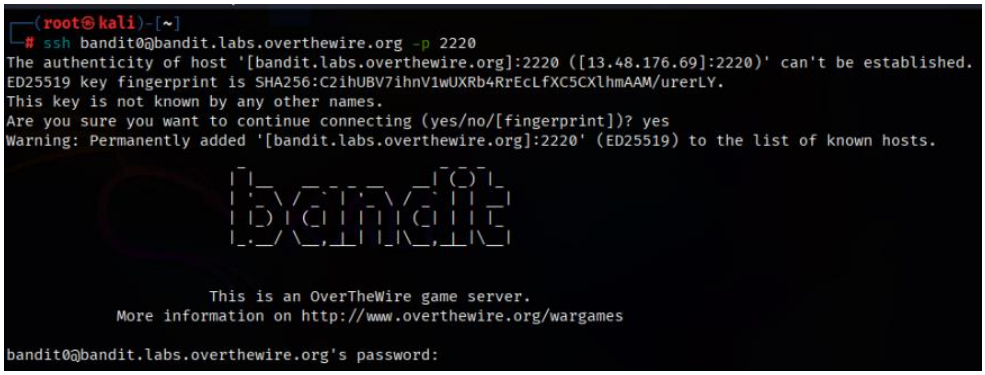
Bandit Level 0

Level Goal: 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.

Bandit Level 0 → Level 1

Step 1: Use ssh to login



Step 2: file listing, then printing content:

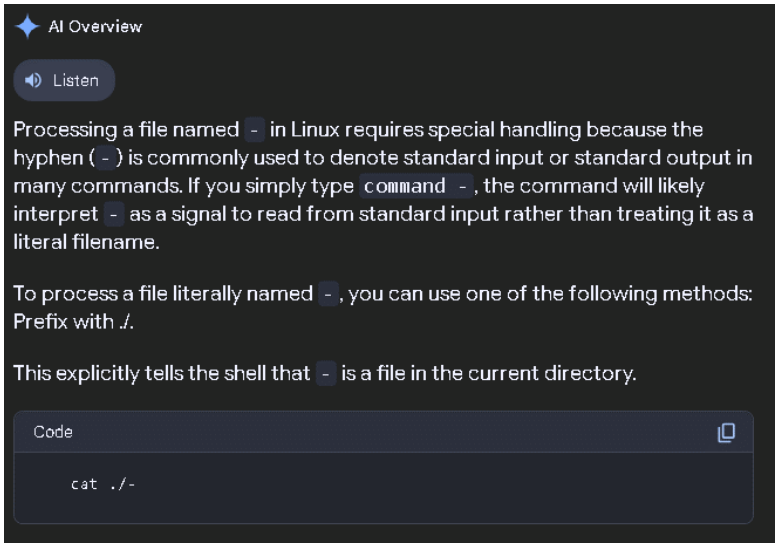
Password for Level 2 is **ZjLjTmM6FvvyRnrb2rfNWOZOTa6ip5If**

https://manpages.ubuntu.com/manpages/noble/man1/

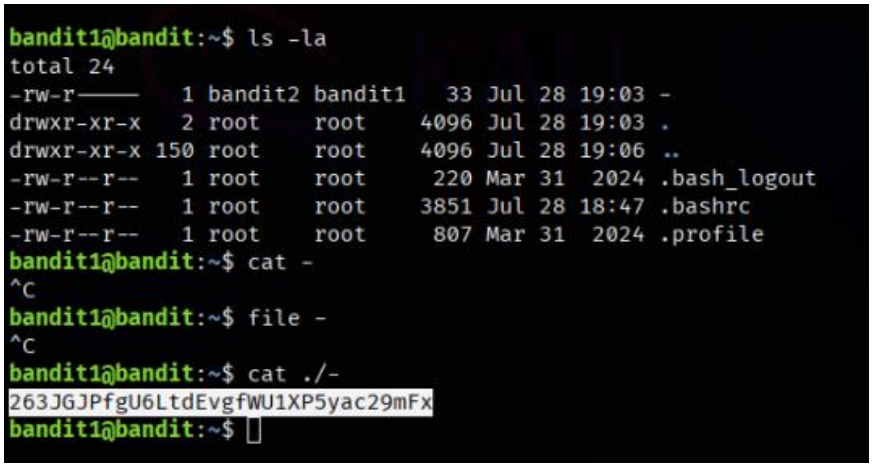
Bandit Level 1 → Level 2

Level Goal

The password for the next level is stored in a file called - located in the home directory



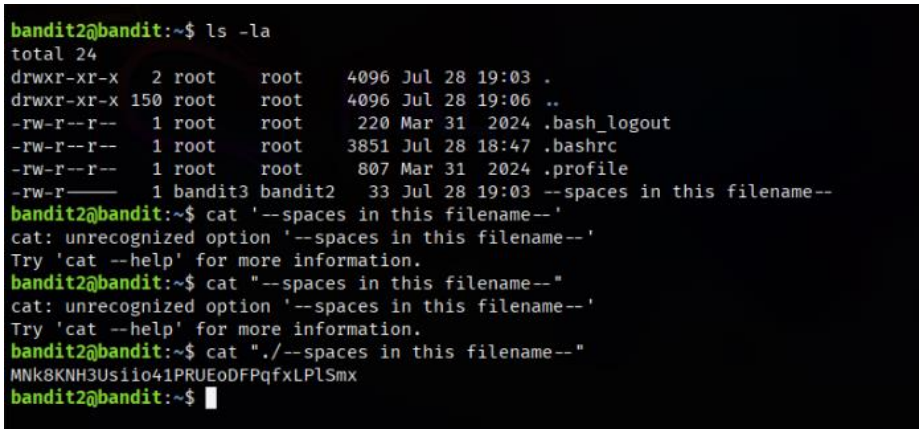
List all file usinf ls –la then use cat to see content:



Password for level 2 **263JGJPfgU6LtdEvgfWU1XP5yac29mFx**

Bandit Level 2 → Level 3

Level Goal: The password for the next level is stored in a file called –spaces in this filename– located in the home directory

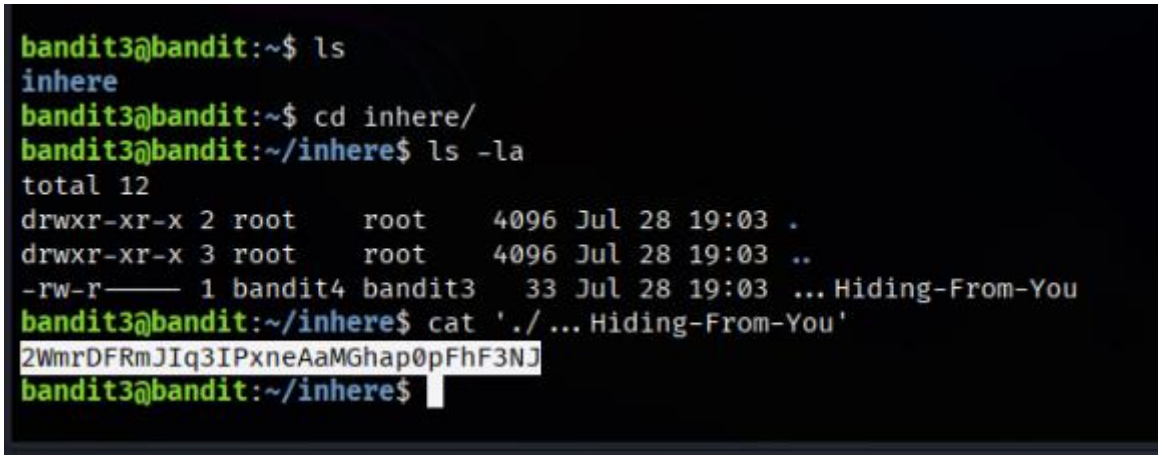


Password for level 3 is **MNk8KNH3Usiio41PRUEoDFPqfxLPlSmx**

Bandit Level 3 → Level 4

Level Goal

The password for the next level is stored in a hidden file in the inhere directory.



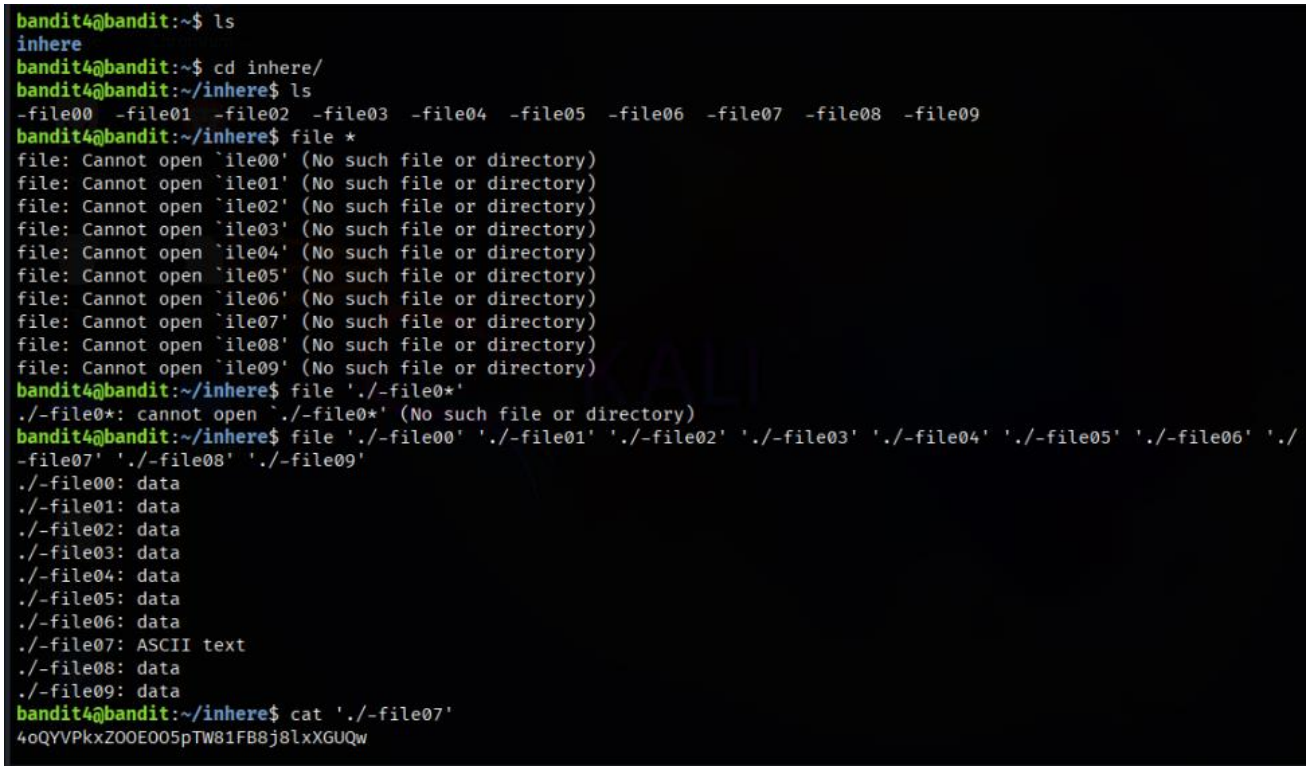
Password for level 4 is **2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ**

Bandit Level 4 → Level 5

Level Goal

The password for the next level is stored in the only human-readable file in the inhere directory.

Tip: if your terminal is messed up, try the “reset” command.



Password for level 5 is **4oQYVPkxZOOEOO5pTW81FB8j8lxXGUQw**

Bandit Level 5 → Level 6

Level Goal

The password for the next level is stored in a file somewhere under the inhere directory and has all of the following properties:

human-readable

1033 bytes in size

not executable

first we use file command to check which file is executable and which is human readable.

