdido

- *Objetivo del nivel*

Debemos encontrar un archivo en algún lugar del servidor, y tiene las siguientes propiedades: es propiedad del usuario " bandit7 " y del grupo " bandit6 " , además de tener 33 bytes de tamaño.

- ** _Comandos utilizados
 1. find
 2. xargs
 3. cat
 4. 2>/dev/null: Redirige los errores al /dev/null . Este ultimo es un recurso existente que hace desaparecer cualquier archivo que le mandemos. Pero el comando si se esta ejecutando.

- *Explicación paso a paso

Podemos ver en la imagen que no hay nada, cuando tengamos un archivo que esta oculto en algún lado, tenemos que buscar desde la raíz.

```
bandit6@bandit:~$ ls -l  
total 0  
bandit6@bandit:~$ ls -la  
total 20  
drwxr-xr-x 2 root root 4096 Apr 10 14:22 .  
drwxr-xr-x 70 root root 4096 Apr 10 14:24 ..  
-rw-r--r-- 1 root root 220 Mar 31 2024 .bash_logout  
-rw-r--r-- 1 root root 3771 Mar 31 2024 .bashrc  
-rw-r--r-- 1 root root 807 Mar 31 2024 .profile
```

Para ir a la raíz podemos hacer un " cat / " , el " / " representa la raíz.

```
bandit6@bandit:$ cd /
```

Le decimos a la terminal que queremos buscar archivos donde el usuario sea " bandit7 " y que el grupo asignado sea " bandit6 " . Mediante los parámetros " -user " y " -group "

```
bandit6@bandit:$ find . -type f -user bandit7 -group bandit6 -size 33c 2>/dev/null  
.var/lib/dpkg/info/bandit7.password  
bandit6@bandit:$ find . -type f -user bandit7 -group bandit6 -size 33c 2>/dev/null | xargs cat  
morbNTDkSW6jILUc0ym0dMaLn0lFVAaj
```

Los errores en Linux se definen como " stderr ", y se puede referenciar como el numero . Una manera de decirle a la terminal que no queremos ver los errores es mediante el siguiente comando: " 2>/dev/null " . En caso de que no queramos ver las salidas de los comandos lo que podemos hacer es " &>/dev/null " .

****Nivel 8: Buscando "millionth" en data.txt**

- *Objetivo del nivel*

Se nos solicita buscar una palabra especifica dentro del archivo llamado data.txt.

- **_Comandos utilizados**

1. awk 'NF{print \$NF}' : Este comando nos permite quedarnos con el ultimo argumento de una linea o de un argumento.
2. grep
3. cat

- ***Explicación paso a paso**

Si le hacemos un cat al archivo tendremos miles de cadenas de texto, como se ve en la imagen.

deformities	sIBU0A0au2Iwicvvv8v6Yi7ySzcT95LK
breasted	rqWoZM6G7xqIGvk15bl7v7kCs9x9P0PF
pressure	nJbkKfVEAgqDQ54oRp7DPufZGqcg9su
motoring's	XL21vzySn4XiHTXMd4WGJeBfARb0SUPh
Ludwig's	z6fI06wvj85FRHqckDAAGr8n7E7klSv0
slowdowns	vLWaiM6K4TIrFpjlpjr6WME2thubF408
resplendence's	rp0GatFtcPUibCJWQrV6kL3kixwfRL1e
cortices	j7SHBuFXeSy7Cfk7cXykwcAD63xP8YHi
warranties	9JJlpbIj1Hymyw9RKxPvigInHpJEp0t1
inexpensively	FIBUGtDRuqAsHWAFa86eLknnPb9llKJw
Carranza's	XhRzIJ2p5RcbVrvnJGwS88SjebVevHg3
Diana's	jghQzrV6SSzcdRyKOXPaozm7HvvGSoxx
trusteeship's	nL0QTGaG1z4o061zkdVXC6GlCTJRGPg
hardwoods	SR7FimZLjm9tH8pmKzFdXtM8whGZN6dp
goldsmith's	CrXR10qN407M0soFsY4f4V2Crb0W5bYn
paired	Smaa0tgolrvpzd6DAII3pmLTeUBRLK14
Becky	NpPZj10YboeQzrPFSSIMHuW6SLDXNPgw
columnists	PU7f5XF5sn47kvIy3DOAWwxlzE4UPPof
avocations	FA877P0tbmQ2kbINOGSxaBI1JuXFkbyK
foolery's	nLXn7DjEF5FM9JokQVHKopbo0yxpnlMv
papyrus	MgIBiJSnVBcApEI04KAoV6PlIt7GcFmp
palace's	3P3NTvuCAgizOFPEL6BwqnbU6l3Esole
attiring	Phq40txF8gzbPKg9mfjygBGGFaiUKYUW
bogeyman's	7tA8l0QUUcwQYSpEePqF04zgJ0Ica991
plaid	ochCeaKS7xxVEf1xdTw0E5bfs8WrrrFx
crinoline's	7m7MqeWz0EPlyyRzKSkkWvB8Cq0SWDHF
lovely	blteD0eGjobqhKcZKXChSXB0PFBbyxc2
hounding	quXySkWoe6hDLiIsmZsqiCbdwYWxjzXX
worriers	AyRiV6aruz2YuxCEFLLeumeiWQCS83KK6
whens	zL6LbzIsVx8Vx6HbdJeWd4soLQfAiEda
betray	iGLlyYYf6ZY0chupwUgQc6C4n5XXESTr
nakedness	2jyweZvd0g0lWi7Ua0llS1z2lwoY0gpr
doghouse's	BJ6LehVqSdL47EfZJ6V7PxbSCFKasXqn
mitigates	C09l8FpGWYywQhKYZeMViASVnQgtrLRm
donor's	TRFc1w69NkypAViQhDYGv3wNDx3jewG9
kitchening	DXIVn1Qnd9qjLD5bISLRIHS4WnaYeMko
reclining	6GyvNuAGX1HsImFYeA106ySrzb0dwFcb
achoo	uZoJJJIm4xxaQTQKTTjpVH6q0goWRP6W
detonators	0WF07wP9ZAnjSoS80AP5tkgtZXPs6nZ0
steeple's	wYdpBlfQnvMCtt2BwVkpMUPkAlYUikv0

Pero nos dicen que tenemos que buscar una cadena específica de texto, concretamente "`millionth`" .

1. Podemos usar "`cat`" y greppear por `millionth`

```
bandit7@bandit:~$ cat data.txt | grep "millionth"
millionth      dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc
bandit7@bandit:~$ |
```

2. Podemos hacer `awk 'NF{print $NF}'` .

```
bandit7@bandit:~$ cat data.txt | grep "millionth" | awk 'NF{print $NF}'  
dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc
```

3. O de otra manera: Decirle que nos muestre el segundo argumento.

```
bandit7@bandit:~$ cat data.txt | grep "millionth" | awk 'NF{print $2}'  
dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc
```

**Nivel 9: Identifica una linea única

- *Objetivo del nivel*

Se nos pide encontrar la única linea de texto que aparece una sola vez dentro del archivo `data.txt`.

- ** _Comandos utilizados

1. `sort` : Con este comando lo que hacemos es listar la salida en orden alfabético para que se vea mas organizado
2. `uniq -u` : Podemos listar líneas que no se repiten o líneas únicas.

- *Explicación paso a paso

Si hacemos un `cat data.txt`, en la salida tendremos bastantes cadenas de texto:

```
YLbWtoCeR7TRMhSiwGDk7xocg1UIIFIy
DEgKKTHJK83mIJgDaxAZNJ5KZ0JB1ft0
qb1EjP5NsHPJhx1jQilWY977tkCqMfq5
PXbEGNSaB8ReZ0rjbboxv4GPJZSF8v0QT
mEM65MuAhYISobkugdEBhSVg87syAp00
kksTcfsR8py5YBRxDKPYTM9tAer6YaM9
8xgLhGPgRmIV7nXsh0BdMv7cIZEjG9hr
xWDdYK8PVu6m056eatb9j1vDuXIixIpx
r30J9XDT3s0tZ0Bv8EUUypHixRED0skL
DhvgXMoEf0c49M7M6hl8Mdo80quArQXn
pZX0E5uS8oJx5019kaKFKRhww2f9XrsS
5EL94fXpDzA3o08q2IFwAQ7Wwd0BnUz2
JoD4RBiS2kB4KUTpbszENo7mNYeSfxdG
L3ZCH71RRxt8Kmy3X3R0NqQTmebcmkQ4
RKnUMxdc5dktmV027DgidzidB97AJn2B
KunlkUBqxgsi0u1xL77Yt7IzXhhPOPXL
NGt8vC4H9olcH0BxYr3oBhrFpGSjizfM
L3ZCH71RRxt8Kmy3X3R0NqQTmebcmkQ4
vj92XunE4hdfcXGbYl70c7T5T2w2rzeM
YIzzzqQ1DCUaxm4ayMP8wKeasb2BPzz0
hAtk84h6phrc6IllyV4kxSwquo90A4E
qb1EjP5NsHPJhx1jQilWY977tkCqMfq5
gM80SuCwsQIMJt4diQnPZATpoLLYCFiJ
ohZzHYHff6HnbACrnI3sZhPt47zFxadd
fI0sApLCZ04upjltfVGR8UB7oxbLRBNk
gM80SuCwsQIMJt4diQnPZATpoLLYCFiJ
4rrSr6I0NT8TbtjY0fBa6G5SxLu76X4U
YhMcUN1mfgdF0FAakkGxKMRgvMn1o4KH
JoD4RBiS2kB4KUTpbszENo7mNYeSfxdG
YM8Xdm1i2vWfYoyKUUD3Ga4vMSCB5N4R
sdV2bSL1drP8kJygoqQLjrSRPitd2TbR
hBGHA7F1dx8yhwoR9vRX9W0hieHqRAB
AkqHcvZholsKFEZYMFpRCsH0XWLNldqX
ZoQXvA7JpWaVsWZp2KdXwVEP70FjPCbG
u0erDzWpAhlfp0I7RljvBP7hZNtUo1q
upcf7w9pbq6tz6J09dMzZ1UuztuvYdR6
KunlkUBqxgsi0u1xL77Yt7IzXhhPOPXL
fI0sApLCZ04upjltfVGR8UB7oxbLRBNk
r30J9XDT3s0tZ0Bv8EUUypHixRED0skL
5EL94fXpDzA3o08q2IFwAQ7Wwd0BnUz2
NknAyxnPgpoEcWHizP4TA8ALEIyco1VT
```

Con el comando `sort data.txt`:

Y finalmente con el comando `uniq -u` .

```
bandit8@bandit:~$ sort data.txt | uniq -u  
4CKMh1JI91bUIZZPXDqGanal4xvAg0JM
```

****Nivel 10: Extrayendo texto oculto: buscando cadenas legibles**

- ***Objetivo del nivel***

Este nivel introduce el concepto de extraer cadenas legibles dentro de un archivo que puede contener datos binarios o mezclados. La contraseña estará en una de las pocas cadenas humanamente legibles, precedida por varios caracteres.

