Mark Selitenrikov

OverTheWire - Bandit

Bandit: https://overthewire.org/wargames/bandit/

Level 0

Level Goal

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

This level is quite easy, all you have to do is to ssh to the correct port with the given credentials.

```
┌──(kali㉿kali)-[~]
└─$ ssh bandit0@bandit.labs.overthewire.org -p 2220
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames

bandit0@bandit.labs.overthewire.org's password:
```

And we're in!

```
  Enjoy your stay!

bandit0@bandit:~$
```

Level 0 -> level 1

Level Goal

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

I am already in. I just need to read that "readme" file and ssh as bandit1 to the same domain and same port.

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

Now I'll ssh and paste this password when I log in

```
┌──(kali㉿kali)-[~]
└─$ ssh bandit1@bandit.labs.overthewire.org -p 2220
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames

bandit1@bandit.labs.overthewire.org's password:
```

And I'm in!

```
bandit1@bandit:~$
```

Level 1 -> level 2

Level Goal

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

This one is a bit tricky here's why:

```
bandit1@bandit:~$ ls -alh
total 24K
-rw-r-----  1 bandit2 bandit1   33 May  7  2020 -
drwxr-xr-x  2 root    root    4.0K May  7  2020 .
drwxr-xr-x 41 root    root    4.0K May  7  2020 ..
-rw-r--r--  1 root    root     220 May 15  2017 .bash_logout
-rw-r--r--  1 root    root    3.5K May 15  2017 .bashrc
-rw-r--r--  1 root    root     675 May 15  2017 .profile
```

The file we want to read (which contains the password) is in a file named "-". So in order to read it we cant just use "cat" normally.

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

We must use "./" so it won't refer to "-" as just bash syntax.

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

Now as in previous levels we ssh to the next user (bandit2) with the given password.

```
bandit2@bandit:~$
```

Level 2 -> level 3

Level Goal

Donate!

The password for the next level is stored in a file called **spaces in this filename** located in the home directory

For this one you solve it like this:

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

Because the filename has spaces, you can use either quotes or backslash in order to regard the spaces as a part of a string.

Now we just ssh to bandit3 with this password.

```
bandit3@bandit:~$
```

We're in :)

Level 3 -> level 4

Level Goal

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

This one wasn't hard as well. When there's a "." before a file or a directory, they are treated as hidden by the OS. meaning you need to use "-a" to view all files (including hidden files).

```
bandit3@bandit:~$ ls -alh
total 24K
drwxr-xr-x  3 root root 4.0K May  7  2020 .
drwxr-xr-x 41 root root 4.0K May  7  2020 ..
-rw-r--r--  1 root root  220 May 15  2017 .bash_logout
-rw-r--r--  1 root root 3.5K May 15  2017 .bashrc
drwxr-xr-x  2 root root 4.0K May  7  2020 inhere
-rw-r--r--  1 root root  675 May 15  2017 .profile
bandit3@bandit:~$ cd inhere/
bandit3@bandit:~/inhere$ ls -alh
total 12K
drwxr-xr-x 2 root    root    4.0K May  7  2020 .
drwxr-xr-x 3 root    root    4.0K May  7  2020 ..
-rw-r----- 1 bandit4 bandit3   33 May  7  2020 .hidden
bandit3@bandit:~/inhere$ cat .hidden
pIwrPrtPN36QITSp3EQaw936yaFoFgAB
```

And now ssh to the next user:

```
bandit4@bandit:~$
```

Level 4 -> level 5

Level Goal

The password for the next level is stored in the only human-readable file in the **inhere** directory. Tip: if your terminal is messed up, try the "reset" command.

When listing the "inhere" directory you see these files:

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

I wanted to figure out what type of file each one of them is (because of the hint):

```
bandit4@bandit:~/inhere$ for i in ./* ;do file $i;done;
./-file00: data
./-file01: data
./-file02: data
./-file03: data
./-file04: data
./-file05: data
./-file06: data
./-file07: ASCII text
./-file08: data
./-file09: data
```

And that pretty much solved the challenge

```
bandit4@bandit:~/inhere$ cat ./-file07
koReBOKuIDDepwhWk7jZC0RTdopnAYKh
```

Level 5 -> level 6

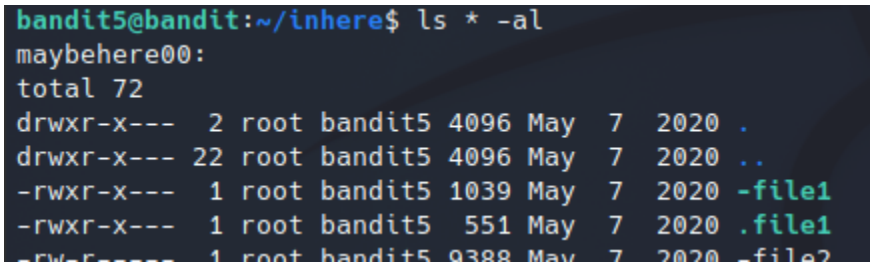
Level Goal

Donate!

