BANDIT WALKTHROUGH

OverTheWire Bandit - CTF

Brief Description

In this document I will show the solutions for each level of the Linux shell CTF made by OverTheWire.

Alberto Gómez Ferrández
albertogomezferrandez@gmail.com

Connection to the machine

To access any level of the CTF, we will need to make an SSH connection to a machine located at bandit.labs.overthewire.com. The SSH service is served at port 2220.

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

LEVEL 0

Level 0 is as simple as expected. When we log-in via SSH and access the home directory, we will find a readme file with the password inside.

```
bandit0@bandit:~$ ls
readme
bandit0@bandit:~$ cat readme
boJ9jbbUNNfktd78OOpsqOltutMc3MY1
bandit0@bandit:~$
```

LEVEL 1

In level 1, we find a file called "-". We can't use "cat -", so we'll have to specify its path:

```
bandit1@bandit:~$ ls
-
bandit1@bandit:~$ cat ./-
CV1DtqXWVFXTvM2F0k09SHz0YwRINYA9
bandit1@bandit:~$
```

LEVEL 2

In level 2 there's a file with blank spaces in its name, which obstructs the use of cat command. If we typed "cat spaces in this filename", it would search four files. We need to use the "\" character to escape the blank spaces:

```
bandit2@bandit:~$ ls
spaces in this filename
bandit2@bandit:~$ cat spaces\ in\ this\ filename
UmHadQclWmgdLOKQ3YNgjWxGoRMb5luK
bandit2@bandit:~$
```

LEVEL 3

In level 3 we find a hidden file. In order to find it, we need to use the *-a* option of *ls* command, which shows hidden files. Then we just have to read it with *cat*.

```
bandit3@bandit:~$ ls
inhere
bandit3@bandit:~$ ls inhere/
bandit3@bandit:~$ ls -la inhere/
total 12
drwxr-xr-x 2 root    root    4096 May  7  2020 .
drwxr-xr-x 3 root    root    4096 May  7  2020 ..
-rw-r----- 1 bandit4 bandit3   33 May  7  2020 .hidden
bandit3@bandit:~$ cat inhere/.hidden
pIwrPrtPN36QITSp3EQaw936yaFoFgAB
bandit3@bandit:~$
```

LEVEL 4

In level 4 we are told:

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

We can use the command *file* to check the content type of the files in the directory with:

```
bandit4@bandit:~$ ls
inhere
bandit4@bandit:~$ ls inhere/
-file00  -file01  -file02  -file03  -file04  -file05  -file06  -file07  -file08  -file09
bandit4@bandit:~$ file inhere/-file*
inhere/-file00: data
inhere/-file01: data
inhere/-file02: data
inhere/-file03: data
inhere/-file04: data
inhere/-file05: data
inhere/-file06: data
inhere/-file07: ASCII text
inhere/-file08: data
inhere/-file09: data
bandit4@bandit:~$ cat inhere/-file07
koReBOKuIDDepwhWk7jZC0RTdopnAYKh
bandit4@bandit:~$
```

LEVEL 5

In level 5 we are told:

"*The password for the next level is stored in a file somewhere under the inhere directory and has all of the following properties: human-readable, 1033 bytes in size, not executable*."

We need to use the *find* command with following options:

- ".": to search on the actual directory (*inhere*).
- -type f: to tell the command to only search files.
- -size 1033c: to only search 1033 bytes long files with".
- ! -executable: to not include executables on the search.

```
bandit5@bandit:~$ find . -type f -size 1033c ! -executable
./inhere/maybehere07/.file2
bandit5@bandit:~$ cat inhere/maybehere07/.file2
DXjZPULLxYr17uwoI01bNLQbtFemEgo7
```

LEVEL 6

In level 6 we are told:

"*The password for the next level is stored somewhere on the server and has all of the following properties: owned by user bandit7, owned by group bandit6 33 bytes in size*."