- **_Comandos utilizados

1. String: Lista las cadenas de caracteres imprimibles. Permite ver los caracteres legibles para humanos dentro de cualquier archivo.

- *Explicación paso a paso

Si hacemos un cat data.txt, en la salida tendremos cadenas que para nosotros son ilegibles.

Cuando un archivo no sea legible tenemos que lanzarle un string. Entonces con strings

data.txt obtenemos:

```
j`VcX
2#U?
V/3(o
=3?l0t
bD/@
2wNXK|
===== FGUW5illLVJrxX9kMYMmlN4MgbpfMiqey
v3A|
"y-
#kT\
=D!f
gD6YbTJ
2a18
fKes
8xXZX
frw\
2*Fx
`9ZD
S*7w
h04W]
hKQ\
JYTP
swj
d%g.
qoB[
pIW[
A}5{
PM1nN
?^` ,A
H =sS
mi s
l?w;?
67lF
w)      @U
g3@".<eD
Xe}c3
^wC*
WuNW
l,o<
z<7p
>IF<
```

**Nivel 11: Extrayendo texto oculto: buscando cadenas legibles

- *Objetivo del nivel*

Este nivel introduce el concepto de decodificación de datos en formato Base64, entonces tenemos que convertir la información codificada en texto legible. Convertir el contenido del archivo data.txt desde su formato codificado en Base64 a texto legible.

Para identificar si una cadena de texto esta en Base64, podemos guiarnos de las siguientes características:

1. Termina con uno o dos signos " = "
 2. Usa solo caracteres alfabéticos, numéricos y algunos símbolos como / , + o = .
- **_Comandos utilizados

1. base64 -d : Nos retorna la cadena de texto legible que era anteriormente.

2. cat

- ***Explicación paso a paso**

Si hacemos un cat data.txt , en la salida obtenemos lo siguiente:

```
bandit10@bandit:~$ cat data.txt  
VGhlIHBhc3N3b3JkIGlzIGR0UjE3M2ZaS2IwUlJzREZTR3NnMlJXbnB0VmozcVJyCg==
```

Esta en Base64 debido a los dos signos de igual = que tiene al final, entonces podemos convertirlo a la cadena de texto que era anteriormente. Con el siguiente comando:

```
bandit10@bandit:~$ cat data.txt | base64 -d  
The password is dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr
```

Le hacemos un cat a data.txt y esa salida la transformamos a una cadena de texto legible mediante el comando base64 -d , el parámetro -d significa decode o decodificar.

```
bandit10@bandit:~$ cat data.txt | base64 -d  
The password is dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr
```

****Nivel 12: Cifrado y decodificación: desbloqueando el texto en ROT13**

- ***Objetivo del nivel***

Este nivel introduce el concepto de cifrado ROT13, este es un método de sustitución que rota las letras del alfabeto 13 posiciones hacia adelante. Tenemos que decodificar el contenido del archivo, el cual ha sido cifrado con ROT13.

- **_Comandos utilizados
- ***Explicación paso a paso**

Podemos ir a la pagina de rot13 para decodificar el contenido .

```
bandit11@bandit:~$ cat data.txt  
Gur cnffjbeq vf 7k16JArUVv5LxVuJfsSVdbbtHGlw9D4
```

rot13.com

[About ROT13](#)

```
Gur cnffjbeq vf 7k16JArUVv5LxVuJfsSVdbbtHGlw9D4
```



ROT13 ▾



```
The password is 7x16WNeHIi5YkIhWsffIqoognUTyj9Q4
```

**Nivel 13: Descifrando el hexdump

- ***Objetivo del nivel***

Tenemos que decodificar el archivo `data.txt` desde su formato Hexdump, descomprimirlo varias veces hasta recuperar su contenido original y obtener la contraseña. Se nos dan varias recomendaciones como crear un directorio temporal.

- **Comandos utilizados**

1. `xxd -r` : Revierte un Hexdump y devuelve los datos a su formato original.
2. `mktemp -d` : Crea un directorio temporal.
3. `cp` : Nos ayuda a copiar archivos y directorios de una ubicación a otra.
4. `file`: Nos ayuda a identificar el tipo de archivo de un archivo que le mandemos.

5. `7z l` : Lista el contenido del archivo.
6. `7z x` : Extrae el contenido del archivo.
7. `gzip -d`: Descomprime archivos en formato `.gz`.
8. `tar -xvf`: El comando `tar -xvf` en Linux se usa para extraer archivos desde un archivo comprimido en formato `tar`.
9. `mv`: Sirve para mover archivos y directorios de una ubicación a otra, o para renombrarlos.

- ***Explicación paso a paso**

Teniendo todo esto en hexadecimal, podemos reversearlo y usar el comando `sponge` para guardar la salida en el mismo archivo, de esta manera todo lo que estaba en hexadecimal lo devolvemos a su estado anterior y esa misma salida la guardamos en el mismo archivo donde tenemos todo el contenido en hexadecimal.

```

bandit12@bandit:~$ cat data.txt
00000000: 1f8b 0808 41d4 f767 0203 6461 7461 322e ....A..g..data2.
00000010: 6269 6e00 0149 02b6 fd42 5a68 3931 4159 bin..I...BZh91AY
00000020: 2653 59a8 ffa7 8f00 001d 7fff dbef 7ffa &SY.....
00000030: bb7f a5ef bb7e f5fb fdff b7c7 f3ff ff7f .....~.....
00000040: ff7f fff7 deba fdःa eff7 dddf b001 3b19 .....4....@.F
00000050: a200 d01a 0190 0034 0006 800d 0340 0346 .....4....@.F
00000060: 8000 0340 0320 0069 a034 0640 0346 4680 ...@. .i.4.@.FF.
00000070: 68d1 a68c 8321 9313 4da4 f510 6406 8003 h....!..M...d...
00000080: 4006 9a00 000d 000d 0069 a007 a9a0 001a @.....i.....
00000090: 1b50 03d4 01a6 9ale a001 a343 4683 469a .P.....CF.F.
000000a0: 3d40 001a 7a8d 01a0 074c 801e ala6 8064 =@..z....L....d
000000b0: 01a3 d434 00c4 0d00 000d 0001 a680 1a19 ...4.....
000000c0: 0061 0f53 41a0 0000 0d00 341a 0320 0034 .a.SA.....4...4
000000d0: dlea 0168 4882 8244 0130 5550 f16b f52e ...hH..D.0UP.k..
000000e0: a322 cb9f bb8c aaf6 e244 cc70 b151 47c8 .".....D.p.QG.
000000f0: 6c03 a3ae 4a81 lee0 03ce 840e a978 2046 l...J.....x F
00000100: 630b 4b0d 9883 7078 e7e8 5bfb 68f1 f685 c.K...px..[.h...
00000110: 6f46 771c 3920 449f f0cb 39e2 0841 10b5 oFw.9 D...9..A..
00000120: 8714 e981 115c d1bc 2da4 318b 106c 904e .....\\...1..l.N
00000130: 9328 5e97 405a 4054 21db e049 1a32 5f3d .(^.Z@T!.I.2 =
00000140: 7069 408f f0a4 8ce5 fbea 282c 51d1 49e4 pi@.....(,Q.I.
00000150: d52f 0762 dd90 27b8 79d3 0499 52e0 060c ./b..'.y...R...
00000160: fd91 a474 d408 88f3 1fda d2d1 325a baeb ...t.....2Z..
00000170: bfe7 f0f6 cc3c 776d f369 e73c 47d4 66ea ....<wm.i.<G.f.
00000180: 4b90 e404 03b3 6a09 4687 945d 09ef 706b K.....j.F..]..pk
00000190: 8f82 2503 80d0 0a0a 3e60 f879 bf02 2d42 ..%....>`y...B
000001a0: bf37 9c96 4b22 585c 35c8 3cf1 da9f 518b .7..K"X\5.<...Q.
000001b0: ccd5 a68c 9647 aa38 8a50 89d2 f89c 1ff0 .....G.8.P.....
000001c0: 1042 18c3 6549 400d fe17 ec74 3171 6d74 .B..eI@....t1qmt
000001d0: a8bb 0def f11a 5a69 0e70 aa34 0037 b180 .....Zi.p.4.7..
000001e0: 1540 c4d2 0af7 e290 8784 ce9e 147a 6836 .@.....zh6
000001f0: 944b 3f18 2ba2 c620 af92 fb01 184f 3def .K?+.. ....0=.
00000200: 1b7d 0162 733d adca 90ac 7142 8319 f703 .}.bs=....qB....
00000210: 5930 69e2 8320 9110 5d63 0db9 9294 d4ef Y0i.. ..]c.....
00000220: 50b9 5907 0924 92c1 014e a284 25ce a6ef P.Y..$.N..%...
00000230: 67b2 4e06 6d21 4136 2ac0 292d 6638 033c g.N.m!A6*.)-f8.<
00000240: 21af be4e 13bb b74f 2c10 18c7 eea3 c436 !..N...0,.....6
00000250: c988 05e6 5638 1ff1 7724 5385 090a 8ffa ....V8..w$S.....
00000260: 78f0 d951 192d 4902 0000 x..Q.-I...
bandit12@bandit:~$
```

Sucede que cuando un archivo se convierte a un Hexdump con el comando `xxd` , su contenido se presenta en valores hexadecimales, por eso para revertir este proceso usamos el comando `xxd -r` .