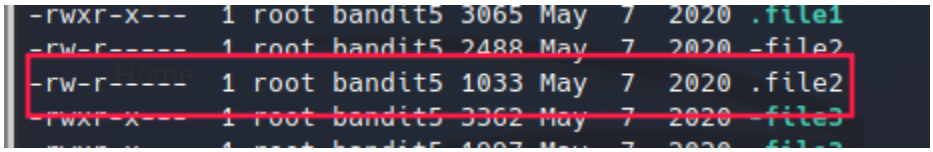
The password for the next level is stored in a file somewhere under the **inhere** directory and has all of the following properties:

- human-readable
- 1033 bytes in size
- not executable

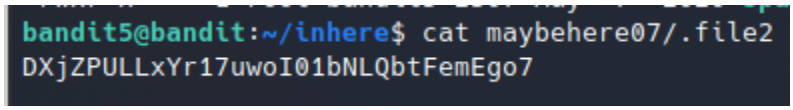
For this challenge we are supposed to find this a file containing a password with the requirements above.



Then I scrolled down and saw this:



Then I examined what's inside it.



And boom!

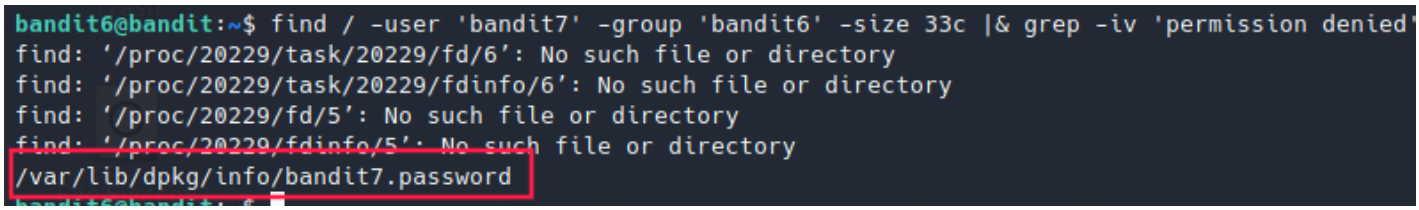
Level 6 -> level 7

Level Goal

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

- owned by user bandit7
- owned by group bandit6
- 33 bytes in size

For this challenge I constructed a "find" command to answer to all these options.



And here's the password:

```
bandit6@bandit:~$ cat /var/lib/dpkg/info/bandit7.password
HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs
```

Level 7 -> level 8

Level Goal

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

Okay before going overkill on searching the file I checked the current directory.

```
bandit7@bandit:~$ ls -al
total 4108
drwxr-xr-x  2 root    root       4096 May  7  2020 .
drwxr-xr-x 41 root    root       4096 May  7  2020 ..
-rw-r--r--  1 root    root        220 May 15  2017 .bash_logout
-rw-r--r--  1 root    root       3526 May 15  2017 .bashrc
-rw-r-----  1 bandit8 bandit7 4184396 May  7  2020 data.txt
-rw-r--r--  1 root    root        675 May 15  2017 .profile
```

Then I used strings on it with grep to see how near the password is to it.

```
bandit7@bandit:~$ strings data.txt | grep millionth -C 3
whirr   a99nLztiXHlqSqxwCicQAKgT1c8zO8rC
covenants       wRcmgvIJTRSgpV1iurw1gc7Ar2IU1EVQ
Halley  H7Mg53D6bPDpleFYGp1KF1SKTQh7jiNl
millionth       cvX2JJa4CFALtqS87jk27qwqGhBM9plV
shied   OfMI7PpeOvra4NWlZz7JOzyjL236NFVF
sesame  K1M1XuyPoCOkuEz6QB9gsyCW9dUqGXKx
Elul    FmCv0XrAW75I468ELOulmg7lGEslnUFL
```

Very close apparently lol.