We need to use the *find* command with following options:

- "/": to search on the root directory.
- -type f: to tell the command to only search files.
- -size 33c: to only search 1033 bytes long files with".
- -user bandit7: to search files owned by user bandit7.
- -group bandit6: to search files owned by group bandit6.
- 2>/dev/null: only to avoid error printing on screen.

```
bandit6@bandit:~$ find / -type f -size 33c -user bandit7 -group bandit6 2>/dev/null
/var/lib/dpkg/info/bandit7.password
bandit6@bandit:~$ cat /var/lib/dpkg/info/bandit7.password
HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs
bandit6@bandit:~$
```

LEVEL 7

In level 7 we are told:

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

That means the password is on the same line as the password, so we can use grep to extract the line in which "*millionth*" is located:

```
bandit7@bandit:~$ ls
data.txt
bandit7@bandit:~$ cat data.txt | grep 'millionth'
millionth       cvX2JJa4CFALtqS87jk27qwqGhBM9plV
bandit7@bandit:~$
```

LEVEL 8

In level 8 we are told:

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

Uniq filters adjacent matching lines from INPUT; -u option prints only the unique lines. We use *sort* command so that matching lines are adjacent to each other.

In this one we need to combine the use of the *sort* and *uniq* commands.

```
bandit8@bandit:~$ ls
data.txt
bandit8@bandit:~$ sort data.txt | uniq -u
UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhR
bandit8@bandit:~$
```

LEVEL 9

In level 9 we are told:

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

Strings is a useful command to search string on binary files. We can use it and filter its output with grep, so only lines with several "=" characters are shown.

```
bandit9@bandit:~$ ls
data.txt
bandit9@bandit:~$ strings data.txt | grep '=='
========== the*2i"4
========== password
Z)========== is
&========== truKLdjsbJ5g7yyJ2X2R0o3a5HQJFuLk
bandit9@bandit:~$
```

LEVEL 10

In level 10 we are told:

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

We can easily encode or decode base64 data with *base64* command.

```
bandit10@bandit:~$ ls
data.txt
bandit10@bandit:~$ base64 -d data.txt
The password is IFukwKGsFW8MOq3IRFqrxE1hxTNEbUPR
bandit10@bandit:~$
```

LEVEL 11

In level 11 we are told:

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

To achieve this, we can use the *tr* command, which allow us to translate characters from a string.

```
bandit11@bandit:~$ ls
data.txt
bandit11@bandit:~$ cat data.txt | tr '[a-z]' '[n-za-m]' | tr '[A-Z]' '[N-ZA-M]'
The password is 5Te8Y4drgCRfCx8ugdwuEX8KFC6k2EUu
bandit11@bandit:~$
```

LEVEL 12

We are told data.txt is a hexdump of a compressed file, so we can use *xxd -r* to reverse the hexdump. With *file* we check the type of file it is. We see it is a gzip file, so we change the filename and give it a .gz extension. Then, we must unzip it and check the file type again. We see it is a bzip2 file.

```
bandit12@bandit:/tmp/bandit12$ xxd -r data.txt > data
bandit12@bandit:/tmp/bandit12$ file data
data: gzip compressed data, was "data2.bin", last modified: Thu May  7 18:14:30 2020, max compression
, from Unix
bandit12@bandit:/tmp/bandit12$ mv data data.gz
bandit12@bandit:/tmp/bandit12$ gzip -d data.gz
bandit12@bandit:/tmp/bandit12$ ls
data  data.txt
bandit12@bandit:/tmp/bandit12$ file data
data: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/bandit12$
```

Again, we change the extension to match the file type and then decompress it. The result file is a tar archive.

```
bandit12@bandit:/tmp/bandit12$ mv data data.bz2
bandit12@bandit:/tmp/bandit12$ bzip2 -d data.bz2
bandit12@bandit:/tmp/bandit12$ ls
data  data.txt
bandit12@bandit:/tmp/bandit12$ file data
data: gzip compressed data, was "data4.bin", last modified: Thu May  7 18:14:30 2020, max compression
, from Unix
bandit12@bandit:/tmp/bandit12$ mv data data.gz
bandit12@bandit:/tmp/bandit12$ gzip -d data.gz
bandit12@bandit:/tmp/bandit12$ file data
data: POSIX tar archive (GNU)
bandit12@bandit:/tmp/bandit12$
```