Entonces:

Creamos un directorio temporal con el comando `mktemp -d`

```

bandit12@bandit:~$ mktemp -d
/tmp/tmp.En49Wqr3x4
```

Ingresamos a ese directorio

```
bandit12@bandit:~$ cd /tmp/tmp.En49Wqr3x4
```

Y copiamos el archivo `data.txt` a ese directorio, para eso debemos poner el comando `cp` seguido de la ruta absoluta:

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ cp /home/bandit12/data.txt .
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data.txt
```

Ya teniendo el archivo copiado a nuestro directorio temporal, procedemos a usar el comando `xxd -r` para revertir el Hexdump y devolver el archivo a su estado original. Y eso lo metemos en un archivo llamado `data1`.

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ cat data.txt | xxd -r > data1
```

Ahora, con el comando `file` podemos ver que es un archivo comprimido, para extraer archivos comprimidos, podemos usar los comandos `7z`, con el parámetro `l`, nos muestra cual es el archivo que vamos a extraer o el contenido que se va a extraer. Y con el parámetro `x` extraemos su contenido.

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data1 data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data1
data1: gzip compressed data, was "data2.bin", last modified
bandit12@bandit:/tmp/tmp.En49Wqr3x4$
```

O también podemos usar el comando `gzip -d`, para ello tendríamos que cambiar el nombre que se le asigno al archivo que contiene el `data.txt`, agregarle al final `.gz`, mediante el comando `mv data1 data1.gz`. Para que al usar el comando `gzip -d` no tengamos errores y pueda identificarlo correctamente.

Vemos que ahora es un archivo comprimido de tipo `bzip`. Entonces tenemos que cambiar el tipo de archivo de `data1.gz` a `data2.bz`. Ya que `data1` ya se uso y es el archivo extraído.

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data1.gz data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ gzip -d data1.gz
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data1 data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data1
data1: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ |
```

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data1 data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data1 data2.bz
bandit12@bandit:/tmp/tmp.En49Wqr3x4$
```

Ahora, usando el comando `bzip2 -d data`, extraemos el contenido

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data2.bz data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ bzip2 -d data2.bz
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data2 data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data2
data2: gzip compressed data, was "data4.bin", last modified:
```

Vemos en la imagen que llegamos a un punto donde el archivo ahora es de tipo `tar`. Entonces tenemos que mover `data3` a otro archivo de tipo `tar`.

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data2 data3.gz
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data3.gz data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ gzip -d data3.gz
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data3 data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data3
data3: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ |
```

Y usando el comando `tar -xvf`, extraemos el contenido del archivo.

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data3
data3: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data3 data4.tar
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data4.tar data.txt
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ tar -xvf data4.tar
data5.bin
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data5.bin
data5.bin: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls
data4.tar data5.bin data.txt
```

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data5.bin data6.tar
```

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ tar -xvf data4.tar  
data5.bin  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data5.bin  
data5.bin: POSIX tar archive (GNU)  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls  
data4.tar data5.bin data.txt  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data5 data6.tar  
mv: cannot stat 'data5': No such file or directory  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data5.bin data6.tar  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls  
data4.tar data6.tar data.txt  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ tar -xvf data6.tar  
data6.bin  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data6.bin  
data6.bin: bzip2 compressed data, block size = 900k  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data6.bin data7.bz  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls  
data4.tar data6.tar data7.bz data.txt  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ bzip2 -d data7.bz  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$
```

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls  
data4.tar data6.tar data7 data.txt  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data7  
data7: POSIX tar archive (GNU)  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data7 data8.tar  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls  
data4.tar data6.tar data8.tar data.txt  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ tar -xvf data8.tar  
data8.bin  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data8.bin  
data8.bin: cannot open `data8.bin' (No such file or directory)  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data8.bin  
data8.bin: gzip compressed data, was "data9.bin", last modified:  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ mv data8.bin data9.gz  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ |
```

```
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ ls  
data4.tar data6.tar data8.tar data9 data.txt  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ file data9  
data9: ASCII text  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ cat data9  
The password is F05dwFsc0cbaIiH0h8J2eUks2vdTDwAn  
bandit12@bandit:/tmp/tmp.En49Wqr3x4$ |
```

**Nivel 14: Autenticación con claves SSH

- *Objetivo del nivel*

Utilizar la clave privada SSH para conectarnos como el usuario `bandit14`, acceder al sistema y leer el archivo donde esta almacenada la contraseña.

- ** _Comandos utilizados

1. `ssh -i /home/bandit13/sshkey.private bandit14@localhost -p 2220 .`
2. `chmod 600 id_rsa .`
3. `cp nombre_del_archivo_con_la_clave_publica authorized_keys .`
4. `ssh-keygen .`

- *Explicación paso a paso

Aquí entran en juego el par de claves, con el comando `ssh-keygen` podemos crearnos un par de claves. Una publica y una privada. Las claves privadas deben disponer del privilegio `chmod 600 id_rsa`. Esto para que solo el propietario pueda leer y manipular la clave.

Por ejemplo, en nuestra terminal: En el directorio en donde ejecutamos ese comando, se nos van a liberar dos archivos en la ruta que se ve en la imagen. `id_ed` y `id_ed.pub`, este ultimo es la clave privada.

```
> ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/kali/.ssh/id_ed25519):
/home/kali/.ssh/id_ed25519 already exists.
Overwrite (y/n)? y
Enter passphrase for "/home/kali/.ssh/id_ed25519" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kali/.ssh/id_ed25519
Your public key has been saved in /home/kali/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:0vGpiPom+k60NsGI0fyLhm/HcFvZpI3n7NxGlyFEko0 kali@kali
The key's randomart image is:
+--[ED25519 256]--+
|      *o.          |
|      E.o .        |
|       . o          |
|       ..o .o       |
|= .     .BS o.     |
|=+.   ..=o+..    |
|..++o.+..       |
|.B+o=  .o..      |
|=BB*o  .o..      |
+---[SHA256]-----+

```

```
> ls
authorized_keys  id_ed25519  id_ed25519.pub  known_hosts

```

Si queremos acceder a nuestro sistema , sin proporcionar contraseña lo que podemos hacer es jugar con la clave publica, es decir, la `id_ed` . Teniendo esa clave publica, si usamos el comando `cp nombre_del_archivo_con_la_clave_publica authorized_keys` . Convertimos la clave publica a una autorizada. Este proceso permite el acceso SSH sin necesidad de ingresar una contraseña, usando **autenticación con claves públicas**.

A la hora de tratar de conectarme a mi maquina, no me va a pedir contraseña.

Por ejemplo, si estamos en otro equipo, y queremos conectarnos a nuestra maquina de forma remota sin proporcionar contraseña. Desde el otro equipo tendremos que hacer un `ssh-keygen` , la clave publica (del usuario atacante) previamente incorporarla en el directorio personal de la maquina del usuario a la que queremos entrar. La clave publica siempre convertirla a una autorizada con el comando que ya mencionamos. Desde la ruta `/home/kali/.ssh` cuando hagamos un `cat` a la clave publica, en vez de que aparezca kali debe aparecer el nombre de usuario que quiere entrar a la maquina. De esta forma, desde su equipo se podrá conectar a la maquina sin proporcionar contraseña, solo con tener añadida la clave a nuestro directorio personal.

Pero existe otra forma, eliminar `authorized_keys` , hacer la clave privada, una autorizada para que cualquiera que disponga de esta clave privada se pueda conectar a la maquina. Con el comando `ssh-copy-id` yo puedo copiar claves SSH a servidores remotos, de modo que se pueda iniciar sesión sin contraseña. Para usarlo, se debe especificar el archivo a copiar con el parámetro `-i` . Entonces, le pasamos la clave privada y luego conectarnos a esa maquina.

Ahora, si hacemos un `cat sshkey.private` : Podemos ver la clave privada