Level 8 -> level 9

Level Goal

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

Okay first lets see how many words/lines we are dealing with.

```
bandit8@bandit:~$ wc data.txt
 1001  1001 33033 data.txt
```

Okay, so to solve this I've found this method:

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

As you can see that's the correct password! What happens here is this:

String data.txt (prints all the strings in the file)
Sort (sorts alphanumerically)
Uniq -u (prints only unique lines based on lines right before or right after it)

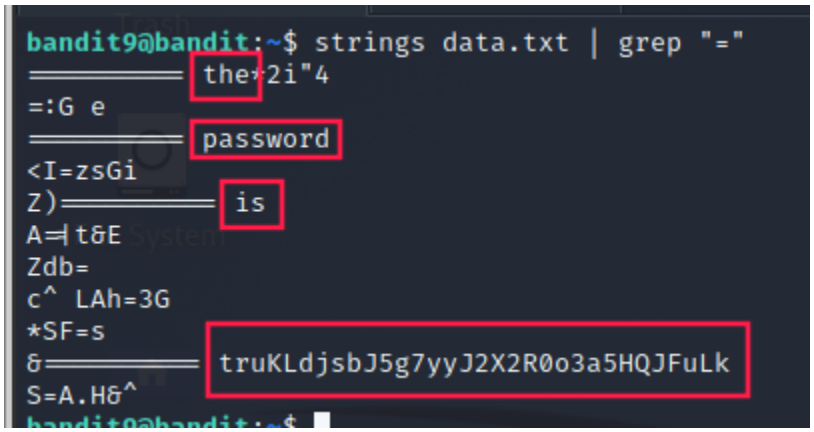
Level 9 -> level 10

Level Goal

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

Commands you may need to solve this level

Okay this time "data.txt" is also in the current directory. With this hint I've constructed this command:



I found it very funny how they decided to tell me that this is the password :)

Level 10 -> level 11

Level Goal

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

Okay so for this there's a handy command to decode base64 which is called "base 64" (lol)

```
bandit10@bandit:~$ ls
data.txt
bandit10@bandit:~$ cat data.txt
VGhlIHBhc3N3b3JkIGlzIElGdWt3S0dzRlc4TU9xM0lSRnFyeEUxaHhUTkViVVBSCg==
bandit10@bandit:~$ base64 -d data.txt
The password is IFukwKGsFW8MOq3IRFqrxE1hxTNEbUPR
bandit10@bandit:~$
```

Level 11 -> level 12

Level Goal

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

So I investigated the contents.

```
bandit11@bandit:~$ ls
data.txt
bandit11@bandit:~$ cat data.txt
Gur cnffjbeq vf 5Gr8L4qetPEsPk8htqjhRK8XSP6x2RHh
bandit11@bandit:~$
```

We are told this is ROT13 cypher so I wrote this script in python3 to decrypt the cypher text:

```python
import string

text = 'Gur cnffjbeq vf 5Gr8L4qetPEsPk8htqjhRK8XSP6x2RHh'

# generate a list of chars
lower = list(string.ascii_lowercase)
upper = list(string.ascii_uppercase)

# itirate over the text
for i in text:

    # if its lowercase
    if i in lower:
        index = lower.index(i)

        index += 13

        # if it passes the 26'th letter it resets to the beggining
        if index > 25:
            index -= 26

        print(lower[index], end='')

    # if its uppercase
    elif i in upper:
        index = upper.index(i)

        index += 13

        # if it passes the 26'th letter it resets to the beggining
        if index > 25:
            index -= 26

        print(upper[index], end='')

    # print original char if its space or something else...
    else:
        print(i, end='')

print()
```

And this is the result:

```
┌──(kali㉿kali)-[~]
└─$ python3 rot13.py
The password is 5Te8Y4drgCRfCx8ugdwuEX8KFC6k2EUu
```