Command: find -type f -size 1033c ! -executable -exec file {} \; | grep "text"

- find -type f → search for files.
- -size 1033c → exactly 1033 bytes.
- ! -executable → not executable.
- -exec file {} \; → run file on each match to check if it's human-readable text.
- grep "text" → keep only text files.

```
bandit5@bandit:~/inhere$ ls
maybehere00  maybehere03  maybehere06  maybehere09  maybehere12  maybehere15  maybehere18
maybehere01  maybehere04  maybehere07  maybehere10  maybehere13  maybehere16  maybehere19
maybehere02  maybehere05  maybehere08  maybehere11  maybehere14  maybehere17
bandit5@bandit:~/inhere$
```

```
bandit5@bandit:~/inhere$ find -type f -size 1033c ! -executable -exec file {} \; | grep "text"
./maybehere07/.file2: ASCII text, with very long lines (1000)
bandit5@bandit:~/inhere$ cat ./maybehere07/.file2
HWasnPhtq9AVKe0dmk45nxy20cvUa6EG
```

Password for level 6 is **HWasnPhtq9AVKe0dmk45nxy20cvUa6EG**

Bandit Level 6 → Level 7

Level Goal: The password for the next level is stored somewhere on the server and has all of the following properties:

owned by user bandit7

owned by group bandit6

33 bytes in size

find / -type f -user bandit7 -group bandit6 -size 33c 2>/dev/null

Explanation:

/ → search from root directory.

-type f → files only.

-user bandit7 → owner is user bandit7.

-group bandit6 → group is bandit6.

-size 33c → exactly 33 bytes.

2>/dev/null → hide permission denied errors.

```
bandit6@bandit:~$ ls
bandit6@bandit:~$ ls -ls
total 0
bandit6@bandit:~$ ls -la
total 20
drwxr-xr-x   2 root root 4096 Jul 28 19:03 .
drwxr-xr-x 150 root root 4096 Jul 28 19:06 ..
-rw-r--r--   1 root root  220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root root 3851 Jul 28 18:47 .bashrc
-rw-r--r--   1 root root  807 Mar 31  2024 .profile
bandit6@bandit:~$ find / -type f -user bandit7 -group bandit6 -size 33c 2>/dev/null
/var/lib/dpkg/info/bandit7.password
bandit6@bandit:~$ cat ^C
bandit6@bandit:~$ ^C
bandit6@bandit:~$ cat /var/lib/dpkg/info/bandit7.password
morbNTDkSW6jIlUc0ymOdMaLnOlFVAaj
bandit6@bandit:~$ ^C
```

Password for level 7 **morbNTDkSW6jIlUc0ymOdMaLnOlFVAaj**

Bandit Level 7 → Level 8

Level Goal

The password for the next level is stored in the file data.txt next to the word millionth

du -b data.txt

cat data.txt | grep millionth

```
bandit7@bandit:~$ ls
data.txt
bandit7@bandit:~$ du -b data.txt
4184396 data.txt
bandit7@bandit:~$ cat data.txt | grep millionth
millionth       dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc
bandit7@bandit:~$
```

Password for level 8 is **dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc**

Bandit Level 8 → Level 9

Level Goal

The password for the next level is stored in the file data.txt and is the only line of text that occurs only once

grep - print lines that match patterns

sort - sort lines of text files

uniq - report or omit repeated lines

strings - print the sequences of printable characters in files

base64 - base64 encode/decode data and print to standard output

tr - translate or delete characters

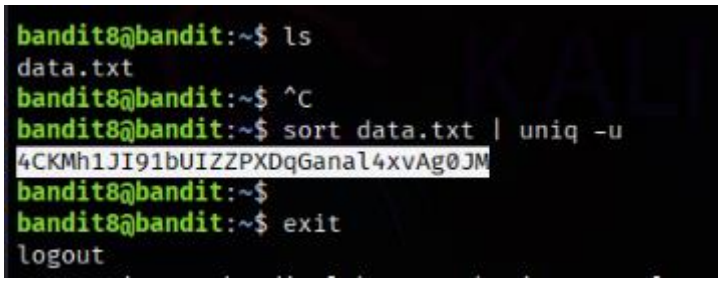
tar - an archiving utility

gzip - compress or expand files

bzip2 - a block-sorting file compressor, v1.0.8

xxd - make a hexdump or do the reverse.

sort data.txt | uniq –u
sort → puts identical lines next to each other.
uniq -u → prints only the lines that occur once.

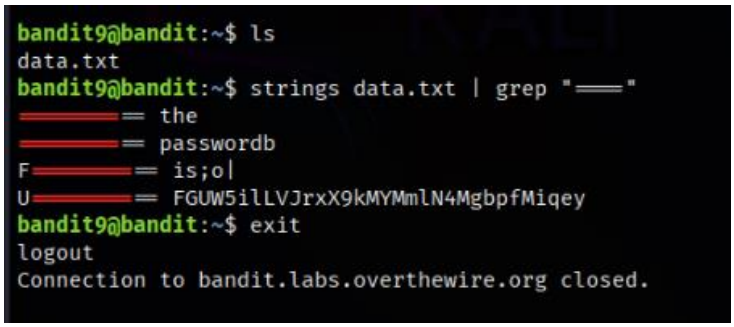


Password for level 9 is **4CKMh1JI91bUIZZPXDqGanal4xvAg0JM**

Bandit Level 9 → Level 10

Level Goal: The password for the next level is stored in the file data.txt in one of the few human-readable strings, preceded by several '=' characters.

strings data.txt | grep "===="

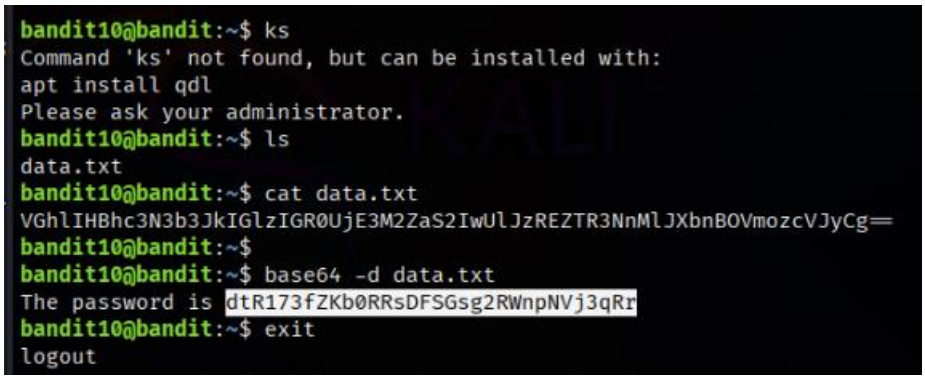


Password for level 10 is **FGUW5ilLVJrxX9kMYMmlN4MgbpfMiqey**

Bandit Level 10 → Level 11

Level Goal

The password for the next level is stored in the file data.txt, which contains base64 encoded data



Password for level 11 is **dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr**

Bandit Level 11 → Level 12

Level Goal

The password for the next level is stored in the file data.txt, where all lowercase (a-z) and uppercase (A-Z) letters have been rotated by 13 positions

ROT13

ROT13 = “rotate by 13 places”.

It’s a substitution cipher where each letter is replaced by the one 13 places ahead in the alphabet.

It works for both uppercase and lowercase:

- a → n, n → a
- A → N, N → A

Numbers, punctuation, and spaces are not changed.

Since the English alphabet has 26 letters, applying ROT13 twice gives you back the original text.

Command: cat data.txt | tr 'A-Za-z' 'N-ZA-Mn-za-m'

```
bandit11@bandit:~$ ls
data.txt
bandit11@bandit:~$ cat data.txt | tr 'A-Za-z' 'N-ZA-Mn-za-m'
The password is 7×16WNeHIi5YkIhWsfFIqoognUTyj9Q4
bandit11@bandit:~$ cat data.txt
Gur cnffjbeq vf 7k16JArUVv5LxVuJfsSVdbbtaHGlw9D4
bandit11@bandit:~$ exit
logout
Connection to bandit.labs.overthewire.org closed.
```

Password for level 12 is **7x16WNeHIi5YkIhWsfFIqoognUTyj9Q4**

Bandit Level 12 → Level 13

Level Goal

The password for the next level is stored in the file data.txt, which is a hexdump of a file that has been repeatedly compressed. For this level it may be useful to create a directory under /tmp in which you can work. Use mkdir with a hard to guess directory name. Or better, use the command “mktemp -d”. Then copy the datafile using cp, and rename it using mv (read the manpages!)

Step1: make directory in tmp directory, because we don’t have permission to make file in current directory

```
bandit12@bandit:~$ file data.txt
data.txt: ASCII text
bandit12@bandit:~$ mkdir /tmp/toni
bandit12@bandit:~$ cd /tmp/toni
bandit12@bandit:/tmp/toni$ ls
bandit12@bandit:/tmp/toni$ cp ~/data.txt .
bandit12@bandit:/tmp/toni$ ls
data.txt
bandit12@bandit:/tmp/toni$
```

Step2: convert the hexdump back into binary

```
bandit12@bandit:/tmp/toni$ xxd -r data.txt data.bin
bandit12@bandit:/tmp/toni$ ls
data.bin  data.txt
```

Step3: check new file, it was compressed with gzip, so decompress it:

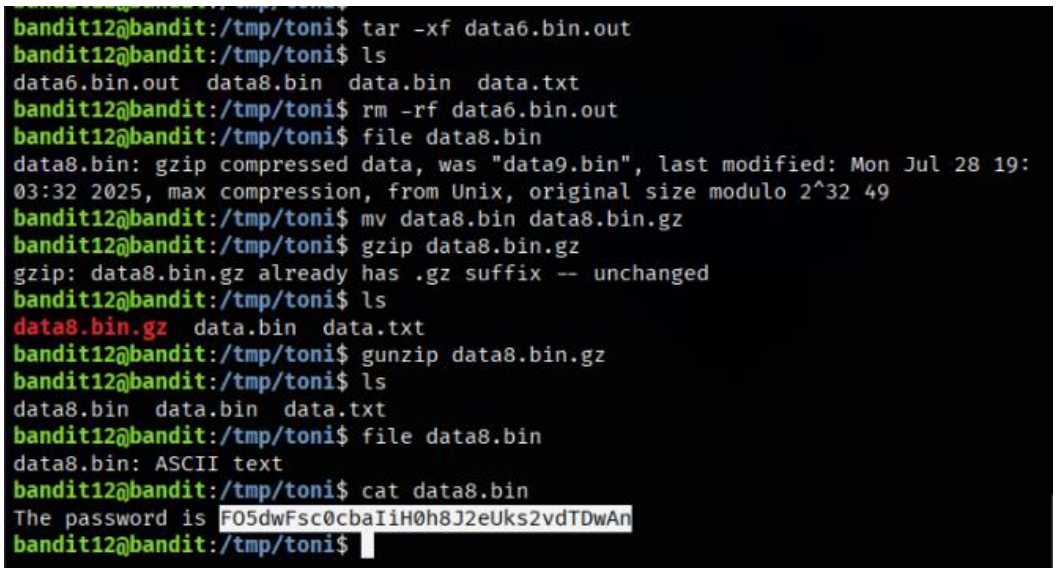
```
bandit12@bandit:/tmp/toni$ file data.bin
data.bin: gzip compressed data, was "data2.bin", last modified: Mon Jul 28 19:0
 original size modulo 2^32 578
bandit12@bandit:/tmp/toni$
bandit12@bandit:/tmp/toni$
bandit12@bandit:/tmp/toni$ file data.bin
data.bin: gzip compressed data, was "data2.bin", last modified: Mon Jul 28 19:0
3:32 2025, max compression, from Unix, original size modulo 2^32 578
bandit12@bandit:/tmp/toni$ gzip -d data.bin
gzip: data.bin: unknown suffix -- ignored
bandit12@bandit:/tmp/toni$ ls
data.bin  data.txt
bandit12@bandit:/tmp/toni$ file data.bin
data.bin: gzip compressed data, was "data2.bin", last modified: Mon Jul 28 19:0
3:32 2025, max compression, from Unix, original size modulo 2^32 578
bandit12@bandit:/tmp/toni$ mv data.bin data.bin.gz
bandit12@bandit:/tmp/toni$ ls
data.bin.gz  data.txt
bandit12@bandit:/tmp/toni$ gunzip data.bin.gz
bandit12@bandit:/tmp/toni$ ls
data.bin  data.txt
bandit12@bandit:/tmp/toni$ file data.bin
data.bin: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/toni$
```