```
bandit13@bandit:~$ cat sshkey.private
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAxkk0E83W2cOT7IWhFc9aPaaQmQDdgzuXCv+ppZHa++buSkN+
gg0tcr7Fw8NLGa5+Uzec2rEg0WmeevB13AIoYp0MZYETq46t+jk9puNwZwIt9XgB
ZufGtZEwWbFWw/vVLNw0XBe4UWStGRWzgPpEeSv5Tb1VjLZIBdGphTIK22Amz6Zb
ThMsimNyJafEwJ/T8PQ03myS91vUHEuo0MAzoUID4kN0MEZ3+XahyK0HJVq68KsV
0befXG1vvA3GAJ29kxJaqvRfgYnqZryWN7w3ChjNU4c/2Jkp+n8L0SnxaNA+WYA7
jiPyTF0is8uzMlYQ4l1Lzh/8/MpvhCQF8r22dwIDAQABoIBAQC6dWBjhyE0zjeA
J3j/RWmap9M5zfJ/wb2bfidNpwbB8rsJ4sZIDZQ7XuIh4LfyoAQSS+bBw3RXvzE
pvJt3SmU8hIDuLsCjL1VnBY5pY7Bju8g8aR/3FyjyNAqx/TLfz1LYf0u7i9Jet67
xAh0tONG/u8FB5I3LAI2Vp60viwvdWeC4n0xCthldpuPKNLA8rmMMVRTKQ+7T2VS
nXmwYckKucUgzoVSpINzaS0zUDypdpy2+tRH3Mqa5kqN1YkjvF8RC47wo0YCktsD
o3FFpGNFec9Taa3Msy+DfQQhHKZFKIL3bJD0NtmrVvtYK40/yeU4aZ/HA2DQzwhe
ol1AfiEhAoGBAOnVjosBkm7sblK+n4IEwPxs8s0mhPnTDUy5WGrpSCrX0msVIBUF
laL3ZGLx3xCiwtCnEucB9DvN2HZkupc/h6hTKUYLqXuyLD8njTrbRhLgbC9QrKrS
M1F2fSTxVqPtZDlDMwjNR04xHA/fKh8bXXyTMq0HNJTHHNhbh3McdURjAoGBANKU
1hqfnw7+aXncJ9bjysr1Zwbq0E5Nd8AFg fwaKuGTTVX2NsUQnCMWd0p+wFak40JH
PKWkJNdBG+ex0H9JNQsTK3X5PBMAS8AfX0GrKeuwKWA6erytVTqj0fLYcdp5+z9s
8DtVCxDuVsM+i4X8UqIG0lvGbtKEVokHPFXP1q/dAoGAcHg5YX7WEehCgCYTzp0+
xysX8ScM2qS6xuZ3MqUWAxUWkh7NGZvhe0sGy9i0dANzwKw7mUUUViaCMR/t54W1
GC83s0s3D7n5Mj8x3Nd08xFit7dT9a245Tva0YQ7KgmqpSg/ScKCw4c3eiLava+J
3btnJeSIU+8ZXq9XjPRpKwUCgYA7z6Li0QKxNeXH3qHXcnHok855maUj5fJNpPbY
iDkyZ8ySF8GlcFsky8Yw6fWCqfG3zDrohJ5l9JmEsBh7SadkwsZhvecQcS9t4vby
9/8X4jS0P8ibfcKS4nPBP+dT81kkkg5Z5MohXBORA7VWx+AcohcDEkprsQ+w32xeD
qT1EvQKBgQDKm8ws2ByvSUVs9GjTilCajFqlJ0eVYzRPaY6f++Gv/UVfAPV4c+S0
kAWpXbv5tbkkzbS0eaLPTKgLzavXtQoTtKwrjp0lHKIHUz6Wu+n4abfAIRFub0dN
/+aLoRQ0yBDRbdXMsZN/jvY44eM+xRLdRVyMmdPtP8belRi2E2aEzA==
-----END RSA PRIVATE KEY-----
bandit13@bandit:~$ |
```

¿ Que hacemos si en una maquina solo tenemos una clave privada ?

1. Hacer toda esa clave privada un archivo de identidad.

Podemos hacer un `ssh -i sshkey.private` para conectarnos con el nombre de usuario a esa maquina.

Usando el comando `ssh -i /home/bandit13/sshkey.private bandit14@localhost -p 2220`. SSH es el comando para establecer una conexión remota segura con otro usuario o otro servidor, `-i /home/bandit13/sshkey.private` especifica una clave privada para autenticación, `bandit14@localhost` intenta conectarse como el usuario `bandit14` en `localhost` . `-p 2220` especifica el puerto de conexión 2220. Para finalizar, `bandit13` usa la clave privada para iniciar sesión como `bandit14` sin necesidad de ingresar contraseña, `localhost` indica que la conexión se hace dentro de la misma maquina, en lugar de otro servidor externo. `-p 2220` especifica el puerto correcto, ya que Bandit usa el 2220 en lugar del puerto estándar 22.

Y ya lograríamos entrar

```
--[ More information ]--  
  
For more information regarding individual wargames, visit  
http://www.overthewire.org/wargames/  
  
For support, questions or comments, contact us on discord or IRC.  
  
Enjoy your stay!  
  
bandit14@bandit:~$
```

**Nivel 15: Descifrando la respuesta del puerto 30000

- *Objetivo del nivel*

Este nivel introduce el concepto de comunicación con puertos de red en un sistema. Para obtener la contraseña del próximo nivel, debemos enviar la contraseña actual al puerto 30000 en `localhost` y recibir la respuesta. Entonces, debemos enviar la contraseña actual al puerto 30000 en `localhost`.

- **_Comandos utilizados

1. `nc localhost 30000 .`
2. `cat /etc/bandit_pass/bandit14 .`

- ***Explicación paso a paso**

Existen 65535 puertos, y varios protocolos, como el protocolo TCP (Transmission Control Protocol), esta esta orientado a conexiones. Garantiza el orden correcto de los paquetes, que lleguen al destinatario, que no haya desfragmentación o que haya perdida de paquetes. Netcat se le conoce como " La navaja suiza del TCP/IP" , se usa para diagnosticar errores y problemas que afecten a la funcionalidad y la seguridad de una red. Netcat también puede escanear puertos y transferir datos. Además de permitir configurar servidores de chat y de web e iniciar consultas por correo. Puede operar en modo servidor y cliente. Con el comando `nc localhost 30000` estamos usando Netcat para conectarnos al puerto 30000 en el `localhost`. Sucede que Netcat permite la lectura y escritura de datos a través de conexiones de red utilizando protocolos TCP o UDP. Es decir, estamos estableciendo una conexión TCP al puerto 30000 en la maquina local.

Hacemos un `cat /etc/bandit_pass/bandit14` para acceder a la contraseña, En Bandit las contraseñas para cada nivel están guardadas en la ruta `/etc/bandit_pass` . Y el nivel nos dice para acceder al siguiente nivel debemos usar la contraseña del nivel a actual, mandándola

al puerto 30000.

```
bandit14@bandit:~$ cat /etc/bandit_pass/bandit14
MU4VWeTyJk8R0of1qqmcBPaLh7lDCPvS

bandit14@bandit:~$ nc localhost 30000
MU4VWeTyJk8R0of1qqmcBPaLh7lDCPvS
Correct!
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
```

**Nivel 16: Descifrando la respuesta rel puerto 30000

- *Objetivo del nivel*

Este nivel introduce la comunicación segura con SSL/TLS. A diferencia del nivel 14, en el que simplemente enviamos la contraseña al puerto 30000, ahora debemos enviar la contraseña al puerto 30001 en localhost, pero usando una conexión cifrada con SSL/TLS.

- ** _Comandos utilizados

1. ncat -ssl localhost 30001
2. cat /etc/bandit_pass/bandit15

- ***Explicación paso a paso**

SSL (Secure Sockets Layer) y TLS (Transport Layer Security) son protocolos de seguridad que cifran el tráfico de datos entre un navegador y un sitio web. Con el comando `ncat --ssl localhost 30001` estamos estableciendo una conexión segura ssl a la dirección IP en el puerto 30001. `ncat` se usa para establecer conexiones de red, enviar archivos o diagnosticar problemas de seguridad. `ncat` puede encriptar el tráfico usando SSL. En el modo de conexión, basta con añadir la opción `--ssl`, esta habilitado el cifrado SSL/TLS . Asegurando que los datos enviados no puedan ser interceptados fácilmente.

Hacemos el mismo procedimiento como en el nivel anterior para acceder a la contraseña de `bandit15` , usamos el comando `cat /etc/bandit_pass/bandit15` .

```
bandit15@bandit:~$ cat /etc/bandit_pass/bandit15
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo

bandit15@bandit:~$ ncat --ssl 127.0.0.1 30001
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
Correct!
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
```

La dirección IP `127.0.0.1` es la dirección estándar de `localhost` . En lugar de escribir `localhost` , podemos usar esa ip, ya que ambos significan lo mismo.

**Nivel 17: Encontrando el puerto correcto

- *Objetivo del nivel*

Este nivel introduce el concepto de escaneo de puertos y análisis de protocolos SSL/TLS. Para obtener la contraseña del próximo nivel, necesitamos explorar los puertos 31000-32000 en localhost, determinar cuales tienen un servidor activo, identificar si se usan SSL/TLS, y luego enviar la contraseña al único servidor que responda con las credenciales correctas.

- ** _Comandos utilizados

1. nmap -p 31000-32000 127.0.0.1
2. mktemp -d
3. nc -ssl 217.0.0.1 puerto
4. chmod 600 : Nos ayuda a gestionar los permisos de acceso a archivos y directorios.
5. ssh -i /home/bandit16/private.key bandit17@localhost -p 2220 : El parámetro -i lo usamos para especificar una clave privada que será usada para autenticarse en el servidor en lugar de una contraseña.

- *Explicación paso a paso

En este nivel es necesario que comencemos a hacer uso de una herramienta conocida como `nmap` , que significa " Network Mapper " que nos ayuda a explorar redes y sistemas mediante el escaneo de direcciones IP, puertos y servicios. Se nos especifica que debemos mirar que puertos están abiertos en el rango de 31000-32000 , para ello podemos usar el comando `nmap -p 31000-32000 127.0.0.1` . Con ese comando le estamos diciendo a nmap que escanee los puertos con el parámetro -p en el rango de 31000 a 32000 en el localhost (127.0.0.1) .

```
bandit16@bandit:~$ nmap -p 31000-32000 127.0.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-27 17:29 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00015s latency).
Not shown: 995 closed tcp ports (conn-refused)
PORT      STATE     SERVICE
31046/tcp open      unknown
31337/tcp filtered Elite
31518/tcp open      unknown
31691/tcp open      unknown
31790/tcp open      unknown
31960/tcp open      unknown
```

Si queremos realizar la conexión con el servidor, tenemos que mirar puerto por puerto cual esta en escucha, pasarle la clave de `bandit16` y nos dará algo en la salida.

En este caso nos dio una clave privada:

```
bandit16@bandit:~$ nc -ssl 127.0.0.1 31790
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
Correct!
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAvmOkuifmMg6HL2YPI0jon6iWfbp7c3jx34YkYWqUH57SUdyJ
imZzeyGC0gtZPGujUSxiJSWI/oTqexh+cAMTSMLOJf7+BrJ0bArnx9Y7YT2bRPQ
Ja6Lzb558YW3FZL870Ri0+rW4LCDNd2lUvLE/GL2GWyuKN0K5iCd5TbtJzEkQTu
DSt2mcNn4rhAL+JFr56o4T6z8WWAW18BR6yGrMq7Q/kALHYW30ekePQAzL0VUYbW
JGTi65CxbCnzC/w4+mqQyvmzpWtMAzJTzAzQxNbK2MBGySxDLrjg0LWN6sK7wNX
x0YVztz/zbIkPjfku1jHS+9EbVNj+D1XF0JuaQIDAQABAoIBABagpxpM1aoLWfvD
KHcj10nqcoBc4oE11aFYQwik7xfW+24pRNuDE6SFth0ar69jp5R1LwD1NhPx3iBl
J9n0M80J0VToum43U0S8YxF8WwhXriYGnc1sskbwpXOUDc9uX4+UESzH22P29ovd
d8WErY0gPxun8pbJLmxkAtWNhpMvfe0050vk9TL5wqbu9AlbssgTccXkMQnPw9nC
YNN6DDP2lbcBrvgT9YCNL6C+ZKufD52y0Q9q0kwFTEQpjF4uNtJom+asvlpmS8A
vLY9r60wYSvmZhNqBURj7lyCtXMIu1kkd4w7F77k+DjHoAXyxcUp1DGL51s0mama
+T0WwgECgYEAE8JtPxP0GRJ+IQkX262jM3dEIka8ky5moIwUqYdsx0NxHgRRhORT
8c8hAuRBb2G82so8vUHK/fur850Efc9TncnCY2crpoqsgifKLxrLgtT+qDpfZnx
SatLdt8GfQ85yA7hnWWJ2MxF3NaeSDm75Lsm+tBbAiyc9P2jGRNtMSkCgYEAypHd
HCctNi/FwjulhttFx/rHYKhLidZDFYeie/v45bN4yFm8x7R/b0iE7KaszX+Exdvt
SghaTdcG0Knyw1bpJVyusavPzpaJMjdJ6tcFhVAbAjm7enCIvGCSx+X3l5SiWg0A
R57hJglezIiVjv3aGwHwv1ZvtszK6zV6oXFAu0ECgYAbjo46T4hyP5tJi93V5HDi
TtieK7xRVxUl+iU7rWkGAXFpMLFteQEsr7PJ/lemmEY5eTDAFMLy9FL2m9oQWCg
R8VdwSk8r9FGLS+9aKcV5PI/WEKlwgXinB30hYimtiG2Cg5JCqIZFHxD6MjEG0iu
L8ktHMPvodBwNsSBULpG0QKBgBAplTfC1HOnWiMGOU3KPwYwt006CdTkmJ0mL8Ni
blh9elyZ9FsGxsgtRBXRsqXuz7wtsQAgLHxbdLq/ZJQ7YfzOKU4ZxEnabvXnvWkU
Y0djHdSOoKvDQNwu6ucyLRAWFuISExw9a/9p7ftpxm0TSgyvmfLF2MIAEwyzRqaM
77pBAoGAMmjIJDjp+Ez8duyn3ieo36yrttF5NSsJLABxFpd1cgvTCWW+9Cq0b
dxviW8+TFVEBl104f7HVm6EpTscdDxU+bCXWkfjuRb7Dy9G0tt9JPsX8MBTakzh3
vBgsyi/sN3RqRBcGU40f0oZyfAMT8s1m/uYv5206IgeuZ/ujbjY=
-----END RSA PRIVATE KEY-----
```

Ahora, lo que tenemos que hacer es crear un archivo que se llame `private.key`, guardar esa clave privada para luego darle permisos con `chmod 600`. Y hacer lo mismo para el nivel anterior, establecer la conexión con la maquina de `bandit17` usando la clave privada, con el comando `ssh -i /home/bandit16/private.key bandit17@localhost -p 2220`. Debemos cambiar la ruta en la que esta la clave privada. En este caso en el directorio temporal.

```
bandit16@bandit:/tmp/tmp.cXNUyrNSLS$ chmod 600 private.key
```

```
bandit16@bandit:/tmp/tmp.cXNUyrNSLS$ ssh -i /tmp/tmp.cXNUyrNSLS/private.key bandit17@localhost -p 2220
```

**Nivel 18: Descifrando la contraseña escondida

- *Objetivo del nivel*

Este nivel se trata de comparar archivos y detectar cambios, en el directorio personal tenemos dos archivos, `passwords.old` que contiene la versión antigua de las contraseñas y `passwords.new` que contiene la versión nueva, con una única linea modificada. Para obtener

la contraseña de Bandit18, debemos encontrar la linea que se cambio entre `passwords.old` y `passwords.new`.

- **_Comandos utilizados

1. `wc -l *`
2. `diff archivo_1 archivo_2`

- *Explicación paso a paso

Con el comando `wc -l *` podemos listar o contar las líneas de archivos que se encuentran en el directorio actual de trabajo.

```
bandit17@bandit:~$ ls  
passwords.new passwords.old  
bandit17@bandit:~$ wc -l *  
100 passwords.new  
100 passwords.old  
200 total
```

Con el comando `diff` podemos comparar las diferencias entre ficheros que le pasemos linea a linea. Es muy útil cuando queremos comprobar las diferencias entre archivos.

```
bandit17@bandit:~$ diff passwords.old passwords.new  
42c42  
< C6XNBdY0kgt5ARXESMKWWOUwBeaIQZ0Y  
---  
> x2gLTTjFwM0hQ8oWNbMN362QKxfRqGlo  
bandit17@bandit:~$ |
```

Donde `<` significa que se ha quitado y `>` que se ha agregado.

**Nivel 19: Evita la expulsión y descubre la contraseña

- *Objetivo del nivel*

La contraseña para el nivel 19 esta guardada en un archivo llamado `readme` dentro del directorio principal, sin embargo, tenemos un problema, al conectarnos a bandit19 con ssh, el archivo `.bashrc` ha sido modificado para desconectaros automáticamente, lo que impide que podamos ejecutar comandos normalmente.

- **_Comandos utilizados

1. `sshpass -p 'flag' ssh bandit19@bandit.labs.overthewire.org -p 2220 bash`

- *Explicación paso a paso

Como ya mencionamos, al intentar la conexión con SSH, nos desconectamos inmediatamente, mostrándonos el siguiente mensaje:

```
* gabin1t (http://  
* pwntools (http://  
* radare2 (http://  
  
--[ More information  
  
For more information:  
http://www.overthewire.org/wargames  
  
For support, questions,  
or feedback:  
  
Enjoy your stay!  
  
Byebye !  
Connection to bandit18 closed.
```

Para hallar la contraseña podemos al final del comando para la conexión, agregar un `bash`, de esta manera accedemos a la bash.

```
> sshpass -p 'x2gLTTjFwM0hQ8oWNbMN362QKxfRqGl0' ssh bandit18@bandit.labs.overthewire.org -p 2220 bash  
  
[|██████████|] 100%  
  
This is an OverTheWire game server.  
More information on http://www.overthewire.org/wargames
```

Y hacemos un `ls`, para ver el archivo `readme` y hacerle un `cat` para finalmente obtener la contraseña.

```
ls  
readme  
cat readme  
cGWPMaKXVwDUNgPAVJbWYuGHVn9zl3j8
```

**Nivel 20: Accediendo con privilegios

- *Objetivo del nivel*

Este nivel introduce el concepto de SetUID binaries, una técnica clave en sistemas Linux. El archivo ejecutable ubicado en el directorio principal tiene el bit SetUID activado, lo que significa que, cuando se ejecuta, toma los privilegios del propietario en lugar de los del usuario que lo ejecuta.

- ** _Comandos utilizados

1. `ls -l`

```
2. ./bandit20-do cat /etc/bandit_pass/bandit20
```

- *Explicación paso a paso

Si queremos saber si un archivo es SUID, es hacer un `ls -l`, de esta manera se nos van a mostrar los permisos de los archivos. Si el archivo tiene el bit SUID establecido, veremos una `s` en los permisos del propietario en lugar de una `x`.

```
bandit19@bandit:~$ ls  
bandit20-do  
bandit19@bandit:~$ ls -l  
total 16  
-rwsr-x--- 1 bandit20 bandit19 14884 Apr 10 14:23 bandit20-do  
bandit19@bandit:~$ |
```

SUID es un permiso de archivo especial para archivos ejecutables que permite a los usuarios ejecutar el archivo con los permisos efectivos del propietario del archivo. SGID, por el contrario, es un permiso de archivo especial que también se aplica a los archivos ejecutables y permite a otros usuarios heredar el GID efectivo del propietario del grupo de archivos. SUID significa "establecer ID de usuario" o Set owner User ID y SGID significa "establecer ID de grupo" o Set Group ID up on execution.

Si ejecutamos el comando `./bandit20-do cat /etc/bandit_pass/bandit20`, podemos hallar la flag . En este comando el punto y la barra indican que estamos ejecutando un archivo en el directorio actual , bandit20-do es un binario con SetUID activado, con el `./` le estamos indicando explícitamente a la terminal que el archivo esta en el directorio actual, para después pasarle la ruta de la `bandit_pass` .

```
bandit19@bandit:~$ ./bandit20-do sh  
$ lsl  
sh: 1: lsl: Permission denied  
$ ls  
bandit20-do  
$ whoami  
bandit19  
$  
env. cat/etc/bandit_pass/bandit20: No such file or directory  
bandit19@bandit:~$ ./bandit20-do cat /etc/bandit_pass/bandit20  
0qXahG8Zj0VMN9Ghs7i0WsCfZyXOUbY0
```

**Nivel 21: Descifrando la clave mediante puertos abiertos

- *Objetivo del nivel*

Este nivel nos desafía a interactuar con un binario SetUID, pero ahora incorporando comunicación en red mediante puertos abiertos. En el directorio principal hay un binario

especial que toma un numero de puerto como argumento y se conecta a `localhost`.

- **_Comandos utilizados

1. `nc -nlvp .`
2. `./suconnect 35000 .`

- ***Explicación paso a paso**

Con el comando `nc -nlvp` , podemos abrir un puerto temporal, mediante el cual podemos estar en escucha, tenemos que pasarle un puerto.

```
bandit20@bandit:~$ ls
suconnect
bandit20@bandit:~$ nc -nlvp 35000
Listening on 0.0.0.0 35000
```

Luego , abrimos otra terminal y nos volvemos a conectar a bandit20, para después ejecutar el `suconnect` en el puerto que especificamos, con el comando `./suconnect 35000` .

```
bandit20@bandit:~$ ./suconnect 35000
```

En la terminal principal se nos aparecerá un mensaje de conexión recibida.

```
Connection received on 127.0.0.1 41878
|
```

Ahí mismo colocamos la contraseña que usamos para ingresar a `bandit20` .

```
Connection received on 127.0.0.1 41878
0qXahG8Zj0VMN9Ghs7i0WsCfZyX0UbY0
```

En nuestra segunda terminal, nos aparecerá que se leyó la contraseña que le mandamos , para después mandarnos de vuelta la nueva contraseña (en nuestra terminal principal) .