Level 12 -> level 13

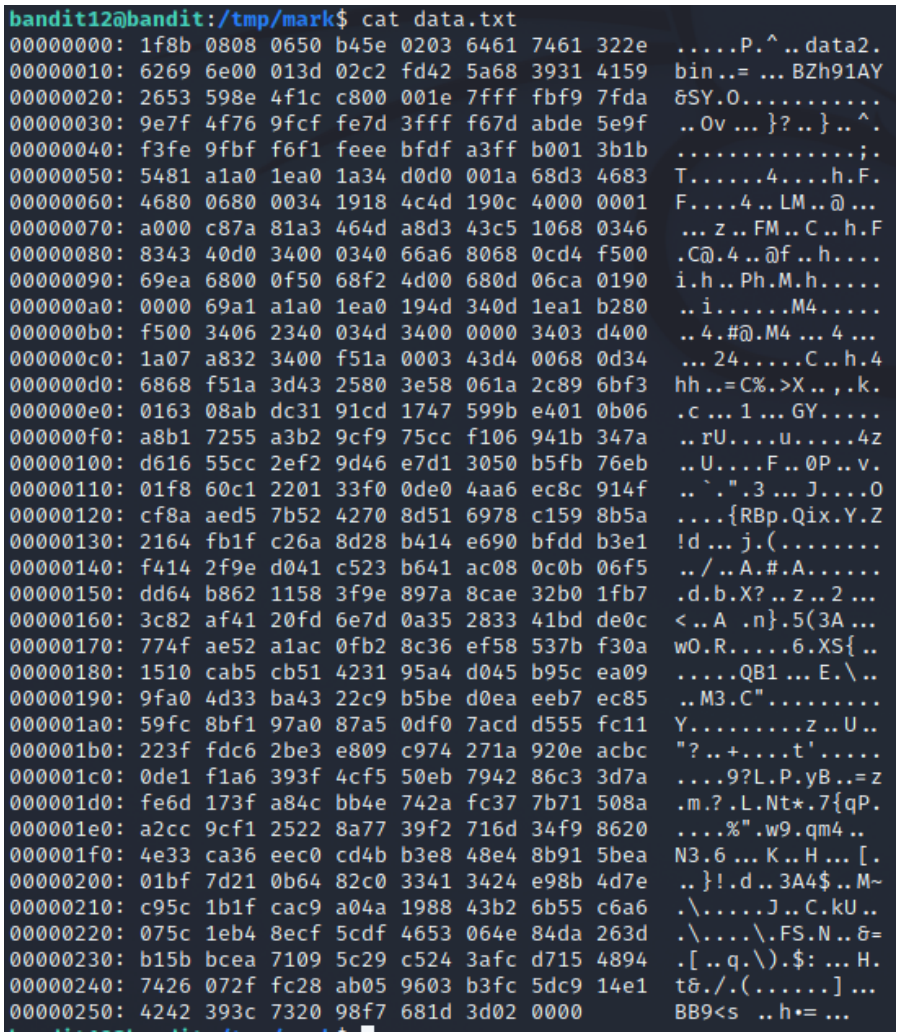
Level Goal

The password for the next level is stored in the file **data.txt**, which is a hexdump of a file that has been repeatedly compressed. For this level it may be useful to create a directory under /tmp in which you can work using mkdir. For example: mkdir /tmp/myname123. Then copy the datafile using cp, and rename it using mv (read the manpages!)

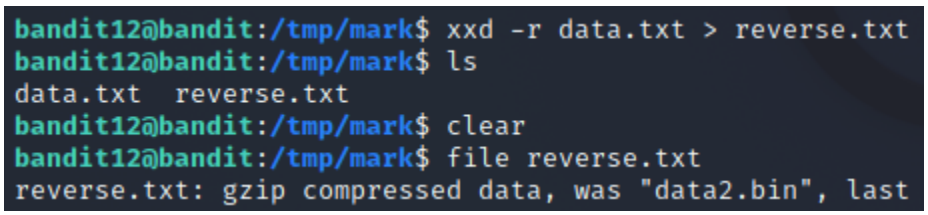
We are told we can go to /tmp and try to analyze and try to inflate it there:

```
bandit12@bandit:/tmp/mark$ ls
data.txt
```

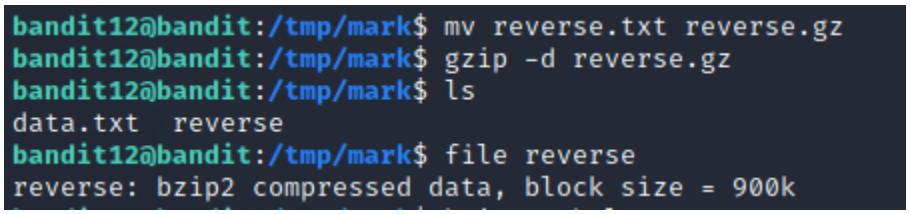
“data.txt” is a hex dump.



There’s a way to reverse a hex dump in linux to the original file like so:



Now I made it so I can inflate it in gzip.