We change the file extension to match *tar* type and extract the files inside. We find the data5.bin file, which is a tar achive again. We repeat the the process until we get a data8.bin file, which is a gzip file again.

```
bandit12@bandit:/tmp/bandit12$ mv data data.tar
bandit12@bandit:/tmp/bandit12$ tar -xvf data.tar
data5.bin
bandit12@bandit:/tmp/bandit12$ ls
data5.bin  data.tar  data.txt
bandit12@bandit:/tmp/bandit12$ file data5.bin
data5.bin: POSIX tar archive (GNU)
bandit12@bandit:/tmp/bandit12$ tar -xvf data5.bin
data6.bin
bandit12@bandit:/tmp/bandit12$ tar -xvf data6.bin
data8.bin
bandit12@bandit:/tmp/bandit12$ tar -xvf data8.bin
bandit12@bandit:/tmp/bandit12$ ls
data5.bin  data6.bin  data8.bin  data.tar  data.txt
bandit12@bandit:/tmp/bandit12$ file data8.bin
data8.bin: gzip compressed data, was "data9.bin", last modified: Thu May  7 18:14:30 2020, max compre
ssion, from Unix
```

We decompress it and find a *data8* file with the password inside.

```
bandit12@bandit:/tmp/bandit12$ mv data8.bin data8.gz
bandit12@bandit:/tmp/bandit12$ gzip -d data8.gz
bandit12@bandit:/tmp/bandit12$ ls
data5.bin  data6.bin  data8  data.tar  data.txt
bandit12@bandit:/tmp/bandit12$ cat data8
The password is 8ZjyCRiBWFYkneahHwxCv3wb2a1ORpYL
bandit12@bandit:/tmp/bandit12$
```

LEVEL 13

In level 13 we don't have to look for a password. We are given an SSH key to authenticate via public key to the SSH server. We must copy the content of the key file to a local file and use it to log in with the user "bandit14" with the "*-i*" option.

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

LEVEL 14

In level 14 we are told:

"*The password for the next level can be retrieved by submitting the password of the current level to port 30000 on localhost.*"

To achieve this we can establish a simple connection with telnet protocol. Then, we must send the password for level 14 through the connection. This password can be found in the folder /etc/bandit_pass/bandit14:

```
bandit14@bandit:~$ telnet localhost 30000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
4wcYUJFw0k0XLShlDzztnTBHiqxU3b3e
Correct!
BfMYroe26WYalil77FoDi9qh59eK5xNr

Connection closed by foreign host.
bandit14@bandit:~$
```

LEVEL 15

In level 15 we are told:

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

To achive this, we can use the "openssl s_client" commands, which are used to open SSL connections to the given host:port.

```
bandit15@bandit:~$ openssl s_client -connect localhost:30001
CONNECTED(00000003)
depth=0 CN = localhost
verify error:num=18:self signed certificate
verify return:1
depth=0 CN = localhost
verify return:1
```

Then we introduce the current password:

```
---
BfMYroe26WYalil77FoDi9qh59eK5xNr
Correct!
cluFn7wTiGryunymYOu4RcffSxQluehd

closed
bandit15@bandit:~$
```

LEVEL 16

Using Nmap we can see which ports are open:

```
bandit16@bandit:~$ nmap localhost -p31000-32000

Starting Nmap 7.40 ( https://nmap.org ) at 2021-09-10 12:03 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00050s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
31046/tcp open  unknown
31518/tcp open  unknown
31691/tcp open  unknown
31790/tcp open  unknown
31960/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds
bandit16@bandit:~$
```

Using nmap with -A option will give us information about the services running on the ports, so we can see which of them use SSL encryption. We discover that ports 31518 and 31790 offer SSL encryption.

Port number 31790 specific has the string “Please enter the correct current password”, so that must the the good one.

```
31790/tcp open  ssl/unknown
| fingerprint-strings:
|   FourOhFourRequest, GenericLines, GetRequest, HTTPOptions, Help, Kerberos, LDAPSearchReq, LPDString,
RTSPRequest, SIPOptions, SSLSessionReq, TLSSessionReq:
|_    Wrong! Please enter the correct current password
| ssl-cert: Subject: commonName=localhost
| Subject Alternative Name: DNS:localhost
| Not valid before: 2021-08-05T21:23:01
|_Not valid after:  2022-08-05T21:23:01
|_ssl-date: TLS randomness does not represent time
```

We type “openssl s_client -connect localhost:31790” and enter the password:

```
---
cluFn7wTiGryunymYOu4RcffSxQluehd
Correct!
-----BEGIN RSA PRIVATE KEY-----
```

As a response, we are given a key to log in on the next level.