Step4: Now it showing it is compressed with bzip2 let's decompressed it:

```
bandit12@bandit:/tmp/toni$ bzip2 -d data.bin
bzip2: Can't guess original name for data.bin -- using data.bin.out
bandit12@bandit:/tmp/toni$ ls
data.bin.out  data.txt
bandit12@bandit:/tmp/toni$ file data.bin.out
data.bin.out: gzip compressed data, was "data4.bin", last modified: Mon Jul 28
19:03:32 2025, max compression, from Unix, original size modulo 2^32 20480
bandit12@bandit:/tmp/toni$
```

Step5: Again decompresses until we found what we want

```
bandit12@bandit:/tmp/toni$ mv data.bin.out data.bin.gz
bandit12@bandit:/tmp/toni$ ls
data.bin.gz  data.txt
bandit12@bandit:/tmp/toni$ file data.bin.gz
data.bin.gz: gzip compressed data, was "data4.bin", last modified: Mon Jul 28 1
9:03:32 2025, max compression, from Unix, original size modulo 2^32 20480
bandit12@bandit:/tmp/toni$ gunzip data.bin.gz
bandit12@bandit:/tmp/toni$ ls
data.bin  data.txt
bandit12@bandit:/tmp/toni$ file data.bin
data.bin: POSIX tar archive (GNU)
bandit12@bandit:/tmp/toni$ tar -xf data.bin
bandit12@bandit:/tmp/toni$ ls
data5.bin  data.bin  data.txt
bandit12@bandit:/tmp/toni$ file data5.bin
data5.bin: POSIX tar archive (GNU)
bandit12@bandit:/tmp/toni$ tar -xf data5.bin
bandit12@bandit:/tmp/toni$ ls
data5.bin  data6.bin  data.bin  data.txt
bandit12@bandit:/tmp/toni$ rm -rf data5.bin
bandit12@bandit:/tmp/toni$ file data6.bin
data6.bin: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/toni$ bzip2 -d data6.bin
bzip2: Can't guess original name for data6.bin -- using data6.bin.out
bandit12@bandit:/tmp/toni$ ls
data6.bin.out  data.bin  data.txt
bandit12@bandit:/tmp/toni$ file data6.bin.out
data6.bin.out: POSIX tar archive (GNU)
bandit12@bandit:/tmp/toni$
```



Password for level 13 is **FO5dwFsc0cbaIiH0h8J2eUks2vdTDwAn**

Command used:
File: to check which compression is used
Gunzip: decompress - gzip compressed data
tar –xf: decompress POSIX tar archive
bzip2 -d: decompress bzip2 compressed data
xxd –r : convert hexadecimal data to binary data

Bandit Level 13 → Level 14

Level Goal
The password for the next level is stored in /etc/bandit_pass/bandit14 and can only be read by user bandit14. For this level, you don't get the next password, but you get a private SSH key that can be used to log into the next level. Note: localhost is a hostname that refers to the machine you are working on

Here's what is happening:

1. Key pair: sshkey.private is the private half of an SSH key pair. The matching public key is (already) stored on the server in ~bandit14/.ssh/authorized_keys.
2. Client offers public key: When you attempt to connect, your SSH client advertises which public keys it can use.
3. Server checks authorized_keys: The SSH server checks whether the offered public key is allowed for the target user (exists in authorized_keys).
4. Proof of possession: If the public key matches, the server sends a challenge that only the corresponding private key can sign. Your client uses sshkey.private to sign the challenge and returns the response.
5. Authentication success: If the signature verifies, the server accepts you and opens the shell as bandit14.

To do this we will use this command:
ssh -i sshkey.private bandit14@localhost -p 2220

```
bandit13@bandit:~$ ls
sshkey.private
bandit13@bandit:~$ ssh -i sshkey.private -p 2220
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface] [-b bind_address]
           [-c cipher_spec] [-D [bind_address:]port] [-E log_file]
           [-e escape_char] [-F configfile] [-I pkcs11] [-i identity_file]
           [-J destination] [-L address] [-l login_name] [-m mac_spec]
           [-O ctl_cmd] [-o option] [-P tag] [-p port] [-R address]
           [-S ctl_path] [-W host:port] [-w local_tun[:remote_tun]]
           destination [command [argument ...]]
       ssh [-Q query_option]
bandit13@bandit:~$ ls -la
total 24
drwxr-xr-x   2 root     root     4096 Jul 28 19:03 .
drwxr-xr-x 150 root     root     4096 Jul 28 19:06 ..
-rw-r--r--   1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root     3851 Jul 28 18:47 .bashrc
-rw-r--r--   1 root     root      807 Mar 31  2024 .profile
-rw-r-----   1 bandit14 bandit13 1679 Jul 28 19:03 sshkey.private
bandit13@bandit:~$ chmod 600 sshkey.private
chmod: changing permissions of 'sshkey.private': Operation not permitted
bandit13@bandit:~$ ssh -i sshkey.private bandit14@localhost -p 2220
The authenticity of host '[localhost]:2220 ([127.0.0.1]:2220)' can't be establi
shed.
ED25519 key fingerprint is SHA256:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/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/bandit13/.ssh' (Permission denied).
Failed to add the host to the list of known hosts (/home/bandit13/.ssh/known_ho
sts).
```

```
bandit14@bandit:~$ cat /etc/bandit_pass/bandit14
MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS
bandit14@bandit:~$
```

Password for level 14 is **MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS**

Bandit Level 14 → Level 15

Level Goal: The password for the next level can be retrieved by submitting the password of the current level to port 30000 on localhost.

nc is alias for the netcat.

netcat (aka nc) is the swiss-army knife of TCP/UDP networking: it can open raw TCP/UDP connections, listen for incoming connections, transfer files, do simple port scans, grab banners, and be used for debugging or quick ad-hoc services.

Here we will use it to connect to the localhost's port 30000.

```
bandit14@bandit:~$ ls
bandit14@bandit:~$ nc localhost 30000
MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS
Correct!
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
```

Password for level 15 is **8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo**

Bandit Level 15 → Level 16

Level Goal

The password for the next level can be retrieved by submitting the password of the current level to port 30001 on localhost using SSL/TLS encryption.

Helpful note: Getting “DONE”, “RENEGOTIATING” or “KEYUPDATE”? Read the “CONNECTED COMMANDS” section in the manpage.

Here I used previous command but nothing returned, so check password:

```
bandit15@bandit:~$ nc localhost 30001
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
bandit15@bandit:~$
bandit15@bandit:~$ cat /etc/bandit_pass/bandit15
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
```

But password is correct so we have to use another command,

In goal it saying using ssl/tls encryption. Let’s find out:

Why nc localhost 30001 fails:

- nc (Netcat) just connects via TCP and sends bytes as-is.
- Port 30001 expects an SSL/TLS handshake before sending/receiving any data.
- Without the handshake, the server sees garbage and closes the connection.

How **openssl s_client** works

- openssl is a cryptography toolkit.
- s_client subcommand lets you act like an SSL/TLS client.
- -connect localhost:30001 tells it to connect to that host:port and start the TLS handshake.
- Once the handshake is done, the connection is encrypted, and you can type/send data securely.

```
bandit15@bandit:~$ openssl s_client -connect localhost:30001
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = SnakeOil
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN = SnakeOil
verify return:1
---
Certificate chain
 0 s:CN = SnakeOil
   i:CN = SnakeOil
   a:PKEY: rsaEncryption, 4096 (bit); sigalg: RSA-SHA256
   v:NotBefore: Jun 10 03:59:50 2024 GMT; NotAfter: Jun  8 03:59:50 2034 GMT
---
Server certificate
```

```
    Start Time: 1755081189
    Timeout   : 7200 (sec)
    Verify return code: 18 (self-signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
Correct!
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx

closed
```

Password for level 16 is **kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx**

Bandit Level 16 → Level 17

Level Goal

The credentials for the next level can be retrieved by submitting the password of the current level to a port on localhost in the range 31000 to 32000. First find out which of these ports have a server listening on them. Then find out which of those speak SSL/TLS and which don't. There is only 1 server that will give the next credentials, the others will simply send back to you whatever you send to it.

```
bandit16@bandit:~$ nmap -sV -p 31000-32000 -T4 localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-08-16 13:22 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00051s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT      STATE SERVICE     VERSION
31046/tcp open  echo
31518/tcp open  ssl/echo
31691/tcp open  echo
31790/tcp open  ssl/unknown
31960/tcp open  echo
1 service unrecognized despite returning data. If you know the service/version, pl
ease submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-s
ervice :
SF-Port31790-TCP:V=7.94SVN%T=SSL%I=7%D=8/16%Time=68A08624%P=x86_64-pc-linu
SF:x-gnu%r(GenericLines,32,"Wrong!\x20Please\x20enter\x20the\x20correct\x2
SF:0current\x20password\.\n")%r(GetRequest,32,"Wrong!\x20Please\x20enter\x
SF:20the\x20correct\x20current\x20password\.\n")%r(HTTPOptions,32,"Wrong!\
SF:x20Please\x20enter\x20the\x20correct\x20current\x20password\.\n")%r(RTS
SF:PRequest,32,"Wrong!\x20Please\x20enter\x20the\x20correct\x20current\x20
SF:password\.\n")%r(Help,32,"Wrong!\x20Please\x20enter\x20the\x20correct\x
SF:20current\x20password\.\n")%r(FourOhFourRequest,32,"Wrong!\x20Please\x2
SF:0enter\x20the\x20correct\x20current\x20password\.\n")%r(LPDString,32,"W
SF:rong!\x20Please\x20enter\x20the\x20correct\x20current\x20password\.\n")
SF:%r(SIPOptions,32,"Wrong!\x20Please\x20enter\x20the\x20correct\x20curren
SF:t\x20password\.\n");

Service detection performed. Please report any incorrect results at https://nmap.o
rg/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 145.50 seconds
bandit16@bandit:~$
```

As we can see in image port 31790 is the one we are looking for

```
bandit16@bandit:~$ openssl s_client --connect localhost:31790
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = SnakeOil
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN = SnakeOil
verify return:1
---
```

```
---
read R BLOCK
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
KEYUPDATE
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
KEYUPDATE
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
KEYUPDATE
```

As we can see it didn't giving us the password for the next level despite submitting the correct password, what is the problem:

I found that by default, OpenSSL runs in interactive mode. This means entering anything with the characters Q, R, K, and k will be treated as commands and our password is starting **K**.

CONNECTED COMMANDS ¶

If a connection is established with an SSL server then any data received from the server is displayed and any key presses will be sent to the server. If end of file is reached then the connection will be closed down. When used interactively (which means neither **-quiet** nor **-ign_eof** have been given), then certain commands are also recognized which perform special operations. These commands are a letter which must appear at the start of a line. They are listed below.