Now when you get a bzip file and do the same.

```
bandit12@bandit:/tmp/mark$ bzip2 -d reverse
bzip2: Can't guess original name for reverse -- using reverse.out
bandit12@bandit:/tmp/mark$ ls
data.txt  reverse.out
bandit12@bandit:/tmp/mark$ file reverse.out
reverse.out: gzip compressed data, was "data4.bin", last modified: Th
ix
```

You continue doing like so (not for too long) til you get to the last file:

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

(It can be solved more efficiently with a script, but after solving I discovered that I can use vim inside “/tmp”)

Level 13 -> level 14

Level Goal

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

For this challenge we are told to connect to the same machine with the ssh key that we have in bandit13’s home directory.

```
bandit13@bandit:~$ ls
sshkey.private
```

Now we just ssh with the key.

```
bandit13@bandit:~$ ssh bandit14@localhost -i sshkey.private
Could not create directory '/home/bandit13/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:98UL0ZWr85496EtCRkKlo20X3OPnyPSB5tB5RP
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/bandit13/.ssh
```

We’re in!

```
  Enjoy your stay!

bandit14@bandit:~$
```

Level 14 -> level 15

Level Goal

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

First, we know the password for the current user is located in "/etc/bandit_pass/bandit14" from the previous challenge.

```
bandit14@bandit:~$ cat /etc/bandit_pass/bandit14
4wcYUJFw0k0XLShlDzztnTBHiqxU3b3e
bandit14@bandit:~$
```

Now what we must do is submit it to localhost on port 30000, I'll be doing that with netcat.

```
bandit14@bandit:~$ nc localhost 30000
4wcYUJFw0k0XLShlDzztnTBHiqxU3b3e
Correct!
BfMYroe26WYalil77FoDi9qh59eK5xNr
```

Level 15 -> level 16

Level Goal

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

Helpful note: Getting "HEARTBEATING" and "Read R BLOCK"? Use -ign_eof and read the "CONNECTED COMMANDS" section in the manpage. Next to 'R' and 'Q', the 'B' command also works in this version of that command...

I didn't know much openssl before trying this challenge. But after reading some of the "help" and research on the web i found this method:

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

At first I thought that I need to create a certificate, but apparently it has a determined certificate already. Then just submitted the last password:

```
BfMYroe26WYalil77FoDi9qh59eK5xNr
Correct!
cluFn7wTiGryunymYOu4RcffSxQluehd

closed
bandit15@bandit:~$
```

Level 16 -> level 17

Level Goal

Donate!

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

So my first reaction was to use nmap for this.

```
bandit16@bandit:~$ nmap -sV localhost -p 31000-32000

Starting Nmap 7.40 ( https://nmap.org ) at 2021-06-20 19:25 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00023s latency).
```

These are the results:

```
Not shown: 996 closed ports
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
```

So there are 2 ports with ssl. Ill try them in order.

```
bandit16@bandit:~$ openssl s_client -connect localhost:31518
CONNECTED(00000003)
depth=0 CN = localhost
verify error:num=18:self signed certificate
```

```
cluFn7wTiGryunymYOu4RcffSxQluehd
cluFn7wTiGryunymYOu4RcffSxQluehd
```

That one wasn't correct so lets try the next one!

```
bandit16@bandit:~$ openssl s_client -connect localhost:31790
CONNECTED(00000003)
depth=0 CN = localhost
verify error:num=18:self signed certificate
```

```
cluFn7wTiGryunymYOu4RcffSxQluehd
Correct!
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAvmOkuifmMg6HL2YPIOjon6iWfbp7c3jx34YkYWqUH57SUdyJ
imZzeyGC0gtZPGujUSxiJSWI/oTqexh+cAMTSMlOJf7+BrJObArnxd9Y7YT2bRPQ
Ja6Lzb558YW3FZl87ORiO+rW4LCDCNd2lUvLE/GL2GWyuKN0K5iCd5TbtJzEkQTu
DSt2mcNn4rhAL+JFr56o4T6z8WWAW18BR6yGrMq7Q/kALHYW3OekePQAzL0VUYbW
```

We are correct and we are given an ssh key! Lets try copying it and connecting to the next level.

```
┌──(root💀kali)-[/home/kali]
└─# ssh bandit17@bandit.labs.overthewire.org -p 2220 -i ssh_key_level_17
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames
```

```
bandit17@bandit:~$
```

(Don’t forget to set chmod to 640)

Level 17 -> level 18

Level Goal

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

NOTE: if you have solved this level and see ‘Byebye!’ when trying to log into bandit18, this is related to the next level, bandit19