LEVEL 17

In level 17 we are told:

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

We can use diff command to compare both files. The output will be the lines that differ from each other.

```
bandit17@bandit:~$ diff passwords.old passwords.new
42c42
< w0Yfolrc5bwjS4qw5mq1nnQi6mF03bii
---
> kfBf3eYk5BPBRzwjqutbbfE887SVc5Yd
bandit17@bandit:~$
```

Being the second one the correct password.

LEVEL 18

In level 18 we are logged out of the server when we log in using SSH connection. We are told that the password is stored in a *readme* file in the home directory, so we have to learn how to read it sending commands through SSH.

```
C:\Users\alber>ssh bandit18@bandit.labs.overthewire.org -p 2220 "cat ./readme"
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames

bandit18@bandit.labs.overthewire.org's password:
IueksS7Ubh8G3DCwVzrTd8rAVOwq3M5x

C:\Users\alber>
```

LEVEL 19

In level 19 we are told:

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

If we check the permissions, we find out that we can execute the script (with the group permissions).

```
bandit19@bandit:~$ ./bandit20-do
Run a command as another user.
  Example: ./bandit20-do id
bandit19@bandit:~$ ls -l
total 8
-rwsr-x--- 1 bandit20 bandit19 7296 May  7  2020 bandit20-do
bandit19@bandit:~$
```

Thanks to the setuid, when we execute the script, we do it with user owner permissions, so we can read the bandit20's password file as if we were bandit20.

```
bandit19@bandit:~$ ./bandit20-do cat /etc/bandit_pass/bandit20
GbKksEFF4yrVs6il55v6gwY5aVje5f0j
bandit19@bandit:~$
```

LEVEL 20

In level 20, we can initiate a Netcat listener at any port (8000, for example) and make it send the value of bandit20's password to anyone that connects to it. and send it to the background with "&" so we can continue.

```
bandit20@bandit:~$ nc -l -p 8000 < /etc/bandit_pass/bandit20 &
[1] 26400
bandit20@bandit:~$
```

Then, we use the given "./suconnect" script to connect to port 8000, returning the next level's password.

```
bandit20@bandit:~$ ./suconnect 8000
Read: GbKksEFF4yrVs6il55v6gwY5aVje5f0j
Password matches, sending next password
bandit20@bandit:~$ gE269g2h3mw3pwgrj0Ha9Uoqen1c9DGr
```

LEVEL 21

In level 21, a program is running automatically at regular intervals from *cron*. We can look at /et/cron.d/ for the configuration and find several files:

```
bandit21@bandit:~$ ls /etc/cron.d/
cronjob_bandit15_root  cronjob_bandit22  cronjob_bandit24
cronjob_bandit17_root  cronjob_bandit23  cronjob_bandit25_root
```

As we are looking for bandit22's password, we should check which tasks are being executed under the *cronjob_bandit22* file:

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

We see it is copying the content of the password file to some file at */tmp* folder. We can find the password inside:

```
bandit21@bandit:~$ cat /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
Yk7owGAcWjwMVRwrTesJEwB7WVOiILLI
bandit21@bandit:~$
```

LEVEL 22

In level 22 we are told the same thing. Let's check what script is being executed under *bandit23*:

```
bandit22@bandit:~$ cat /etc/cron.d/cronjob_bandit23
@reboot bandit23 /usr/bin/cronjob_bandit23.sh  &> /dev/null
* * * * * bandit23 /usr/bin/cronjob_bandit23.sh  &> /dev/null
bandit22@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
bandit22@bandit:~$
```

The script stores user's password on a file with its filename being the result of several functions. It calculates the md5 hash of the string "I am user bandit23" and then uses *cut* to clear the program output.

```
bandit22@bandit:~$ echo I am user bandit23 | md5sum
8ca319486bfbbc3663ea0fbe81326349  -
bandit22@bandit:~$ echo I am user bandit23 | md5sum | cut -d ' ' -f 1
8ca319486bfbbc3663ea0fbe81326349
bandit22@bandit:~$
```

The hash function must be called that way and not "md5sum 'I am user bandit23'" as it is meant to calculate file hashes. That's why we use the pipe.