```
EeoULMCra2q0dSkYj561DX7s1CpBu0Bt
bandit20@bandit:~$ |
```

**Nivel 22: Descubre la clave automatizada espiando Cron Jobs

- *Objetivo del nivel*

Un programa se ejecuta automáticamente a intervalos regulares usando `cron` , el planificador de tareas de Linux. La clave para este nivel esta en un comando que se ejecuta desde `/etc/cron.d/` . Debemos encontrar el cron Job correcto y descubrir que comando esta ejecutando para obtener la contraseña de `Bandit22`.

- **_Comandos utilizados

1. /etc/cron.d/
2. /usr/bin/cronjob_bandit22.sh
3. cat /etc/bandit_pass/bandit22 > /tmp/t706lds9S0RqQh9aMcz6ShpAoZKF7fgv

- ***Explicación paso a paso**

Las tareas cron son tareas que se ejecutan en el sistema a intervalos regulares de tiempo. En un archivo cron, **cinco asteriscos** representan que la tarea se va a ejecutar cada minuto. Si nos vamos al directorio /etc/cron.d/ donde están almacenadas las tareas cron del usuario bandit22 , y hacemos un ls obtenemos esto.

```
bandit21@bandit:~$ cd /etc/cron.d/
bandit21@bandit:/etc/cron.d$ ls
clean tmp cronjob bandit22 cronjob bandit23 cronjob bandit24 e2scrub all otw-tmp-dir sysstat
```

Entonces el usuario bandit22 esta ejecutando ese servicio cada minuto.

```
bandit21@bandit:/etc/cron.d$ cat cronjob_bandit22
@reboot bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
* * * * * bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
bandit21@bandit:/etc/cron.d$ |
```

El script se ejecuta periódicamente por el sistema gracias a cron, lo que significa que la contraseña se actualiza constantemente en /tmp/cadena_de_texto . Si hacemos un cat /usr/bin/cronjob_bandit22.sh podemos ver que esta haciendo el script cronjob_bandit22.sh , que es el responsable de guardar la contraseña en /tmp/ . /usr/bin/cronjob_bandit22.sh , es un script en Bash que se ejecuta automáticamente con cron .

```
bandit21@bandit:/etc/cron.d$ cat /usr/bin/cronjob_bandit22.sh
#!/bin/bash
chmod 644 /tmp/t706lds9S0RqQh9aMcz6ShpAoZKF7fgv
cat /etc/bandit_pass/bandit22 > /tmp/t706lds9S0RqQh9aMcz6ShpAoZKF7fgv
bandit21@bandit:/etc/cron.d$ |
```

El comando cat /etc/bandit_pass/bandit22 > /tmp/t706lds9S0RqQh9aMcz6ShpAoZKF7fgv , toma la contraseña de Bandit22 y la almacena en /tmp/ , permitiendo que cualquier usuario pueda leerla sin necesitar acceso directo a /etc/bandit_pass/ . Entonces la contraseña se almacena en un archivo temporal dentro de /tmp/ .

Si hacemos un cat a esa ruta, obtenemos la contraseña o la flag:

```
bandit21@bandit:/etc/cron.d$ cat /tmp/t706lds9S0RqQh9aMcz6ShpAoZKF7fgv
tRae0UfB9v0UzbCdn9cY0gQnds9GF58Q
bandit21@bandit:/etc/cron.d$ |
```

**Nivel 23: Espiando la automatización

- *Objetivo del nivel*

Un programa que se ejecuta automáticamente en intervalos regulares gracias a `cron`. debemos analizar el script de `cron`, ver que comando ejecuta, y descubrir donde almacena la contraseña de Bandit23 .

- ** _Comandos utilizados

1. cat

2. ls

- *Explicación paso a paso

Con el comando `md5sum` puedo verificar la integridad de los archivos, en la salida veremos un hash, podemos transferirnos archivos entre maquinas , meter un `md5sum` al archivo origen y ser si el hash es diferente . Si es así puede ser que se ha manipulado, se ha corrompido o no se ha enviado correctamente. Si los hashes para el origen y el destino son lo mismo, entonces no sufrió nada.

```
bandit22@bandit:/etc/cron.d$ echo I am user $myname | md5sum  
7db97df393f40ad1691b6e1fb03d53eb -  
bandit22@bandit:/etc/cron.d$ echo I am user $myname | md5sum  
7db97df393f40ad1691b6e1fb03d53eb -  
bandit22@bandit:/etc/cron.d$ echo I am user $myname | md5sum  
7db97df393f40ad1691b6e1fb03d53eb -  
bandit22@bandit:/etc/cron.d$ |
```

El script realiza acciones de forma automática gracias al `cron job` , primero, obtiene el nombre del usuario actual, luego, genera un identificador único usando `md5sum` , después muestra un mensaje indicado la copia de los archivos, posteriormente, copia la contraseña del usuario en `/tmp/` con el nombre generador.

```
bandit22@bandit:/etc/cron.d$ ls  
clean_tmp cronjob_bandit22 cronjob_bandit23 cronjob_bandit24 e2scrub  
bandit22@bandit:/etc/cron.d$ cat cronjob_bandit23  
@reboot bandit23 /usr/bin/cronjob_bandit23.sh &> /dev/null  
* * * * * bandit23 /usr/bin/cronjob_bandit23.sh &> /dev/null  
bandit22@bandit:/etc/cron.d$ cat /usr/bin/cronjob_bandit23.sh  
#!/bin/bash  
  
myname=$(whoami)  
mytarget=$(echo I am user $myname | md5sum | cut -d ' ' -f 1)  
  
echo "Copying passwordfile /etc/bandit_pass/$myname to /tmp/$mytarget"
```

```
cat /etc/bandit_pass/$myname > /tmp/$mytarget  
bandit22@bandit:/etc/cron.d$ /usr/bin/cronjob_bandit23.sh  
Copying passwordfile /etc/bandit_pass/bandit22 to /tmp/8169b67bd894ddb4412f91573b38db3  
bandit22@bandit:/etc/cron.d$ cat /tmp/8169b67bd894ddb4412f91573b38db3  
tRae0UfB9v0UzbCdn9cY0gQnds9GF58Q  
bandit22@bandit:/etc/cron.d$ echo I am user bandit23 | md5sum | cut -d ' ' -f 1  
8ca319486bfbbbc3663ea0fbe81326349  
bandit22@bandit:/etc/cron.d$ cat /tmp/8ca319486bfbbbc3663ea0fbe81326349  
0Zf1lioIjMVN551jX3CmStKLYqjk54Ga
```

**Nivel 24: Tu primer script en bash

- *Objetivo del nivel*

Este nivel es especial porque te desafía a escribir tu primer script en Bash.

- ** _Comandos utilizados

1. echo "cat /etc/bandit_pass/bandit24 > /tmp/certa_test.txt" > certa6.sh
2. cat /etc/bandit_pass/bandit24 > /tmp/certa_test.txt
3. chmod 777 certa6.sh
4. ls certa6.sh
5. cat /tmp/certa_test.txt

- *Explicación paso a paso

Este es el script

```
bandit23@bandit:~$ cd /etc/cron.d/
bandit23@bandit:/etc/cron.d$ ls
clean_tmp cronjob_bandit22 cronjob_bandit23 cronjob_bandit24 e2scrub_all otw-tmp-dir sysstat
bandit23@bandit:/etc/cron.d$ cat cronjob_bandit24
@reboot bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
* * * * * bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
bandit23@bandit:/etc/cron.d$ cat /usr/bin/cronjob_bandit24.sh
#!/bin/bash

myname=$(whoami)

cd /var/spool/$myname/foo
echo "Executing and deleting all scripts in /var/spool/$myname/foo:"
for i in *.*;
do
    if [ "$i" != "." -a "$i" != ".." ];
    then
        echo "Handling $i"
        owner=$(stat --format "%U" ./${i})
        if [ "${owner}" = "bandit23" ]; then
            timeout -s 9 60 ./${i}
        fi
        rm -f ./${i}
    fi
done
bandit23@bandit:/etc/cron.d$
```

Nos vamos al directorio donde se ejecutan y se eliminan los scripts

```
bandit23@bandit:/etc/cron.d$ cd /var/spool/bandit24/foo
bandit23@bandit:/var/spool/bandit24/foo$ |
```

Con el comando echo creamos un archivo certa6.sh que contiene la instrucción: cat /etc/bandit_pass/bandit24 > /tmp/certa_test.txt que tiene como finalidad copiar la

contraseña de Bandit24 al archivo /tmp/certa_test.txt.

```
bandit23@bandit:/var/spool/bandit24/foo$ echo "cat /etc/bandit_pass/bandit24 > /tmp/certa_test.txt" > certa6.sh
bandit23@bandit:/var/spool/bandit24/foo$ chmod 777 certa6.sh
bandit23@bandit:/var/spool/bandit24/foo$ ls certa6.sh
certa6.sh
bandit23@bandit:/var/spool/bandit24/foo$ cat /tmp/certa_test.txt
gb8KRRCCsshuZXI0tUuR6yp0FjiZbf3G8
bandit23@bandit:/var/spool/bandit24/foo$ |
```

**Nivel 25: Fuerza bruta para obtener la clave correcta

- *Objetivo del nivel*

Este nivel introduce fuerza bruta para descifrar un código de acceso. Un Daemon o proceso en segundo plano escucha en el puerto 30002 . Si le enviamos la contraseña de Bandit24 y el código PIN de 4 dígitos correcto, nos dará la clave de Bandit25. Sucede que el PIN no se puede obtener directamente, por lo que debemos probar todas las combinaciones posibles , desde 0000 hasta 9999, usando fuerza bruta.

- ** _Comandos utilizados

1. `for i in {00..09}; do echo $i; done .`
2. `cat combinations.txt | nc localhost 30002 | grep -v "Wrong" .`
3. `for pin in $(seq -w 0000 9999); do echo "gb8KRRCCsshuZXI0tUuR6yp0FjiZbf3G8 $pin"; done | nc -q 1 localhost 30002 .`

- **Explicación paso a paso*

El Daemon espera recibir la contraseña de bandit24 y un PIN numérico de 4 dígitos, que no conocemos y debemos probar todas las posibles combinaciones posibles.

Aquí tenemos todas nuestras combinaciones, desde el 0000 a 9999, en este ejercicio nos pedía hacer fuerza bruta, al ingresar la contraseña, y luego el pincode de 4 dígitos, en alguna de esas combinaciones estaría la respuesta.

Podemos comenzar haciendo un `for i in {00..09}; do echo $i; done .`

```
bandit24@bandit:~$ for i in {00..09}; do echo $i; done
00
01
02
03
04
05
06
07
08
09
bandit24@bandit:~$ for pin in {00..09}; do echo $pin; done
00
01
02
03
04
05
06
07
08
09
```

```
bandit24@bandit:/tmp/tmp.ZW7EoJb7rB$ for pin in {0000..0009}; do echo "gb8KRRCsshZXI0tUuR6yp0FjiZbf3G8 $pin"; done > combinatios.txt
```

En el comando nc localhost 30002 , estamos usando netcat para establecer una conexión con el puerto 30002 en la maquina local. Con el cat le estamos mandando todas las combinaciones. Al momento de ejecutar el comando, veremos todos los errores de todas las combinaciones, para ello podemos usar el comando grep con el parámetro -v , que lo que hace es quitar de la cadena de texto lo que le pasemos.