For this challenge what I’ve done was use a solution for one of the earlier challenges. But I concatenated 2 files this time in the output:

```
bandit17@bandit:~$ cat passwords.old passwords.new | sort | uniq -u
kfBf3eYk5BPBRzwjqutbbfE887SVc5Yd
w0Yfolrc5bwjS4qw5mq1nnQi6mF03bii
bandit17@bandit:~$
```

As the instructions in the challenge, the first password doesn’t work:

```
Byebye !
Connection to bandit.l

┌──(kali㉿kali)-[~]
└─$
```

We actually managed to connect but we get this message because of the next challenge. (I tried the "**diff**") command from the hints and apparently you can solve this way also!

Level 18 -> level 19

Level Goal

Donate!

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

So for this challenge I've learned a cool trick. The problem was that when ".bashrc" is sourced, it logs the user out! So I found a way to log in without sourcing ".bashrc"! Like so:

```
┌──(kali㉿kali)-[~]
└─$ ssh bandit18@bandit.labs.overthewire.org -p 2220 bash --norc
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames

bandit18@bandit.labs.overthewire.org's password:
ls
readme
```

Now lets see the contents of "readme"

```
cat readme
IueksS7Ubh8G3DCwVzrTd8rAVOwq3M5x
```

done! :)

Level 19 -> level 20

Level Goal

Donate!

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.

We get an executable that has the SUID bit set:

```
total 8
-rwsr-x--- 1 bandit20 bandit19 7296 May  7  2020 bandit20-do
bandit19@bandit:~$
```

When trying to run it with no parameters:

```
bandit19@bandit:~$ ./bandit20-do
Run a command as another user.
  Example: ./bandit20-do id
bandit19@bandit:~$
```

We are hinted that the password is located in "/etc/bandit_pass/". And to see bandit20's password you probably have to go to his password file using the permissions you get from the SUID file.

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

GbKksEFF4yrVs6il55v6gwY5aVje5f0j

That's the password!

Level 20 -> level 21

Level Goal

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

NOTE: Try connecting to your own network daemon to see if it works as you think

You get this SUID:

```
bandit20@bandit:~$ ls
suconnect
```

When running it, it explains what it does:

```
bandit20@bandit:~$ ./suconnect
Usage: ./suconnect <portnumber>
This program will connect to the given port on localhost using TCP. If it receives the correct password from the othe
r side, the next password is transmitted back.
```

Now I know that it creates a connection, So I assumed I can do it with netcat. I logged in as bandit19 again and listened on port 4444 which I chose randomly:

```
bandit19@bandit:~$ nc -lnvp 4444
listening on [any] 4444 ...
```

Then I connected to this port via the SUID binary on bandit20:

```
bandit20@bandit:~$ ./suconnect 4444
```

Then it connected to the port that I was already listening on so I submitted the password and received the password for the next level:

```
bandit19@bandit:~$ nc -lnvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 41272
GbKksEFF4yrVs6il55v6gwY5aVje5f0j
gE269g2h3mw3pwgrj0Ha9Uoqen1c9DGr
```

Done! This is how it looked on bandit20's shell:

```
bandit20@bandit:~$ ./suconnect 4444
Read: GbKksEFF4yrVs6il55v6gwY5aVje5f0j
Password matches, sending next password
```

Level 21 -> Level 22

Level Goal

Don

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

So as per the goal I changed directory to /etc/cron.d/ and saw this:

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

I'm trying to advance to bandit22 so I checked out what's inside the bandit22 cronjob config.

```
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$
```

As you can see it executes a shell script (".sh"). So Lets see what's inside it!

```
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$
```

As you can see, it prints bandit22's password to "/tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv". So all we have to do is try to print it to the terminal.

```
bandit21@bandit:/etc/cron.d$ cat /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
Yk7owGAcWjwMVRwrTesJEwB7WVOiILLI
bandit21@bandit:/etc/cron.d$
```

Bom! That's the password :)

Level 22 -> Level 23

Level Goal

Donate!

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

NOTE: Looking at shell scripts written by other people is a very useful skill. The script for this level is intentionally made easy to read. If you are having problems understanding what it does, try executing it to see the debug information it prints.