- Q

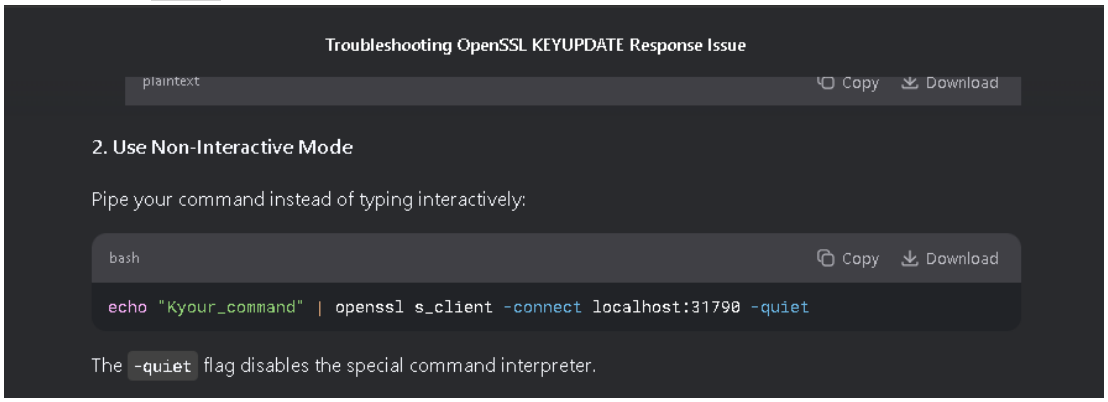
  End the current SSL connection and exit.
- R

  Renegotiate the SSL session (TLSv1.2 and below only).
- k

  Send a key update message to the server (TLSv1.3 only)
- K

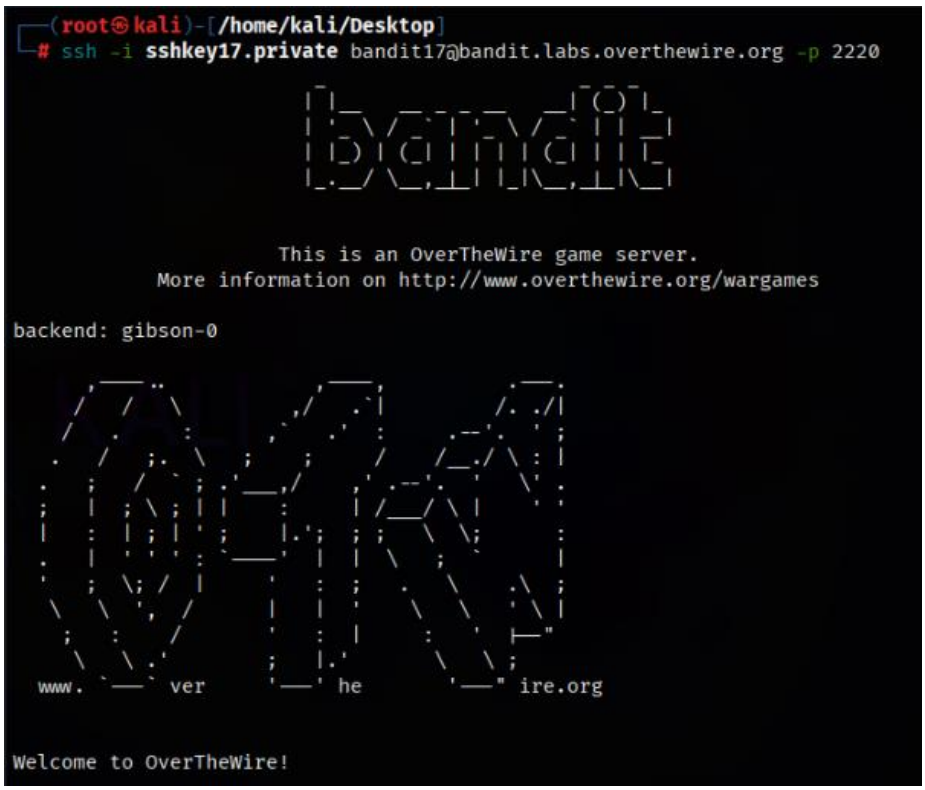
  Send a key update message to the server and request one back (TLSv1.3 only)

Solution: use –quiet switch




```
bandit16@bandit:~$ openssl s_client -quiet -connect localhost:31790
Can't use SSL_get_servername
depth=0 CN = SnakeOil
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN = SnakeOil
verify return:1
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
Correct!
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAvmOkuifmMg6HL2YPIOjon6iWfbp7c3jx34YkYWqUH57SUdyJ
imZzeyGC0gtZPGujUSxiJSWI/oTqexh+cAMTSMlOJf7+BrJObArnxd9Y7YT2bRPQ
Ja6Lzb558YW3FZl87ORiO+rW4LCDCNd2lUvLE/GL2GWyuKN0K5iCd5TbtJzEkQTu
DSt2mcNn4rhAL+JFr56o4T6z8WWAW18BR6yGrMq7Q/kALHYW3OekePQAzL0VUYbW
JGTi65CxbCnzc/w4+mqQyvmzpWtMAzJTzAzQxNbkR2MBGySxDLrjg0LWN6sK7wNX
x0YVztz/zbIkPjfkU1jHS+9EbVNj+D1XFOJuaQIDAQABAoIBABagpxpM1aoLWfvD
KHcj10nqcoBc4oE11aFYQwik7xfW+24pRNuDE6SFthOar69jp5RllwD1NhPx3iBl
J9nOM8OJ0VToum43UOS8YxF8WwhXriYGnc1sskbwpXOUDc9uX4+UESzH22P29ovd
d8WErY0gPxun8pbJLmxkAtWNhpMvfe0050vk9TL5wqbu9AlbssgTcCXkMQnPw9nC
YNN6DDP2lbcBrvgT9YCNL6C+ZKufD52yOQ9qOkwFTEQpjtF4uNtJom+asvlpmS8A
vLY9r60wYSvmZhNqBUrj7lyCtXMIu1kkd4w7F77k+DjHoAXyxcUp1DGL51sOmama
+TOWWgECgYEA8JtPxP0GRJ+IQkX262jM3dEIkza8ky5moIwUqYdsx0NxHgRRhORT
8c8hAuRBb2G82so8vUHk/fur85OEfc9TncnCY2crpoqsghifKLxrLgtT+qDpfZnx
SatLdt8GfQ85yA7hnWWJ2MxF3NaeSDm75Lsm+tBbAiyc9P2jGRNtMSkCgYEAypHd
HCctNi/FwjulhttFx/rHYKhLidZDFYeiE/v45bN4yFm8x7R/b0iE7KaszX+Exdvt
SghaTdcG0Knyw1bpJVyusavPzpaJMjdJ6tcFhVAbAjm7enCIvGCSx+X3l5SiWg0A
R57hJglezIiVjv3aGwHwvlZvtszK6zV6oXFAu0ECgYAbjo46T4hyP5tJi93V5HDi
Ttiek7xRVxUl+iU7rWkGAXFpMLFteQEsRr7PJ/lemmEY5eTDAFMLy9FL2m9oQWCg
R8VdwSk8r9FGLS+9aKcV5PI/WEKlwgXinB3OhYimtiG2Cg5JCqIZFHxD6MjEGOiu
L8ktHMPvodBwNsSBULpG0QKBgBAplTfC1HOnWiMGOU3KPwYWt0O6CdTkmJOmL8Ni
blh9elyZ9FsGxsgtRBXRsqXuz7wtsQAgLHxbdLq/ZJQ7YfzOKU4ZxEnabvXnvWkU
YOdjHdSOoKvDQNWu6ucyLRAWFuISeXw9a/9p7ftpxm0TSgyvmfLF2MIAEwyzRqaM
```

Ok, so after entering password it will give us rsa private key which we can use to login to the bendit17. For this we gave to make a key file and save this rsa into it, let’s do it and login:



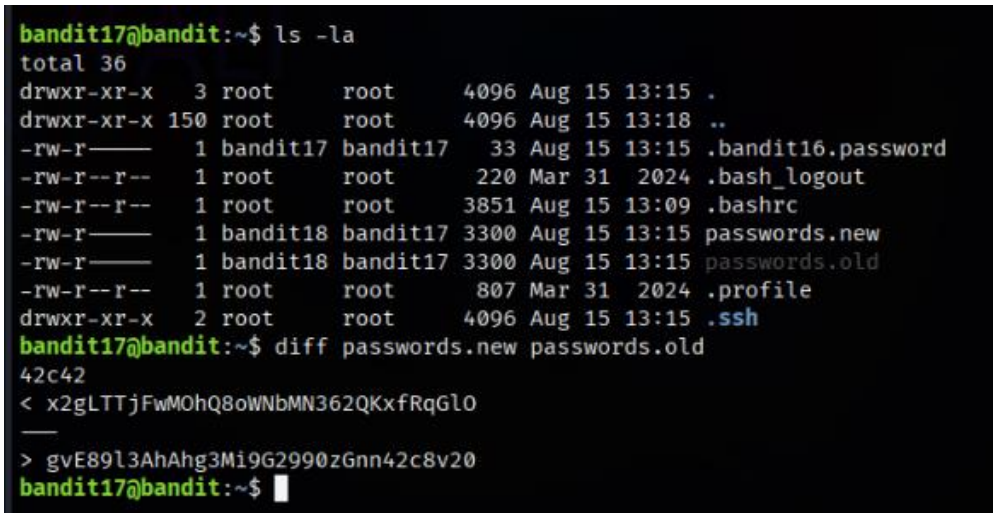
And success.

Password for level 17 is **EReVavePLFHtFlFsjn3hyzMlvSuSAcRD**

Bandit Level 17 → Level 18

Level Goal

There are 2 files in the homedirectory: passwords.old and passwords.new. The password for the next level is in passwords.new and is the only line that has been changed between passwords.old and passwords.new



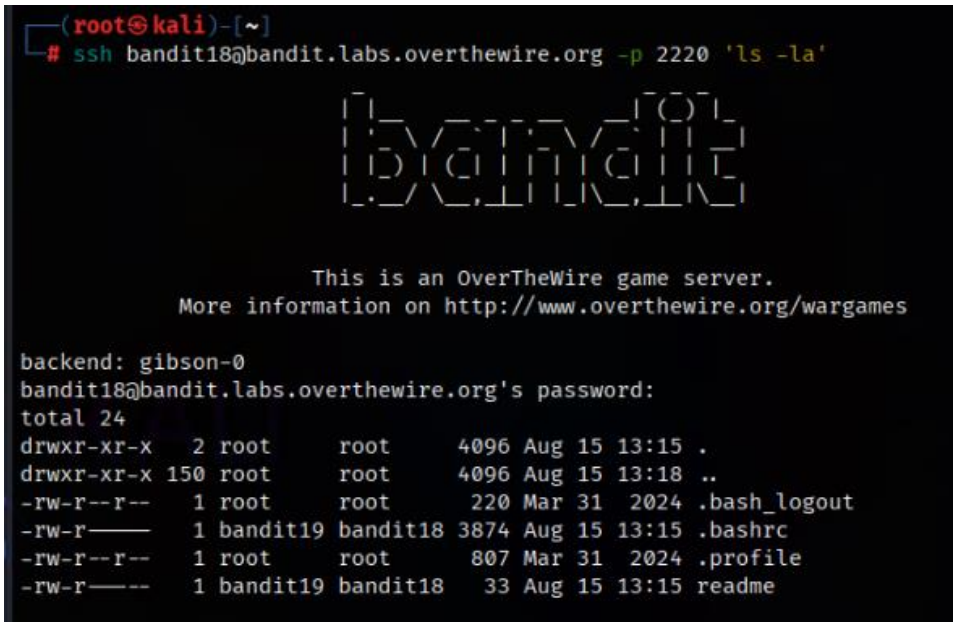
Password for level 18 is **x2gLTTjFwMOhQ8oWNbMN362QKxfRqGlO**

Bandit Level 18 → Level 19

Level Goal

The password for the next level is stored in a file readme in the home directory. Unfortunately, someone has modified .bashrc to log you out when you log in with SSH.