We'll find the password inside */tmp/$filename*:

```
bandit22@bandit:~$ cat /tmp/8ca319486bfbbc3663ea0fbe81326349
jc1udXuA1tiHqjIsL8yaapX5XIAI6i0n
bandit22@bandit:~$
```

LEVEL 23

In level 23 we are told the same thing. Let's check what script is being executed under *bandit24*:

```
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/$myname
echo "Executing and deleting all scripts in /var/spool/$myname:"
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:~$
```

We can see how scripts at /var/spool/bandit24 are being executed, so we have to write a script on that folder (in which we have writing permission).

The script will copy the content of */etc/bandit_pass/bandit24* on a file we have access to. We can create that file at a folder inside */tmp*. After the 1-minute timeout, we get the password in our file.

```
bandit23@bandit:~$ mkdir /tmp/bandit23
bandit23@bandit:~$ cd /tmp/bandit23
bandit23@bandit:/tmp/bandit23$ touch pass
bandit23@bandit:/tmp/bandit23$ chmod 777 pass
bandit23@bandit:/tmp/bandit23$ echo "cat /etc/bandit_pass/bandit24 > /tmp/bandit23/pass" > getpass.sh
bandit23@bandit:/tmp/bandit23$ chmod 777 getpass.sh
bandit23@bandit:/tmp/bandit23$ cp getpass.sh /var/spool/bandit24/getpass.sh
bandit23@bandit:/tmp/bandit23$ cat pass
UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ
bandit23@bandit:/tmp/bandit23$
```

LEVEL 24

In level 24 we are told:

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

We create a script in a folder in which we can write, like */tmp/bandit24*:

```
for i in {0000..9999}; do
        echo UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ $i
done | nc localhost 30002 | grep -v Wrong
```

The script sends all the combinations of the actual password plus the pin code to the *Netcat* input, and uses *grep* to filter the output, showing only the correct solution:

```
bandit24@bandit:/tmp/bandit24$ ./pass.sh
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.
Correct!
The password of user bandit25 is uNG9O58gUE7snukf3bvZ0rxhtnjzSGzG

Exiting.
bandit24@bandit:/tmp/bandit24$
```

LEVEL 25

In level 25 we are given a SSH key to directly access bandit26 shell.

```
bandit25@bandit:~$ ls
bandit26.sshkey
```