Based on the goal I checked the corresponding cronjob:

```
bandit22@bandit:/etc/cron.d$ ls
cronjob_bandit15_root  cronjob_bandit22  cronjob_bandit24
cronjob_bandit17_root  cronjob_bandit23  cronjob_bandit25_root
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
```

I can see this script is being executed on reboot and every minute. Then I decided to see if I can read the script:

```
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
```

I can see that it uses the “whoami” command, which when executed by bandit23 would give us “bandit23”.
Then it echos a string with “bandit23” inserted into it. Later it gets hashed in MD5 and the spaces get cut.
When all of it is done, it creates a file in “/tmp” with the hash as it’s name. Later it prints the password of bandit23 into that same file. The easiest way to solve it is like so:

```
bandit22@bandit:/etc/cron.d$ target=$(echo I am user bandit23 | md5sum | cut -d ' ' -f 1)
bandit22@bandit:/etc/cron.d$ cat /tmp/$target
jc1udXuA1tiHqjIsL8yaapX5XIAI6i0n
bandit22@bandit:/etc/cron.d$
```

That’s the password! What I did was just recreate the hash in the same way it is done in the script.

Level 23 -> Level 24

Level Goal

Donate!

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

NOTE: This level requires you to create your own first shell-script. This is a very big step and you should be proud of yourself when you beat this level!

NOTE 2: Keep in mind that your shell script is removed once executed, so you may want to keep a copy around...

Okay for this level we need to figure out the cron job:

```
bandit23@bandit:/tmp/mark-test$ cat /etc/cron.d/cronjob_bandit24
@reboot bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
* * * * * bandit24 /usr/bin/cronjob_bandit24.sh &> /dev/null
```

Now we know what script is executed… lets see what it does:

```
bandit23@bandit:/tmp/mark-test$ 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
```

We can see that it executes everything in “/var/spool/bandit24” (whoami by cron would give “bandit24”).
Now let’s write a script to use bandit24’s privileges to view “/etc/bandit_pass/bandit24” that we know from previous challenges that only the corresponding user can view it’s own password. So this is the script:

```
1 #!/bin/bash
2
3 cat /etc/bandit_pass/bandit24 > /tmp/answer.txt
4
```

It would work because it is executed with bandit24's privileges. Now we make it an executable.

```
bandit23@bandit:/tmp/mark-test$ ls
bandit.sh
bandit23@bandit:/tmp/mark-test$ chmod +x bandit.sh
bandit23@bandit:/tmp/mark-test$ ls
bandit.sh
bandit23@bandit:/tmp/mark-test$
```

And lets copy it to the directory that has it's scripts executed.

```
bandit23@bandit:/tmp/mark-test$ cp bandit.sh /var/spool/bandit24
```

After a minute or less we get this:

```
bandit23@bandit:/tmp/mark-test$ cat /tmp/answer.txt
UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ
```

That's the password! :)

Level 24 -> Level 25

Level Goal

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

Okay, a 10000 combinations pincode would be very very fast to bruteforce. So lets see how we go about it, first a test:

```
bandit24@bandit:~$ nc localhost 30002
I am the pincode checker for user bandit25. Please enter the password for user bandit24 and the secret pincode on a s
ingle line, separated by a space.
UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ 0000
Wrong! Please enter the correct pincode. Try again.
0001
Fail! You did not supply enough data. Try again.
0002
Fail! You did not supply enough data. Try again.
0003
Fail! You did not supply enough data. Try again.
```

As you can see, there's no limit to how many times I'm allowed to try different combinations before I get disconnected. So I wrote this script:

```
#!/bin/bash

for i in {0000..9999}; do

    echo UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ $i

done | nc localhost 30002 |& grep -iv 'Wrong! Please enter the correct pincode. Try again.'

```

I've created a "for" loop that sends strings to the netcat connection with the $i parameter that is all possible 4 digits numbers combinations. In the end I also added a grep to ignore the "wrong" feedback I get from the connection in order to have less clutter in the terminal.

```
bandit24@bandit:/tmp/mark-lvl24$ ./script.sh
I am the pincode checker for user bandit25. Please enter the password for user bandit24 and the secret pincode on a s
ingle line, separated by a space.
Correct!
The password of user bandit25 is uNG9O58gUE7snukf3bvZ0rxhtnjzSGzG

Exiting.
```

That's the password! :)
The main point of the challenge is to manage to send text in a loop to the netcat connection. Which is done by adding a pipeline at "done".