- Here the .bashrc made in a way so that anyone try to login to the Server they will log out immediately.
- So, the solution is to run command along with ssh command so that it will automatically execute after authentication and then we will logout.



Password for level 19 is **cGWpMaKXVwDUNgPAVJbWYuGHVn9zl3j8**

Bandit Level 19 → Level 20

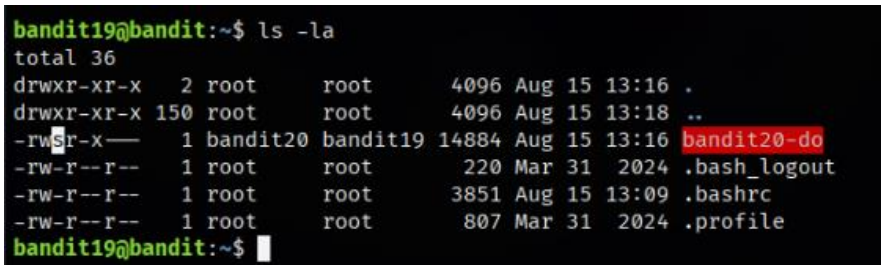
Level Goal

To gain access to the next level, you should use the setuid binary in the home directory. Execute it without arguments to find out how to use it. The password for this level can be found in the usual place (/etc/bandit_pass), after you have used the setuid binary.

setuid and **setgid** (short for set user identity and set group identity)

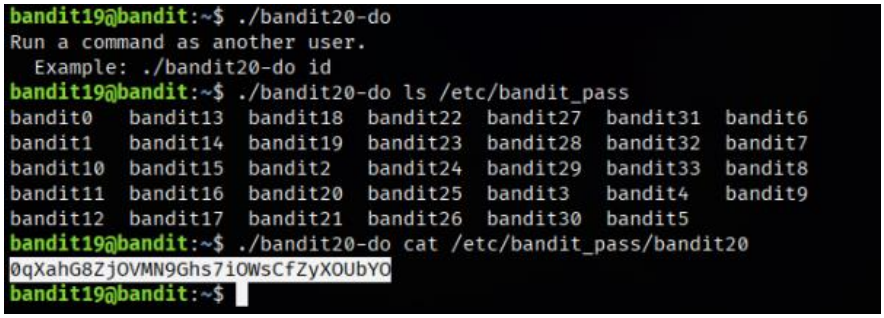
https://en.wikipedia.org/wiki/Setuid

- Setuid (SUID) bit on a binary lets anyone who runs it execute that program with the file-owner's privileges. If the binary is owned by root and has SUID, running it runs with root privileges see below:



- We find bandit20-do owned by user bandit20 and group bandit19 with permissions -rwsr-x---.
- The filename has red background to indicate that it is executable with setuid permission set.
- The s indicates setuid permission, and the x indicates that we, as userbandit19, which is part of group bandit19, can execute the file.
- Because this setuid file is owned by bandit20, if we (bandit19) execute it, it will run with bandit20's privileges.

running setuid binary without arguments should show usage.



Password for level 20 is **0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO**

Bandit Level 20 → Level 21

Level Goal

There is a setuid binary in the home directory that does the following: it makes a connection to localhost on the port you specify as a commandline argument. It then reads a line of text from the connection and compares it to the password in the previous level (bandit20). If the password is correct, it will transmit the password for the next level (bandit21).

As per instruction:

we need to set up a listener that sends the current level password.

We can do that using netcat

- echo '0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO': String to send when a connection occurs.

- nc -l -p 12345: Use netcat to listen on port 12345.
- &: Run it in the background so we can keep using the current terminal.

To confirm the listener is set up properly, we can use jobs command:

```
bandit20@bandit:~$ echo '0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO' | nc -l -p 12345 &
[1] 3068253
bandit20@bandit:~$ jobs
[1]+  Done                    echo '0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO' | nc -l -
p 12345
bandit20@bandit:~$
```

```
bandit20@bandit:~$ ls -la
total 36
drwxr-xr-x   2 root     root      4096 Aug 15 13:16 .
drwxr-xr-x 150 root     root      4096 Aug 15 13:18 ..
-rw-r--r--   1 root     root       220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root      3851 Aug 15 13:09 .bashrc
-rw-r--r--   1 root     root       807 Mar 31  2024 .profile
-rwsr-x---   1 bandit21 bandit20 15608 Aug 15 13:16 suconnect
bandit20@bandit:~$ ./suconnect
Usage: ./suconnect <portnumber>
This program will connect to the given port on localhost using TCP. If it recei
ves the correct password from the other side, the next password is transmitted
back.
bandit20@bandit:~$ ./suconnect 12345
Could not connect
bandit20@bandit:~$ ./suconnect 12345
Could not connect
bandit20@bandit:~$ jobs
bandit20@bandit:~$ echo '0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO' | nc -l -p 12345 &
[1] 3072679
bandit20@bandit:~$ ./suconnect 12345
Read: 0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO
Password matches, sending next password
EeoULMCra2q0dSkYj561DX7s1CpBuOBt
bandit20@bandit:~$
```

Password for level 21 is **EeoULMCra2q0dSkYj561DX7s1CpBuOBt**

Bandit Level 21 → Level 22

Level Goal

A program is running automatically at regular intervals from cron, the time-based job scheduler.

Look in /etc/cron.d/ for the configuration and see what command is being executed.

- We see multiple files, and considering we're looking for the Bandit level 22 password, let's read cronjob_bandit22

```
bandit21@bandit:~$ cd /etc/cron.d/
bandit21@bandit:/etc/cron.d$ ls
behemoth4_cleanup  cronjob_bandit23  leviathan5_cleanup    sysstat
clean_tmp          cronjob_bandit24  manpage3_resetpw_job
cronjob_bandit22   e2scrub_all       otw-tmp-dir
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$
```

It contains two lines:

- @reboot bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null: This means user bandit22 will run the script cronjob_bandit22.sh every time the system reboots. /dev/null means all output will be hidden.
- * * * * bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null: This is the same as above, with the only difference being the occurrence. The * * * * * represents minute, hour, day of the month, month, and day of the week. Because all values are asterisks, it means this cron job will run every minute of every hour, day, month, and day of the week.

Let's see the content of the script run by this cron job:

```
bandit21@bandit:/etc/cron.d$ cat /usr/bin/cronjob_bandit22.sh
#!/bin/bash
chmod 644 /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
cat /etc/bandit_pass/bandit22 > /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
bandit21@bandit:/etc/cron.d$ cat /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
tRae0UfB9v0UzbCdn9cY0gQnds9GF58Q
bandit21@bandit:/etc/cron.d$
```

And we got the password.
Password for level 22 is **tRae0UfB9v0UzbCdn9cY0gQnds9GF58Q**

Bandit Level 22 → Level 23

Level Goal
A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in /etc/cron.d/ for the configuration and see what command is being executed.
We will do same as previous level,

```
bandit22@bandit:~$ cd /etc/cron.d/
bandit22@bandit:/etc/cron.d$ ls
behemoth4_cleanup  cronjob_bandit23  leviathan5_cleanup    sysstat
clean_tmp          cronjob_bandit24  manpage3_resetpw_job
cronjob_bandit22   e2scrub_all       otw-tmp-dir
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$ cat /tmp/$mytarget
cat: /tmp/: Permission denied
bandit22@bandit:/etc/cron.d$
```

And permission denied.
Here's what is happening in .sh file:

- The last line (cat /etc/...) means it copy a password to a file in /tmp.
- It uses two variables, $myname and $mytarget. We need to find the values to understand which password is copied and to which file.
- The value of $myname is the output of the whoami command, which returns the current username.
- If we use it ourselves, it will return bandit22, but because this cron job is meant to be executed by bandit23, the value in this script will be bandit23.

```
bandit22@bandit:/etc/cron.d$ whoami
bandit22
```

- mytarget variable contains $myname, which we know has value bandit23.
- It hashes this value using MD5, and because the output will contain a space and a hyphen, it uses cut -d ' ' -f 1 to get rid of that.
- so the last line of cronjob_bandit23.sh will be /tmp/md5 output, let's see:

```
bandit22@bandit:/etc/cron.d$ echo I am user bandit23 | md5sum
8ca319486bfbbc3663ea0fbe81326349  -
bandit22@bandit:/etc/cron.d$ echo I am user bandit23 | md5sum | cut -d ' ' -f 1
8ca319486bfbbc3663ea0fbe81326349
bandit22@bandit:/etc/cron.d$ cat /tmp/8ca319486bfbbc3663ea0fbe81326349
0Zf11ioIjMVN551jX3CmStKLYqjk54Ga
bandit22@bandit:/etc/cron.d$
```

Password for level 23 is **0Zf11ioIjMVN551jX3CmStKLYqjk54Ga**

Bandit Level 23 → Level 24

Level Goal

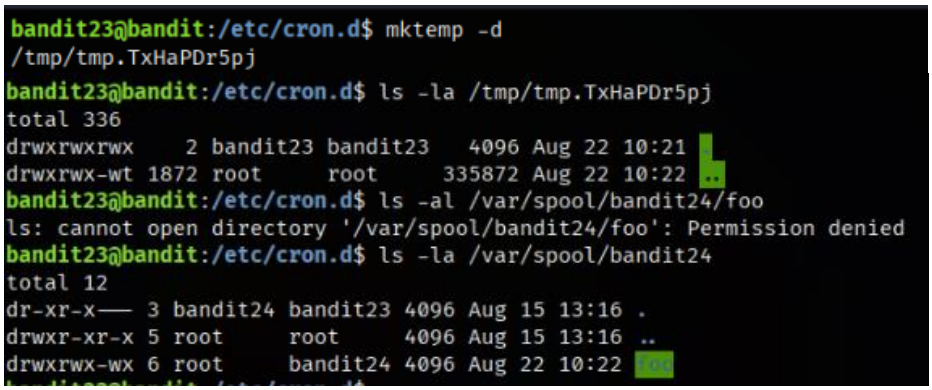
A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in /etc/cron.d/ for the configuration and see what command is being executed.

```
bandit23@bandit:~$ cd /etc/cron.d
bandit23@bandit:/etc/cron.d$ ls
behemoth4_cleanup  cronjob_bandit23  leviathan5_cleanup    sysstat
clean_tmp          cronjob_bandit24  manpage3_resetpw_job
cronjob_bandit22   e2scrub_all       otw-tmp-dir
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
```

Script analyzing:

- myname=$(whoami) => Because this cron job will be run by user bandit24, the output of the whoami command will be bandit24.
- cd /var/spool/$myname/foo => Move to directory /var/spool/bandit24/foo.
- for i in * .*; do => Loop through all files in this directory.
- if ["$i" != "." -a "$i" != ".."]; then => Exclude the current directory (.) and parent directory (..) from this loop.
- owner="$(stat — format "%U" ./$i)" => Use stat command to obtain the file's owner and assign it to the variable owner.

- if [“${owner}” = “bandit23”]; then timeout -s 9 60 ./$i fi => If the owner is bandit23, execute the file. If execution takes longer than 60 seconds, kill the process by sending signal number 9 (In case the executed script has issues).
- rm -f ./$i => Remove the executed file.
- In summary, it executes all files in /var/spool/$myname/foo owned by bandit23 and then deletes them.
- Our current user is bandit23, so we can write a script that copies the level 24 password file (it should work because this cron job will be executed as bandit24).
- For the destination, we can put it in /tmp, where we likely have write access. Let's do that in the next step. (mktemp: create unique dir in /tmp dir.)