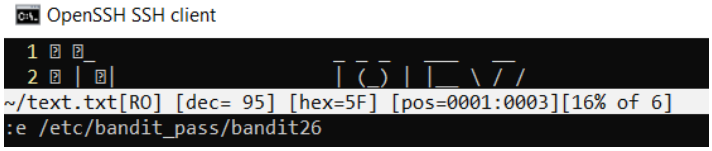
LEVEL 26

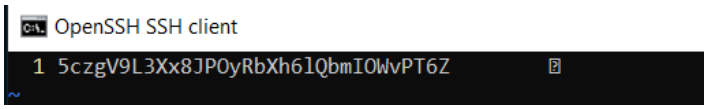
When accessing Level 26 we are immediately logged out from the server. Let's check from level 25 what shell it is using:

```
bandit25@bandit:~$ cat /etc/passwd | grep bandit26
bandit26:x:11026:11026:bandit level 26:/home/bandit26:/usr/bin/showtext
bandit25@bandit:~$ cat /usr/bin/showtext
#!/bin/sh

export TERM=linux

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

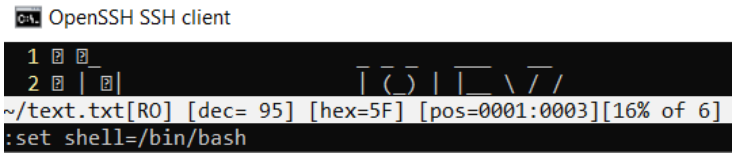
This one is tricky. We have to make the console window small enough so *more* cannot show the entire *text.txt* file:

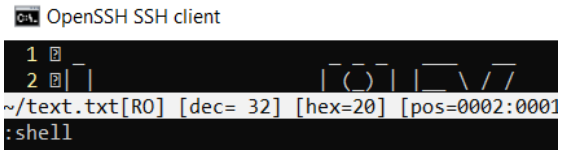
From there, we can type “v” to execute *vim* inside the execution of *more*. From vim, we can read a file with the “:e” syntax. We use it to read this level’s password.



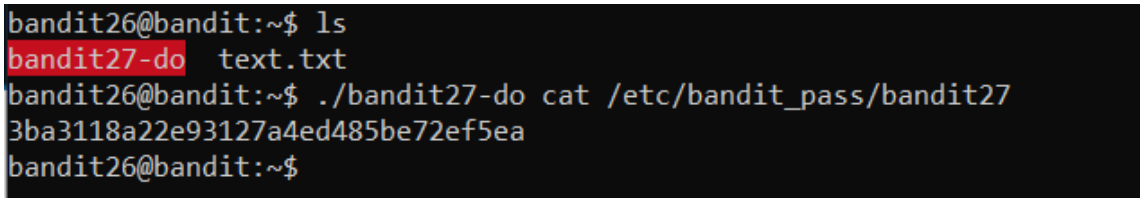


Then, we can change the user shell with “:set” and change to that shell with :shell, which loads the shell of the user.





This way, we get to execute a *bash* shell. We find the *bandit27-do* script, which executes tasks as user *bandit27*. Using it we can read *bandit27*’s password file.



LEVEL 27

In level 27 we are introduced to a git repository we can access. Let's create a folder in which we can clone the repository.

```
bandit27@bandit:~$ mkdir /tmp/bandit27
bandit27@bandit:~$ cd /tmp/bandit27
bandit27@bandit:/tmp/bandit27$ git clone ssh://bandit27-git@localhost/home/bandit27-git/repo
Cloning into 'repo'...
```

We can find the password inside the README file of the repository.

```
bandit27@bandit:/tmp/bandit27$ cd repo/
bandit27@bandit:/tmp/bandit27/repo$ ls
README
bandit27@bandit:/tmp/bandit27/repo$ cat README
The password to the next level is: 0ef186ac70e04ea33b4c1853d2526fa2
bandit27@bandit:/tmp/bandit27/repo$
```

LEVEL 28

In level 28, we create again a working folder and clone the repository. We see on the *git log* that there's a commit fixing some information leak, let's see the code of that commit.

```
bandit28@bandit:/tmp/bandit28$ cd repo/
bandit28@bandit:/tmp/bandit28/repo$ git log
commit edd935d60906b33f0619605abd1689808ccdd5ee
Author: Morla Porla <morla@overthewire.org>
Date:   Thu May 7 20:14:49 2020 +0200

    fix info leak

commit c086d11a00c0648d095d04c089786efef5e01264
Author: Morla Porla <morla@overthewire.org>
Date:   Thu May 7 20:14:49 2020 +0200

    add missing data

commit de2ebe2d5fd1598cd547f4d56247e053be3fdc38
Author: Ben Dover <noone@overthewire.org>
Date:   Thu May 7 20:14:49 2020 +0200

    initial commit of README.md
bandit28@bandit:/tmp/bandit28/repo$ git revert edd935d60906b33f0619605abd1689808ccdd5ee
```

We can find the password on the README file of that commit.

```
bandit28@bandit:/tmp/bandit28/repo$ cat README.md
# Bandit Notes
Some notes for level29 of bandit.

## credentials

- username: bandit29
- password: bbc96594b4e001778eee9975372716b2
```

LEVEL 29

In level 29, we create again a working folder and clone the repository. README file says “no password in production”, so we can figure that the password will be on another branch, like a development branch.