```
I am the pincode checker for user bandit25. Please enter the password for bandit25:
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
Wrong! Please enter the correct current password and pincode. Try again.
```

```
bandit24@bandit:/tmp/tmp.ZW7EoJb7rB$ cat combinatios.txt | nc localhost 30002 | grep -v "Wrong"
```

Aplicamos el comando cat combinations.txt | nc localhost 30002 | grep -v "Wrong" lo que hacemos es automatizar el ataque de fuerza bruta al Daemon en el puerto 30002 . Funciona de la siguiente manera: Mostramos el contenido del archivo combinatios.txt, que contiene todas las posibles combinaciones de la contraseña y el código PIN. Con el | pasamos la salida del comando cat como entrada al siguiente comando nc . nc localhost 30002 , establece una conexión con el Daemon en el puerto 30002 y envía al entrada recibida

desde cat combinatios.txt . Con el segundo pipe | pasamos la salida del comando nc como entrada a grep. El comando grep -v "Wrong" filtra la respuesta del Daemon, excluyendo cualquier linea que contenga " Wrong " .

Finalmente, podemos aplicar for pin in \$(seq -w 0000 9999); do echo

```
"gb8KRRCCsshZXI0tUuR6yp0FjiZbf3G8 $pin"; done | nc -q 1 localhost 30002 , seq -w 0000 9999 nos genera todos los números en formato de 4 dígitos, asegurando que 0001 no se convierta en 1.
```

El parámetro usando en el nc localhost para redirigir cada intento de contraseña + PIN al Daemon que corre en el puerto 30002 permite que al conexión nunca se cierre o se interrumpa.

```
Wrong! Please enter the correct current password and pincode. Try  
Wrong! Please enter the correct current password and pincode. Try  
Wrong! Please enter the correct current password and pincode. Try  
Wrong! Please enter the correct current password and pincode. Try  
Wrong! Please enter the correct current password and pincode. Try  
Wrong! Please enter the correct current password and pincode. Try  
Wrong! Please enter the correct current password and pincode. Try  
Correct!  
The password of user bandit25 is iCi86ttT4KSNe1armKiwbQNmb3YJP3q4  
bandit24@bandit:~$ |
```

**Nivel 26: Como escapar de una terminal rara

- *Objetivo del nivel*

Debemos iniciar sesión en Bandit26 desde Bandit25, pero en lugar de usar /bin/bash, la cuenta usa una shell diferente.

- ** _Comandos utilizados

- *Explicación paso a paso

Como ya mencionamos anteriormente, la shell del usuario Bandit26 es diferente a las que hemos venido trabajando, para saber que tipo de shell usa podemos hacer lo siguiente: cat /etc/passwd | grep "bandit26" , con este comando le estamos diciendo a la terminal que muestre el contenido del archivo /etc/passwd en la terminal. Que contiene información sobre los usuarios del sistema, ID de usuario, ID de grupo, directorio personal y el tipo de shell que usan.

```
bandit26:x:11026:11026:bandit level 26:/home/bandit26:/usr/bin/showtext
```

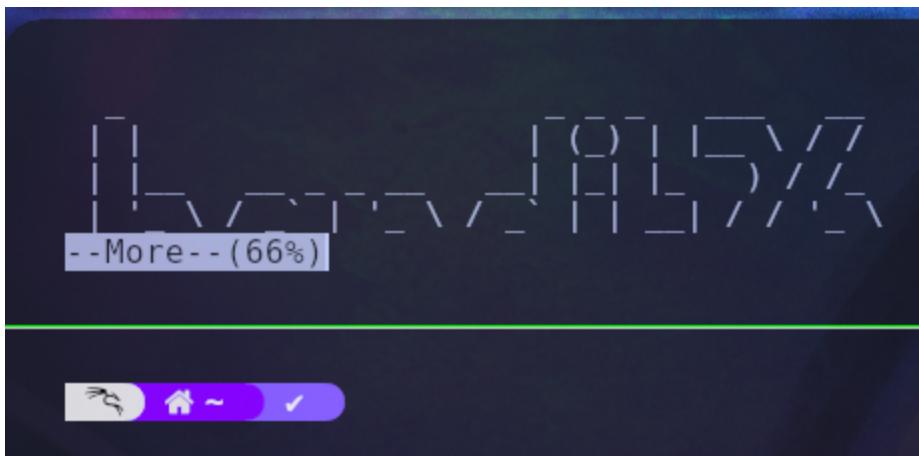
Podemos ver que esta usando la shell showtest , si hacemos cat a esa ruta vemos lo siguiente:

```
bandit25@bandit:~$ cat /usr/bin/showtext
#!/bin/sh

export TERM=linux

exec more ~/text.txt
exit 0
```

Un script que se ejecuta al momento de abrir la shell , y usa el comando more para abrir el archivo txt. El comando ‘more’ lo que nos permite es mostrar el resultado de la ejecución de un comando en la terminal de a una página a la vez. Esto es especialmente útil para casos en los que se ejecutan comandos que puedan llegar a causar un gran desplazamiento. Entonces como tenemos un more, lo que tenemos que hacer es disminuir el tamaño de nuestra terminal.



Posteriormente presionamos `v` para entrar en el modo visual, que nos permitirá ingresar instrucciones.

Luego presionamos `Esc` , y luego :



Ahora, podemos ingresar el comando `set shell=/bin/bash` que nos permitirá usar la bash para ingresar los comandos siendo bandit26.

Siendo ahora bandit26, si hacemos un `ls` encontramos estos dos archivos.

```
:shell
bandit26@bandit:~$ ls
bandit27-do  text.txt
```

**Nivel 27: Apresúrate

- *Objetivo del nivel*

Siendo bandit26, debemos apurarnos y conseguir la flag para bandit27, si no lo logramos la conexión se cerrara.

- **_Comandos utilizados

1. `ls -l`
2. `./bandit27-do cat /etc/bandit_pass/bandit27`

- *Explicación paso a paso

En bandit26, al hacer un `ls` obtenemos `text.txt`, y `bandit27-do`.

```
bandit26@bandit:~$ ls -l
total 20
-rwsr-x--- 1 bandit27 bandit26 14884 Apr 10 14:23 bandit27-do
-rw-r----- 1 bandit26 bandit26    258 Apr 10 14:23 text.txt
bandit26@bandit:~$ |
```

Vemos que el archivo `bandit27-do` es SUID, y como el propietario es bandit27 nosotros podemos ejecutar comandos como el propietario de forma temporal.

```
bandit26@bandit:~$ ./bandit27-do cat /etc/bandit_pass/bandit27
upsNCc7vzaRDx6oZC6GiR6ERwe1MowGB
```

**Nivel 28: Hurgando en Git

- *Objetivo del nivel*

Existe un repositorio Git en la ruta que se nos da, debemos clonar el repositorio y encontrar la contraseña para bandit28. El usuario y la contraseña son los mismos que los de Bandit27.

- **_Comandos utilizados

1. `git clone`: Clona el repositorio que le mandemos.
2. `mktemp -d`

- *Explicación paso a paso

Comenzamos creando un directorio temporal con el comando `mktemp -d`, para después meternos a ese directorio y clonarnos el repositorio con el comando `git clone`

```
bandit27@bandit:/tmp/tmp.TeiIgZTkT$ git clone ssh://bandit27-git@localhost:2220/home/bandit27-git/repo
Cloning into 'repo'...
The authenticity of host '[localhost]:2220 ([127.0.0.1]:2220)' can't be established.
ED25519 key fingerprint is SHA256:C2ihUBV7ihnVlwUXRb4RrEcLfxC5CXlhmAAM/urerLY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Could not create directory '/home/bandit27/.ssh' (Permission denied).
Failed to add the host to the list of known hosts (/home/bandit27/.ssh/known_hosts).
```



```
This is an OverTheWire game server.
More information on http://www.overthewire.org/wargames
```

```
bandit27-git@localhost's password: |
```

Nos dicen que la contraseña es la misma que usamos para ingresar a bandit27.

```
bandit27-git@localhost's password:
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
bandit27@bandit:/tmp/tmp.TeiIgZTkT$ ls
repo
bandit27@bandit:/tmp/tmp.TeiIgZTkT$ cd repo
bandit27@bandit:/tmp/tmp.TeiIgZTkT/repo$ ls
README
bandit27@bandit:/tmp/tmp.TeiIgZTkT/repo$ cat
^Z
[1]+  Stopped                  cat
bandit27@bandit:/tmp/tmp.TeiIgZTkT/repo$ cat README
The password to the next level is: Yz9IpL0sBcCeug7m9uQFt8ZNpS4HZRcN
bandit27@bandit:/tmp/tmp.TeiIgZTkT/repo$ |
```

Importante: Clonar de la siguiente manera -> git clone ssh://bandit27-git@localhost:2220/home/bandit27-git/repo

**Nivel 29: Hurgando en Git parte 2

- *Objetivo del nivel*

Es el mismo objetivo solo que ahora el repositorio pertenece a bandit28

- Comandos utilizados

1. `git clone`: Clona el repositorio que le mandemos.
2. `mktemp -d`
3. `git log`
4. `git show`

- *Explicación paso a paso

En GitHub hay algo conocido que se llaman commits, estos aparecen solo si le hacemos cambios a un proyecto ya existente que tengamos en nuestro Git. Con el comando `git log` podemos ver los commits que ha tenido el proyecto.