As we see in image we don’t have read access to /foo dir of bendit24 user, but we have write and execute access, now we will make script to retrieve password:

- The command adds the shebang (#!/bin/bash) to the file GetPassword24.sh. If the file didn't exist, it will be automatically created, so we don't have to make it manually.
- The second command copies the password to pass_24.txt. Just like before, if the file didn't exist, it will be automatically created.
- The third command gives the cron job (bandit24) permission to execute our script.

And we got our password.

Password for level 24 is **gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8**

Bandit Level 24 → Level 25

Level Goal

A daemon is listening on port 30002 and will give you the password for bandit25 if given the password for bandit24 and a secret numeric 4-digit pincode. There is no way to retrieve the pincode except by going through all of the 10000 combinations, called brute-forcing.

You do not need to create new connections each time

The server reads lines from a single TCP connection: if you pipe all password + pin attempts (one per line) into nc localhost 30002 the server will read them sequentially on the same connection and reply when it finds the correct PIN. This avoids reconnecting 10,000 times.

```
bandit24@bandit:~$ echo r6r87r6t | nc localhost 30002
I am the pincode checker for user bandit25. Please enter the password for user
bandit24 and the secret pincode on a single line, separated by a space.
Wrong! Please enter the correct current password and pincode. Try again.
```

Ai gave me this 2 command after submitting prompt and the output.

If you want to save the service response (the bandit25 password) to a file:

bash — Copy — Edit

```
p=$(cat /etc/bandit_pass)
for i in $(seq -w 0 9999); do echo "$p $i"; done | nc localhost 30002 > bandit25.txt
```

Alternative using `printf` (same effect):

bash — Copy — Edit

```
p=$(cat /etc/bandit_pass)
for i in $(seq 0 9999); do printf "%s %04d\n" "$p" "$i"; done | nc localhost 30002
```

In short, the command above establishes a connection using netcat, then sends a sequence of passwords with PINs from 0000 to 9999. And both failed miserably so I change my command little bit.

```
bandit24@bandit:~$ for i in $(seq 0 9999); do printf "gb8KRRCsshuZXI0tUuR6ypOFj
iZbf3G8 %04d\n" "$i"; done | nc localhost 30002
I am the pincode checker for user bandit25. Please enter the password for user
bandit24 and the secret pincode on a single line, separated by a space.
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.
Correct!
The password of user bandit25 is iCi86ttT4KSNe1armKiwbQNmB3YJP3q4
```

And worked.

Password for level 25 is **iCi86ttT4KSNe1armKiwbQNmB3YJP3q4**

Bandit Level 25 → Level 26

Level Goal

Logging in to bandit26 from bandit25 should be fairly easy… The shell for user bandit26 is not /bin/bash, but something else. Find out what it is, how it works and how to break out of it.

```
bandit25@bandit:~$ ls -la
total 40
drwxr-xr-x   2 root     root     4096 Aug 15 13:16 .
drwxr-xr-x 150 root     root     4096 Aug 15 13:18 ..
-rw-r-----   1 bandit25 bandit25   33 Aug 15 13:16 .bandit24.password
-r--------   1 bandit25 bandit25 1679 Aug 15 13:16 bandit26.sshkey
-rw-r-----   1 bandit25 bandit25  151 Aug 15 13:16 .banner
-rw-r--r--   1 root     root      220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root     3851 Aug 15 13:09 .bashrc
-rw-r-----   1 bandit25 bandit25   66 Aug 15 13:16 .flag
-rw-r-----   1 bandit25 bandit25    4 Aug 15 13:16 .pin
-rw-r--r--   1 root     root      807 Mar 31  2024 .profile
bandit25@bandit:~$ ssh -i bandit26.sshkey bandit26@localhost -p 2220
The authenticity of host '[localhost]:2220 ([127.0.0.1]:2220)' can't be establi
shed.
ED25519 key fingerprint is SHA256:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/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/bandit25/.ssh' (Permission denied).
Failed to add the host to the list of known hosts (/home/bandit25/.ssh/known_ho
sts).
```

Found sshkey so tried to logging in but exited immediately.

```
  Enjoy your stay!

  _                     _ _ _   ___   __
 | |                   | (_) | |__ \ / /
 | |__   __ _ _ __   __| |_| |_   ) / /_
 | '_ \ / _` | '_ \ / _` | | __| / / '_ \
 | |_) | (_| | | | | (_| | | |_ / /| (_) |
 |_.__/ \__,_|_| |_|\__,_|_|\__|____\___/
Connection to localhost closed.
bandit25@bandit:~$
```

this happens because it uses a different shell, so we need to find out what shell used by reading /etc/passwd

```
bandit25@bandit:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
bandit21:x:11021:11021:bandit level 21:/home/bandit21:/bin/bash
bandit22:x:11022:11022:bandit level 22:/home/bandit22:/bin/bash
bandit23:x:11023:11023:bandit level 23:/home/bandit23:/bin/bash
bandit24:x:11024:11024:bandit level 24:/home/bandit24:/bin/bash
bandit25:x:11025:11025:bandit level 25:/home/bandit25:/bin/bash
bandit26:x:11026:11026:bandit level 26:/home/bandit26:/usr/bin/showtext
bandit27:x:11027:11027:bandit level 27:/home/bandit27:/bin/bash
bandit28:x:11028:11028:bandit level 28:/home/bandit28:/bin/bash
```

It's using /usr/bin/showtext shell, let's read it

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

export TERM=linux

exec more ~/text.txt
exit 0
bandit25@bandit:~$
```

- #!/bin/sh: This shebang indicates that the following code should be run by the shell.

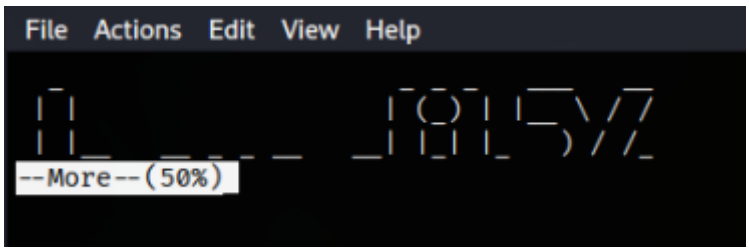
- export TERM=linux: This sets the terminal emulator, which determines the appearance of the shell (font, color, shortcuts, etc.).
- exec more ~/text.txt: This replaces the current shell process with the more command. The more command is similar to cat, except it displays content one screen at a time, depending on the screen size, which is convenient for large documents.
- exit 0: This terminates the process after the user finishes viewing the file, which is why we are logged out immediately after logging in.
- The tricky part is the third line, which replaces the shell with a single command. This means we can't use other commands like cat, ls, or chmod, and we can't even open other files because it is hardcoded for text.txt.
- Additionally, since this code replaces the default shell in /etc/passwd, starting a new session or resetting the system won't solve the issue.
- Before finding a way to use another shell, we need to find a way to prevent being logged out. Let's try it in the next step.

```
bandit25@bandit:~$ cat /home/bandit26/text.txt
cat: /home/bandit26/text.txt: Permission denied
bandit25@bandit:~$ ls -la /home/bandit26
total 44
drwxr-xr-x   3 root     root      4096 Aug 15 13:16 .
drwxr-xr-x 150 root     root      4096 Aug 15 13:18 ..
-rwsr-x---   1 bandit27 bandit26 14884 Aug 15 13:16 bandit27-do
-rw-r--r--   1 root     root       220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root      3851 Aug 15 13:09 .bashrc
-rw-r--r--   1 root     root       807 Mar 31  2024 .profile
drwxr-xr-x   2 root     root      4096 Aug 15 13:16 .ssh
-rw-r-----   1 bandit26 bandit26   258 Aug 15 13:16 text.txt
```

When we try reading the file, we receive a "permission denied" response. Using ls -al, we find that only bandit26 can read the file.

Remember, the more command shows content one screen at a time, depending on the screen size. If we make the terminal shorter than text.txt, we might be able to keep it open.

The ASCII art for bandit26 consists of only 6 lines, so we need to make our terminal really short and then try logging in again.



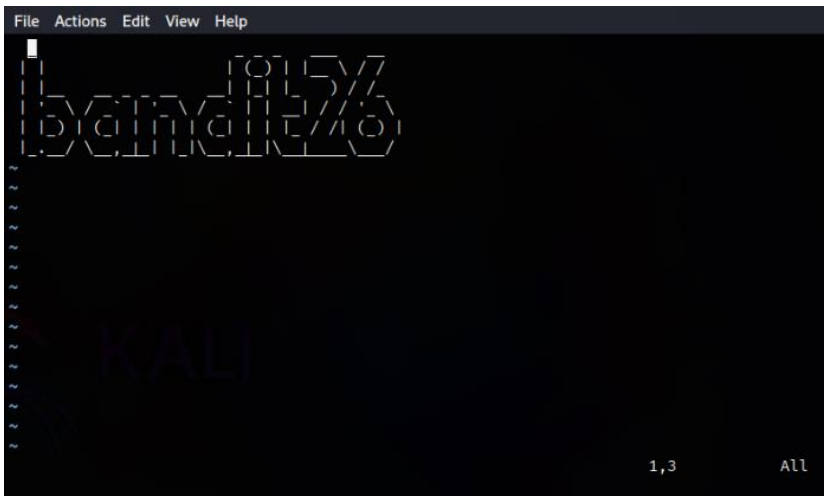
It worked, we didn't logged out.

The ASCII art for bandit26 consists of only 6 lines, so we need to make our terminal really short

Now how to get out of this more command without being logged out?

```
!command or :!command
    Execute command in a subshell.

v
    Start up an editor at current line. The editor is taken from
    the environment variable VISUAL if defined, or EDITOR if
    VISUAL is not defined, or defaults to vi(1) if neither VISUAL
    nor EDITOR is defined.
```

As per manual we can enter in visual editor by pressing v, then we can increase our screen size:

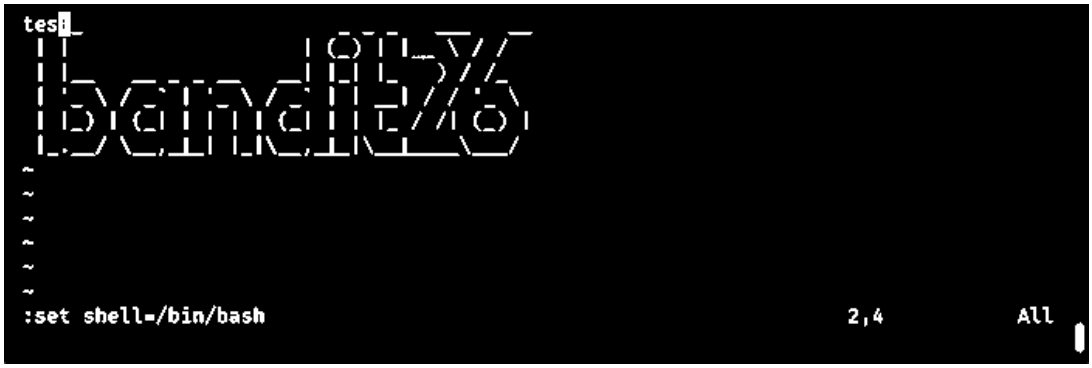


As per manual we can execute command using ':'

From GTFOBins, we find that vi has a feature to change shell it use using :set shell command.

https://gtfobins.github.io/gtfobins/vi/ (this website popular for it's privilege escalation usage)

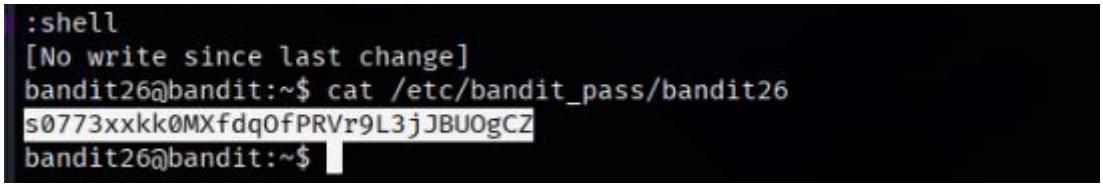
We change the shell with :set shell=/bin/bash and then press Enter



If you enter in insert mode, press Esc to enter command mode again.

Next, we type :shell to spawn it.

Now we see a shell we are familiar with. We can obtain bandit26's password by going to /etc/bandit_pass/bandit26.



Password for level 26 is **s0773xxkk0MXfdqOfPRVr9L3jJBUOgCZ**

Bandit Level 26 → Level 27

Level Goal: Good job getting a shell! Now hurry and grab the password for bandit27!

In this level process is similar to level 20