```
bandit29@bandit:/tmp/bandit29/repo$ cat README.md
# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

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

bandit29@bandit:/tmp/bandit29/repo$
```

We find the branch *dev* in the remote repository, so we access it and find the password in the README file:

```
bandit29@bandit:/tmp/bandit29/repo$ git branch -r
  origin/HEAD -> origin/master
  origin/dev
  origin/master
  origin/sploits-dev
bandit29@bandit:/tmp/bandit29/repo$ git checkout dev
Branch dev set up to track remote branch dev from origin.
Switched to a new branch 'dev'
bandit29@bandit:/tmp/bandit29/repo$ cat README.md
# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

- username: bandit30
- password: 5b90576bedb2cc04c86a9e924ce42faf

bandit29@bandit:/tmp/bandit29/repo$
```

LEVEL 30

In level 30, we create again a working folder and clone the repository. We see that there's a tag called *secret*, so we try to access that version of the code:

```
bandit30@bandit:/tmp/bandit30/repo$ git tag
secret
bandit30@bandit:/tmp/bandit30/repo$ git checkout secret
fatal: reference is not a tree: secret
bandit30@bandit:/tmp/bandit30/repo$
```

As the tag doesn't represent a tree, we can use *git show* to see what it really is. The docs for this command say that it shows one or more objects (blobs, trees, tags and commits).

```
bandit30@bandit:/tmp/bandit30/repo$ git show secret
47e603bb428404d265f59c42920d81e5
bandit30@bandit:/tmp/bandit30/repo$
```

LEVEL 31

After cloning the repository, we see that README file tell us what we have to do:

```
bandit31@bandit:/tmp/bandit31/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

bandit31@bandit:/tmp/bandit31/repo$
```

We create the file and add it to the repository, but when we check the git status, we see that the file is not being considered:

```
bandit31@bandit:/tmp/bandit31/repo$ echo "May I come in?" > key.txt
bandit31@bandit:/tmp/bandit31/repo$ git add .
bandit31@bandit:/tmp/bandit31/repo$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
bandit31@bandit:/tmp/bandit31/repo$
```

We look for hidden files and find a *gitignore* file telling git to ignore all *.txt* files:

```
bandit31@bandit:/tmp/bandit31/repo$ ls -la
total 24
drwxr-sr-x 3 bandit31 root 4096 Sep 13 17:08 .
drwxr-sr-x 3 bandit31 root 4096 Sep 13 17:06 ..
drwxr-sr-x 8 bandit31 root 4096 Sep 13 17:08 .git
-rw-r--r-- 1 bandit31 root    6 Sep 13 17:07 .gitignore
-rw-r--r-- 1 bandit31 root   15 Sep 13 17:08 key.txt
-rw-r--r-- 1 bandit31 root  147 Sep 13 17:07 README.md
bandit31@bandit:/tmp/bandit31/repo$ cat .gitignore
*.txt
bandit31@bandit:/tmp/bandit31/repo$
```

Let's remove it and do it again.

```
bandit31@bandit:/tmp/bandit31/repo$ rm .gitignore
bandit31@bandit:/tmp/bandit31/repo$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        key.txt

no changes added to commit (use "git add" and/or "git commit -a")
bandit31@bandit:/tmp/bandit31/repo$ git add key.txt
bandit31@bandit:/tmp/bandit31/repo$ git commit -m "May I come in?"
[master d505a81] May I come in?
 1 file changed, 1 insertion(+)
 create mode 100644 key.txt
bandit31@bandit:/tmp/bandit31/repo$ git push
```

When pushing the repository, we get the following message:

```
remote: .o0o.o0o.o0o.o0o.o0o.o0o.o0o.o0o.o0o.o0o.
remote:
remote: Well done! Here is the password for the next level:
remote: 56a9bf19c63d650ce78e6ec0354ee45e
remote:
remote: .o0o.o0o.o0o.o0o.o0o.o0o.o0o.o0o.o0o.o0o.
```

LEVEL 32

In level 32, the user's shell is not a regular one. It changes every input and turns it into uppercase, so it is not recognised as a command.

We must make use of the *$0* variable. This variable stores the name of the script being executed (like *bash*, for example). As the value of the variable is in lowercase, calling it will run the script (*sh* in this case).

```
WELCOME TO THE UPPERCASE SHELL
>> $0
$ echo "$0"
sh
$
```

```
WELCOME TO THE UPPERCASE SHELL
>> $0
$ pwd
/home/bandit32
$ cat /etc/bandit_pass/bandit33
c9c3199ddf4121b10cf581a98d51caee
$
```

LEVEL 33

End of the game!!