```
commit 674690a00a0056ab96048f7317b9ec20c057c06b (HEAD -> master, origin/master, origin/HEAD)
Author: Morla Porla <morla@overthewire.org>
Date:   Thu Apr 10 14:23:19 2025 +0000

    fix info leak

commit fb0df1358b1ff146f581651a84bae622353a71c0
Author: Morla Porla <morla@overthewire.org>
Date:   Thu Apr 10 14:23:19 2025 +0000

    add missing data

commit a5fdc97aae2c6f0e6c1e722877a100f24bc当地
Author: Ben Dover <noone@overthewire.org>
Date:   Thu Apr 10 14:23:19 2025 +0000

    initial commit of README.md
bandit28@bandit:/tmp/tmp.nqAZBURcwc/repo$
```

Si vemos el fix info leak o fuga de información, arriba de el podemos el commit o el identificador de ese commit. Si lo queremos ver desde consola podemos hacer un `git show` para ver el cambio

```
bandit28@bandit:/tmp/tmp.nqAZBURcwc/repo$ git show 674690a00a0056ab96048f7317b9ec20c057c06b
WARNING: terminal is not fully functional
Press RETURN to continue
commit 674690a00a0056ab96048f7317b9ec20c057c06b (HEAD -> master, origin/master, origin/HEAD)
Author: Morla Porla <morla@overthewire.org>
Date:   Thu Apr 10 14:23:19 2025 +0000

    fix info leak

diff --git a/README.md b/README.md
index d4e3b74..5c6457b 100644
--- a/README.md
+++ b/README.md
@@ -4,5 +4,5 @@ Some notes for level29 of bandit.
## credentials

- username: bandit29
-- password: 4pT1t5DENaYuqnqvadYs1oE4QLCdjJ7
+- password: xxxxxxxxxxxx

bandit28@bandit:/tmp/tmp.nqAZBURcwc/repo$
```

**Nivel 30: Hurgando en Git parte 3

- *Objetivo del nivel*

Es el mismo objetivo solo que ahora el repositorio pertenece a bandit29

- ** Comandos utilizados

1. `git branch -a`

2. git checkout

- ***Explicación paso a paso**

Una rama en Git es una línea de desarrollo independiente en un repositorio. Permite trabajar en cualquier cambio sin afectar la rama principal. Para ver ramas podemos hacer un `git branch -a`

```
bandit29@bandit:/tmp/tmp.NkfuKrQcXb/repo
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev
  remotes/origin/master
  remotes/origin/sploits-dev
bandit29@bandit:/tmp/tmp.NkfuKrQcXb/repo
```

En general, la rama "dev" en un proyecto Git se utiliza como una rama de desarrollo, lo que significa que es posible que la contraseña este allí. Sin embargo, esto siempre depende de la configuración específica del proyecto. En el caso de OverTheWire, se eligió mostrar la contraseña en la rama "dev" con fines de enseñanza y para hacerlo más fácil de encontrar, con el comando `git checkout dev`.

```
bandit29@bandit:/tmp/tmp.NkfuKrQcXb/repo$ git checkout dev
branch 'dev' set up to track 'origin/dev'.
Switched to a new branch 'dev'
bandit29@bandit:/tmp/tmp.NkfuKrQcXb/repo$ |
```

```
# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

- username: bandit30
- password: qp30ex3VLz5MDG1n91YowTv4Q8l7CDZL

bandit29@bandit:/tmp/tmp.NkfuKrQcXb/repo$ |
```

qp30ex3VLz5MDG1n91YowTv4Q8l7CDZL

**Nivel 31: Hurgando en Git parte 4

- **Objetivo del nivel**

Es el mismo objetivo solo que ahora el repositorio pertenece a bandit30

- ** _Comandos utilizados

1. `git tag`

2. git show

- *Explicación paso a paso

```
bandit30@bandit:/tmp/tmp.1qpn1s3vax/repo$ ls  
README.md  
bandit30@bandit:/tmp/tmp.1qpn1s3vax/repo$ cat README.md  
just an empty file... muahaha  
bandit30@bandit:/tmp/tmp.1qpn1s3vax/repo$ |
```

Aquí entra el juego el concepto de tags en git, que sirve básicamente como una rama que siempre se mantiene inalterable. Sirve para marcar puntos en la historia de nuestro repositorio. Con el comando git tag podemos mostrar las tags, y con git show aqui_va_la_tag , podemos ver el contenido de esa tag como se ve en la imagen.

```
bandit30@bandit:/tmp/tmp.1qpn1s3vax/repo$ git tag  
secret  
bandit30@bandit:/tmp/tmp.1qpn1s3vax/repo$ git show secret  
fb5S2xb7bRyFmAvQYQGEqsbhVyJqhnDy  
bandit30@bandit:/tmp/tmp.1qpn1s3vax/repo$ |
```

fb5S2xb7bRyFmAvQYQGEqsbhVyJqhnDy

**Nivel 32: Hurgando en Git parte 5

- *Objetivo del nivel*

Es el mismo objetivo solo que ahora el repositorio pertenece a bandit31

- **_Comandos utilizados

1. git add -f key.txt .
2. git push -u origin master

- *Explicación paso a paso

```
Details:  
  File name: key.txt  
  Content: 'May I come in?'  
  Branch: master
```

Se nos dice que tenemos que crearnos un archivo llamado key.txt , y ahí meter el contenido que se nos dice. Básicamente que hagamos un commit.

Si queremos añadir un archivo podemos aplicar el comando git add -f key.txt .

A nivel de commits debemos añadirle un texto descriptivo:

```
bandit31@bandit:/tmp/tmp.ohq3x9Rryg/repo$ git commit -m "Nuevo Archivo"
[master 211cded] Nuevo Archivo
 1 file changed, 1 insertion(+)
 create mode 100644 key.txt
bandit31@bandit:/tmp/tmp.ohq3x9Rryg/repo$ |
```

Y ahora tenemos que subirlo para que en el proyecto se actualice y haya un nuevo commit, esto lo logramos mediante el comando `git push -u origin master` porque se nos dice que es en la rama master.

```
bandit31-git@localhost's password:
Permission denied, please try again.
bandit31-git@localhost's password:
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 327 bytes | 
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote: ### Attempting to validate files.
remote:
remote: .o0o.o0o.o0o.o0o.o0o.o0o.o0o.o0o.
remote:
remote: Well done! Here is the password for the next level:
remote: 309RfhqyAlVBEZpVb6LYStshZoqoSx5K
remote:
remote: .o0o.o0o.o0o.o0o.o0o.o0o.o0o.o0o.
remote:
To ssh://localhost:2220/home/bandit31-git
 ! [remote rejected] master -> master (pre-receive hook declined)
error: failed to push some refs to 'ssh://localhost:2220/home/bandit31-git'
bandit31@bandit:/tmp/tmp.ohq3x9Rryg/repo$ |
```

**Nivel 33: Escapa y conquista el ultimo desafío

- *Objetivo del nivel*

Nuestro desafío final es escapar de un entorno limitado.

- ** _Comandos utilizados
- *Explicación paso a paso

La UPPERCASE SHELL es una capa, pero por detrás de ella corre una shell, al usar el comando \$0 dentro de UPPERCASE SHELL este ejecuta una nueva instancia de la shell que va por detrás que no tiene restricciones el cual es una sh, ya estando en la sh puedes ejecutar una nueva instancia de shell que sea bash.

```
$ whoami  
bandit33  
$ bash  
bandit33@bandit:~$
```

```
bandit33@bandit:~$ pwd  
/home/bandit32  
bandit33@bandit:~$ cd ..  
bandit33@bandit:/home$ ls  
bandit0 bandit13 bandit18 bandit22 bandit27 bandit29-git bandit31-git bandit6 drifter1 drifter15 drifter6 formulaone1 krypton1 krypton6  
bandit1 bandit14 bandit19 bandit23 bandit27-git bandit3 bandit32 bandit7 drifter2 drifter7 formulaone2 krypton2 krypton7  
bandit10 bandit15 bandit2 bandit24 bandit28 bandit30 bandit33 bandit8 drifter12 drifter3 drifter8 formulaone3 krypton3 ubuntu  
bandit11 bandit16 bandit20 bandit25 bandit28-git bandit30-git bandit4 bandit9 drifter13 drifter4 drifter9 formulaone5 krypton4  
bandit12 bandit17 bandit21 bandit26 bandit29 bandit31 bandit5 drifter0 drifter14 drifter5 formulaone0 formulaone6 krypton5  
bandit33@bandit:/home$ cd bandit33]
```

```
! REUSE-DISK YOUR ADMINISTRATOR!  
bandit33@bandit:/home/bandit33$ cat README.txt  
Congratulations on solving the last level of this game!
```

At this moment, there are no more levels to play in this game. However, we are constantly working on new levels and will most likely expand this game with more levels soon. Keep an eye out for an announcement on our usual communication channels! In the meantime, you could play some of our other wargames.

If you have an idea for an awesome new level, please let us know!
bandit33@bandit:/home/bandit33\$ |

- **⚠_Errores comunes y cómo evitarlos**

1. No usar `cat` – directamente porque la terminal lo interpreta como un argumento y queda esperando más entrada.
2. SSH solo necesita que el archivo exista, y muy importante: **Que tenga los permisos correctos, es decir, `chmod 600`**.
3. No usar comillas para nombres de archivo con espacios
4. Intentar leer archivos ocultos sin usar `-a` o `-la`
5. No filtrar errores al usar `find`
6. No usar permisos adecuados en claves privadas
7. No especificar el puerto en SSH

4. Conclusión y Reflexiones

- Durante el desarrollo de esta guía, reforcé mis conocimientos sobre el sistema operativo Linux, el uso de comandos avanzados en terminal, la manipulación de archivos, los permisos de sistema y conceptos básicos de red. Este proyecto me ayudó a comprender cómo se estructuran los retos en ciberseguridad, cómo abordarlos paso a paso, y la importancia de automatizar tareas.
- Ahora ya podemos empezar con Hack The Box, Try Hack Me, etc