```
bandit26@bandit:~$ ls -ls
total 20
16 -rwsr-x--- 1 bandit27 bandit26 14884 Aug 15 13:16 bandit27-do
 4 -rw-r----- 1 bandit26 bandit26   258 Aug 15 13:16 text.txt
bandit26@bandit:~$ Connection to bandit.labs.overthewire.org closed.
```

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

Password for level 27 is **upsNCc7vzaRDx6oZC6GiR6ERwe1MowGB**

Bandit Level 27 → Level 28

Level Goal

There is a git repository at ssh://bandit27-git@localhost/home/bandit27-git/repo via the port 2220.

The password for the user bandit27-git is the same as for the user bandit27.

Clone the repository and find the password for the next level.

- First, create a directory to store the cloned repository using the mktemp command with the -d option.
- Then move to that directory
- Then clone the repository.
- Move to the cloned repository; we see a README file.
- Reading the README.md gives us the password.

```
bandit27@bandit:~$ mktemp -d
/tmp/tmp.BWp6TjhPFk
bandit27@bandit:~$ cd /tmp/tmp.BWp6TjhPFk
bandit27@bandit:/tmp/tmp.BWp6TjhPFk$ git clone ssh://bandit27-git@localhost:2220/hom
e/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:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/urerLY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

```
backend: gibson-0
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.BWp6TjhPFk$ ls
repo
bandit27@bandit:/tmp/tmp.BWp6TjhPFk$ cd repo/
bandit27@bandit:/tmp/tmp.BWp6TjhPFk/repo$ ls
README
bandit27@bandit:/tmp/tmp.BWp6TjhPFk/repo$ cat README
The password to the next level is: Yz9IpL0sBcCeuG7m9uQFt8ZNpS4HZRcN
bandit27@bandit:/tmp/tmp.BWp6TjhPFk/repo$
```

Password for level 28 is **Yz9IpL0sBcCeuG7m9uQFt8ZNpS4HZRcN**

Bandit Level 28 → Level 29

Level Goal

There is a git repository at ssh://bandit28-git@localhost/home/bandit28-git/repo via the port 2220.

The password for the user bandit28-git is the same as for the user bandit28.

Clone the repository and find the password for the next level.

- Repeat same step as previous level

```
bandit28@bandit:~$ mktemp -d
/tmp/tmp.fm8KKNN16L
bandit28@bandit:~$ cd /tmp/tmp.fm8KKNN16L
bandit28@bandit:/tmp/tmp.fm8KKNN16L$ git clone ssh://bandit28-git@localhost:2220/hom
e/bandit28-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:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/urerLY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

```
backend: gibson-0
bandit28-git@localhost's password:
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 2), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (2/2), done.
bandit28@bandit:/tmp/tmp.fm8KKNN16L$ ls
repo
bandit28@bandit:/tmp/tmp.fm8KKNN16L$ cd repo/
bandit28@bandit:/tmp/tmp.fm8KKNN16L/repo$ ls
README.md
bandit28@bandit:/tmp/tmp.fm8KKNN16L/repo$ cat README.md
# Bandit Notes
Some notes for level29 of bandit.

## credentials

- username: bandit29
- password: xxxxxxxxxx
```

Password is not here.

- Considering this is a repository, there might be a version where the value is not obfuscated. Let's check the log to view all commits:

```
bandit28@bandit:/tmp/tmp.fm8KKNN16L/repo$ git log
commit 710c14a2e43cfd97041924403e00efb00b3a956e (HEAD -> master, origin/master, orig
in/HEAD)
Author: Morla Porla <morla@overthewire.org>
Date:   Fri Aug 15 13:16:10 2025 +0000

    fix info leak

commit 68314e012fbaa192abfc9b78ac369c82b75fab8f
Author: Morla Porla <morla@overthewire.org>
Date:   Fri Aug 15 13:16:10 2025 +0000

    add missing data

commit a158f9a82c29a16dcea474458a5ccf692a385cd4
Author: Ben Dover <noone@overthewire.org>
Date:   Fri Aug 15 13:16:10 2025 +0000

    initial commit of README.md
```

- The latest commit says "fix info leak." We should check the file's content in the previous commit using the show command.
- The commit hash is 68314e012fbaa192abfc9b78ac369c82b75fab8f, but we can shorten it to 68314e.

```
bandit28@bandit:/tmp/tmp.fm8KKNN16L/repo$ git show 68314e:README.md
# Bandit Notes
Some notes for level29 of bandit.

## credentials

- username: bandit29
- password: 4pT1t5DENaYuqnqvadYs1oE4QLCdjmJ7
```

Password for level 29 is **4pT1t5DENaYuqnqvadYs1oE4QLCdjmJ7**

Bandit Level 29 → Level 30

Level Goal

There is a git repository at ssh://bandit29-git@localhost/home/bandit29-git/repo via the port 2220. The password for the user bandit29-git is the same as for the user bandit29.

Clone the repository and find the password for the next level.

Will do same as previous:

```
bandit29@bandit:~$ mktemp -d
/tmp/tmp.KKheGYVx3F
bandit29@bandit:~$ cd /tmp/tmp.KKheGYVx3F
bandit29@bandit:/tmp/tmp.KKheGYVx3F$ git clone ssh://bandit29-git@localhost:2220/ho
me/bandit29-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:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/urerLY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

```
backend: gibson-0
bandit29-git@localhost's password:
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 16 (delta 2), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (16/16), done.
Resolving deltas: 100% (2/2), done.
bandit29@bandit:/tmp/tmp.KKheGYVx3F$ ls
repo
bandit29@bandit:/tmp/tmp.KKheGYVx3F$ cd repo/
bandit29@bandit:/tmp/tmp.KKheGYVx3F/repo$ ls
README.md
bandit29@bandit:/tmp/tmp.KKheGYVx3F/repo$ cat README.md
# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

- username: bandit30
- password: <no passwords in production!>
```

- While reading the README.md, we found a note “no passwords in production.”
- This suggests that there is another version of this file in a different branch of the repository that contains the password.
- Branch in Git is a feature that allows users to have multiple versions of the same codebase, enabling parallel development and isolating changes. A branch can be merged back into the main branch, usually after reaching a certain milestone.
- For example, in a project, there might be branches for production, development, features, bug fixes, etc.
- We can view all branches of this repository using the git branch command with the -a option:

```
bandit29@bandit:/tmp/tmp.KKheGYVx3F/repo$ git branch -a
* master
  remotes/origin/HEAD → origin/master
  remotes/origin/dev
  remotes/origin/master
  remotes/origin/sploits-dev
bandit29@bandit:/tmp/tmp.KKheGYVx3F/repo$
```

- We found four results: the master branch (which we are currently using, indicated by an asterisk) and three remote branches: dev, master, and sploits-dev.
- A remote branch is one that exists elsewhere (like GitHub, GitLab, or a coworker's PC).
- "origin" is the nickname given to this remote repository.
- The line remotes/origin/HEAD -> origin/master is not a branch but merely a pointer indicating that "master" is the main branch in this remote repository.
- Since the notes mention "no passwords in production," we might be able to find the password in the dev (development) branch.
- We can switch branches using the checkout command:

```
bandit29@bandit:/tmp/tmp.KKheGYVx3F/repo$ git checkout -b dev  origin/dev
branch 'dev' set up to track 'origin/dev'.
Switched to a new branch 'dev'
```

- Here, -b refers to the branch, "dev" is the nickname we give it, and "origin/dev" is the source.
- Now, we will read the log using the --oneline option to view only the summary.
- The latest commit has message "add data." Let's read the README.md at this commit:

```
bandit29@bandit:/tmp/tmp.KKheGYVx3F/repo$ git log --oneline
569126a (HEAD → dev, origin/dev) add data needed for development
76f80f8 add gif2ascii
ccacd76 (origin/master, origin/HEAD, master) fix username
e915edf initial commit of README.md
bandit29@bandit:/tmp/tmp.KKheGYVx3F/repo$ git show 569126a:README.md
# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

- username: bandit30
- password: qp30ex3VLz5MDG1n91YowTv4Q8l7CDZL
```

Password for level 30 is **qp30ex3VLz5MDG1n91YowTv4Q8l7CDZL**

Bandit Level 30 → Level 31

Level Goal

There is a git repository at ssh://bandit30-git@localhost/home/bandit30-git/repo via the port 2220. The password for the user bandit30-git is the same as for the user bandit30.

Clone the repository and find the password for the next level.

```
bandit30@bandit:~$ mktemp -d
/tmp/tmp.oKD2BBLdPY
bandit30@bandit:~$ cd /tmp/tmp.oKD2BBLdPY
bandit30@bandit:/tmp/tmp.oKD2BBLdPY$ git clone ssh://bandit30-git@localhost:2220/ho
me/bandit30-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:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/urerLY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

```
backend: gibson-0
bandit30-git@localhost's password:
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
bandit30@bandit:/tmp/tmp.oKD2BBLdPY$ ls
repo
bandit30@bandit:/tmp/tmp.oKD2BBLdPY$ cd repo/
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$ ls
README.md
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$ cat README.md
just an epmty file... muahaha
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$ git log --oneline
de654f2 (HEAD → master, origin/master, origin/HEAD) initial commit of README.md
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$ git branch -a
* master
  remotes/origin/HEAD → origin/master
  remotes/origin/master
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$ git stash list
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$ git tag
secret
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$ git show secret
fb5S2xb7bRyFmAvQYQGEqsbhVyJqhnDy
bandit30@bandit:/tmp/tmp.oKD2BBLdPY/repo$
```

Since there is nothing in README.md file, we have to do everything:

- **1st Attempt**(Fail): Commit History
  - Using git log, we searched for all commits that have been made, and we used the --oneline option to see only the summary
  - There is only one commit, which means no other version of README.md is stored.
- **2nd Attempt**(Fail): Search Branches
  - Using git branch -a will show all branches in this repository
  - We only have one local branch (the one we are currently on) and its remote version: master.
  - The line remotes/origin/HEAD -> origin/master is not a branch but just a pointer showing that "master" is the main branch in this remote repository.
- **3rd Attempt**(Fail): Find Uncommitted Changes
  - We can use git status to find uncommitted changes
- **4th Attempt**(Fail): Find Stash

- Stash is a feature that saves your temporary changes that you don't want to commit yet, which is useful when you want to change branches.
- We can use the git stash list command to view all stashes

- **5th Attempt**(Success): Find Tags
  - Tags are references to certain commits, often used to mark versions (e.g., v1.0, v1.1).
  - We can use git tag to show all tags
  - We found a tag called "secret." We can read it using:git show secret

We found the password for the next level.
Password for level 31 is **fb5S2xb7bRyFmAvQYQGEqsbhVyJqhnDy**

Bandit Level 31 → Level 32

Level Goal

There is a git repository at ssh://bandit31-git@localhost/home/bandit31-git/repo via the port 2220. The password for the user bandit31-git is the same as for the user bandit31.

Clone the repository and find the password for the next level.

```
bandit31@bandit:~$ mktemp -d
/tmp/tmp.8RE00Qw557
bandit31@bandit:~$ cd /tmp/tmp.8RE00Qw557
bandit31@bandit:/tmp/tmp.8RE00Qw557$ git clone ssh://bandit31-git@localhost:2220/ho
me/bandit31-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:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/urerLY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

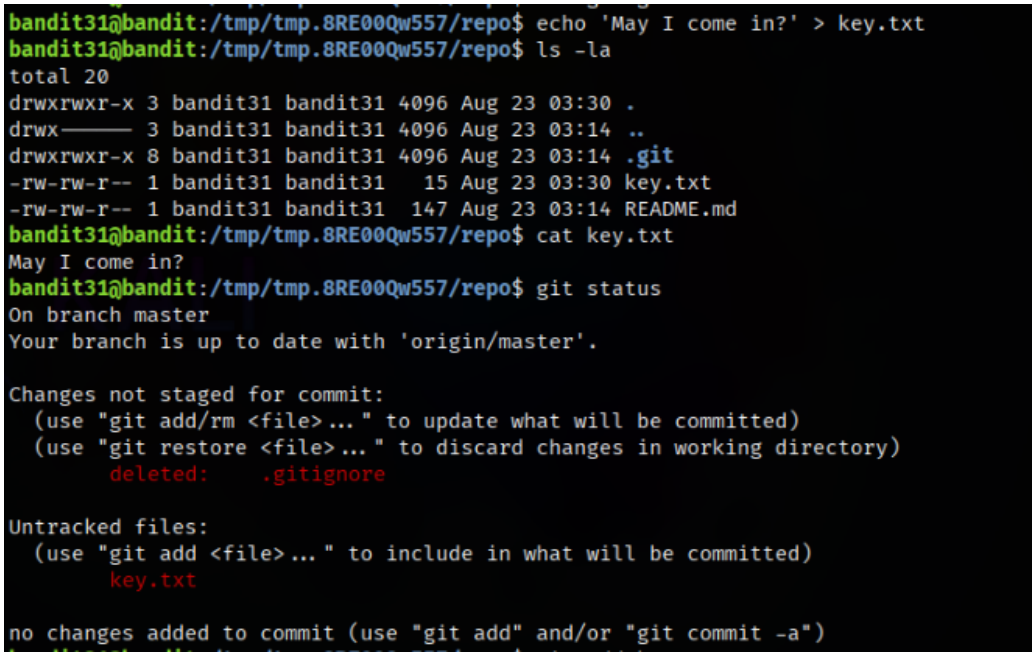
```
backend: gibson-0
bandit31-git@localhost's password:
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
bandit31@bandit:/tmp/tmp.8RE00Qw557$ cd repo/
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ ls
README.md
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ cat  README.md
This time your task is to push a file to the remote repository.

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

- The README.md contains instructions to create a file and push it to the remote repository.
- This is what we're going to do: create key.txt, add it to Git, create a commit, and push it to the remote repository.
- First, we confirm that the remote repository with a branch called master has been added

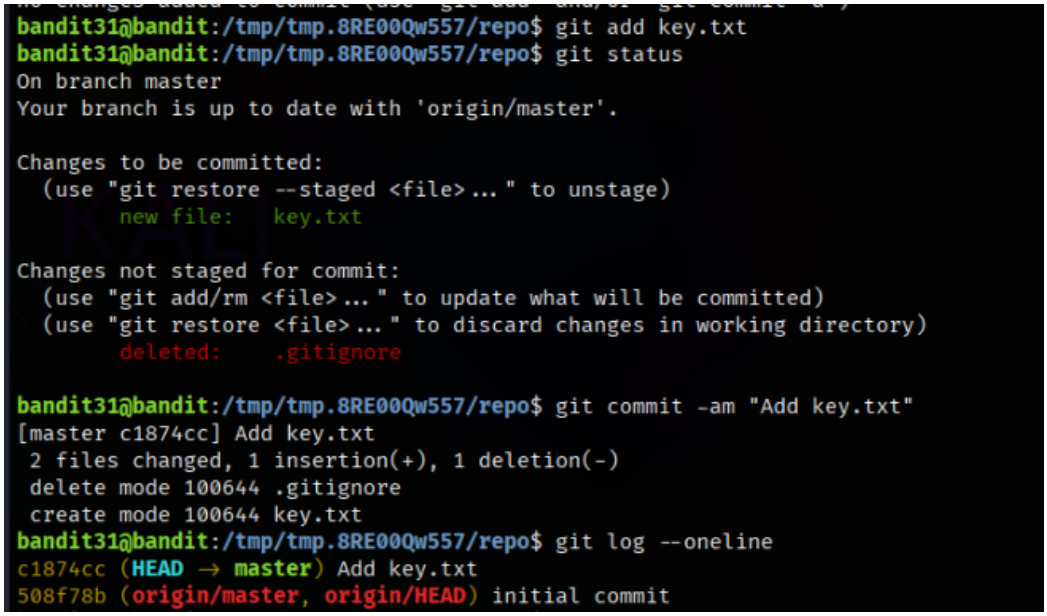
```
bandit31@bandit:~$ cd /tmp/tmp.8RE00Qw557/repo
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ cat .gitignore
*.txt
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ rm .gitignore
```

- Yes, it has been added.
- Then we check the content of .gitignore
- .gitignore file contains of files or directories that won't be tracked by Git.
- These files include log files, temporary files, files with sensitive credentials, dependencies, etc.
- It shows that it will ignore all .txt files, which could be a problem because we're going to create key.txt.
- So, we need to remove it.

All done, so let's make a key.txt file:



- key.txt has been created, and the content has been entered correctly. We also no longer see the .gitignore file that we just deleted.
- Now, we use git status: As expected, it found two changes: the new file key.txt and the deleted file .gitignore.



- We need to tell Git to track this file using git add
- Using git status again, the status of key.txt has changed from untracked to ready for commit:

Let's commit it:

```
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ git add key.txt
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   key.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    .gitignore

bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ git commit -am "Add key.txt"
[master c1874cc] Add key.txt
 2 files changed, 1 insertion(+), 1 deletion(-)
 delete mode 100644 .gitignore
 create mode 100644 key.txt
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ git log --oneline
c1874cc (HEAD -> master) Add key.txt
508f78b (origin/master, origin/HEAD) initial commit
```

- let's check our new commit using git log with the --oneline option to only show the summary.
- We can see our local branch is one commit ahead of the remote one.
- Let's push this new commit to the remote repository.

```
bandit31@bandit:/tmp/tmp.8RE00Qw557/repo$ git push origin master
The authenticity of host '[localhost]:2220 ([127.0.0.1]:2220)' can't be established
.
ED25519 key fingerprint is SHA256:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/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/bandit31/.ssh' (Permission denied).
Failed to add the host to the list of known hosts (/home/bandit31/.ssh/known_hosts)
.
backend: gibson-0
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), 289 bytes | 289.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote: ### Attempting to validate files... ####
remote:
remote: .oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.
remote:
remote: Well done! Here is the password for the next level:
remote: 3O9RfhqyAlVBEZpVb6LYStshZoqoSx5K
remote:
remote: .oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.
remote:
To ssh://localhost:2220/home/bandit31-git/repo
 ! [remote rejected] master -> master (pre-receive hook declined)
error: failed to push some refs to 'ssh://localhost:2220/home/bandit31-git/repo'
```

Password for level 32 is **3O9RfhqyAlVBEZpVb6LYStshZoqoSx5K**

Bandit Level 32 → Level 33

Level Goal

After all this git stuff, it's time for another escape. Good luck!

```
WELCOME TO THE UPPERCASE SHELL
>>
>> ls -la
sh: 1: LS: Permission denied
>> ls ls -la
sh: 1: LS: Permission denied
```

- The response mentions sh, which means it is using the sh shell, another shell similar to bash and zsh.
- This indicates that the uppercase shell acts as a middleman that processes our input and then feeds it to the sh shell: input → uppercase_shell → sh.
- we need to identify what this uppercase shell does and how to exploit it.
- in responce: first -la is not mentioned in second other ls -la is not mentioned.
- LS (in uppercase) is not a valid command, but instead of getting a "command not found" error, we receive a "permission denied" message.
- Among the three words, only one is mentioned, which means it only accepts one input.

Trying Environment Variables

```
>> $HOME
sh: 1: /home/bandit32: Permission denied
>> $USER
sh: 1: bandit32: Permission denied
>> $SHELL
WELCOME TO THE UPPERCASE SHELL
>> $0
$ whoami
bandit33
$ pwd
/home/bandit32
$ ls -la
total 36
drwxr-xr-x   2 root     root      4096 Aug 15 13:16 .
drwxr-xr-x 150 root     root      4096 Aug 15 13:18 ..
-rw-r--r--   1 root     root       220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root      3851 Aug 15 13:09 .bashrc
-rw-r--r--   1 root     root       807 Mar 31  2024 .profile
-rwsr-x---   1 bandit33 bandit32 15140 Aug 15 13:16 uppershell
$ /bin/bash
bandit33@bandit:~$ pwd
/home/bandit32
bandit33@bandit:~$ cat /etc/bandit_pass/bandit33
tQdtbs5D5i2vJwkO8mEyYEyTL8izoeJ0
bandit33@bandit:~$
```

Here's what is happening:

- We typed $HOME. Instead of printing the string /home/bandit32, the program tried to execute it (like running sh -c "/home/bandit32"). But /home/bandit32 is a directory, not an executable file → the kernel/sh refuses to run it → Permission denied.

- We typed $USER. The program again tried to execute the string bandit32 as a command (sh -c "bandit32"). There is no executable named bandit32 in PATH, or the system refused execution → Permission denied again.
- Key point: the shell is not simply echoing variables — it is treating variable values as commands to execute.
- $SHELL points to a path that is an executable (likely the uppercase shell itself). When the program attempts to execute it, it runs another instance of the uppercase shell (hence we see the welcome message again).
- $0 returns the process name/path of the running program (the binary we logged into). Because $0 pointed to the actual uppershell binary file, the uppercase-shell executed that file — and that file is setuid to bandit33 (we'll confirm below).
- When that setuid binary runs, the child process (the new shell we can use) runs with effective UID = bandit33. We then ran whoami and it returned bandit33 — we became bandit33.
- ls -la shows uppershell is present and owned by bandit33:bandit32 with permissions -rwsr-x---.
    - The s in the owner execute bit (rws) means setuid → running this program makes the process run with the owner's privileges (owner = bandit33).
    - Group bandit32 has execute permission (x), so the bandit32 user is allowed to run it.
    - Others have no permissions.
- We launched /bin/bash from that intermediate shell and got a full interactive bash running as bandit33 (confirmed by the prompt and whoami earlier).
- Now we can read /etc/bandit_pass/bandit33 and obtain the next-level password.

Password for level 33 is **tQdtbs5D5i2vJwkO8mEyYEyTL8izoeJ0**

In summary:

- The login shell for bandit32 is not a normal shell — it converts input to UPPERCASE and blocks normal commands, but it reads environment variables.
- When an environment variable's value points to an executable, the uppercase-shell executes that executable instead of just printing the value (this is the key quirk).
- $0 (the shell's program name) contains the path to the running shell program. Reading/using $0 causes the uppercase-shell to invoke the underlying program (which creates an interactive /bin/sh for us).
- The uppercase-shell binary is setuid to bandit33 (owner is bandit33), so when the binary runs and spawns a child shell, that child runs with bandit33 privileges.
- Once we have a normal shell as bandit33, we can read /etc/bandit_pass/bandit33.