Level 25 -> level 26

Level Goal

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

The first thing you see when you log in as bandit25 is bandit26's ssh key:

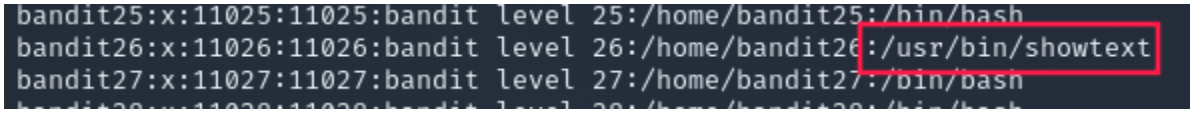
```
bandit25@bandit:~$ ls -al
total 32
drwxr-xr-x  2 root     root     4096 May 14  2020 .
drwxr-xr-x 41 root     root     4096 May  7  2020 ..
-rw-r-----  1 bandit25 bandit25   33 May 14  2020 .bandit24.password
-r--------  1 bandit25 bandit25 1679 May  7  2020 bandit26.sshkey
-rw-r--r--  1 root     root      220 May 15  2017 .bash_logout
-rw-r--r--  1 root     root     3526 May 15  2017 .bashrc
-rw-r-----  1 bandit25 bandit25    4 May 14  2020 .pin
-rw-r--r--  1 root     root      675 May 15  2017 .profile
bandit25@bandit:~$ cat bandit26.sshkey
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEApis2AuoooEqeYWamtwX2k5z9uU1Afl2F8VyXQqbv/LTrIwdW
pTfaeRHXzr0Y0a5Oe3GB/+W2+PReif+bPZlzTY1XFwpk+DiHk1kmL0moEW8HJuT9
/5XbnpjSzn0eEAfFax2OcopjrzVqdBJQerkj0puv3UXY07AskgkyD5XepwGAlJOG
xZsMq1oZqQ0W29aBtfykuGie2bxroRjuAPrYM4o3MMmtlNE5fC4G9Ihq0eq73MDi
1ze6d2jIGce873qxn308BA2qhRPJNEbnPev5gI+5tU+UxebW8KLbk0EhoXB953Ix
```

So my first reaction was to copy it and try to log in with it:

```
┌──(root💀kali)-[/home/kali]
└─# ssh bandit26@bandit.labs.overthewire.org -p 2220 -i bandit26key
```

But sadly the connection gets closed automatically.

So, back to bandit25. I checked what shell bandit26 is using.



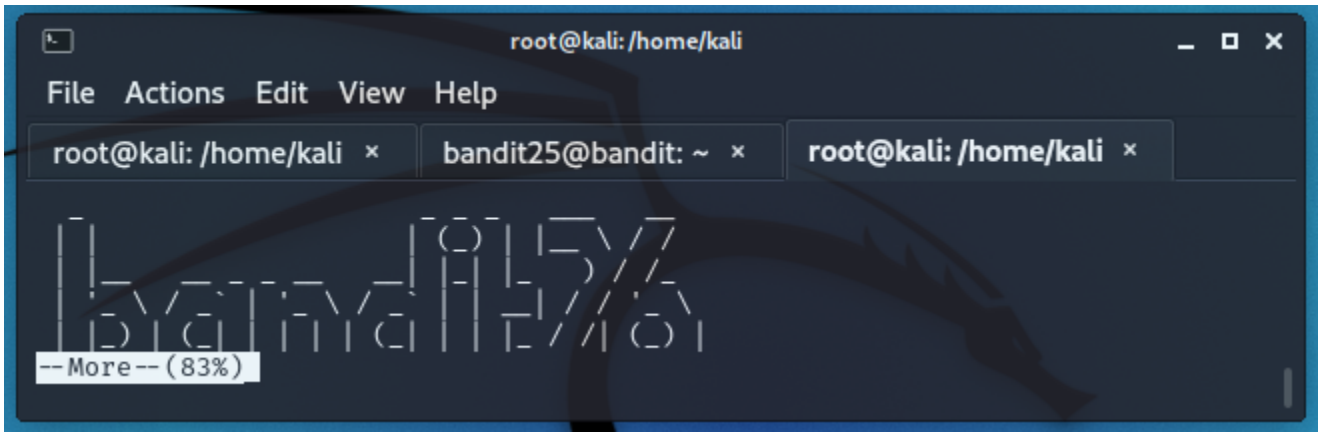
In “/etc/passwd” we can see the default shell which is “/usr/bin/showtext”. So I decided to see it’s source:

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

export TERM=linux

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

You can see that it just prints a text and closes. The text is most likely the “bandit26” ASCII art. Strangely it’s not using “cat”. So I investigated how can I use “more” in order to get a shell. If you run more (i did “man more | more”). And press h you can see this:

```
Most commands optionally preceded by integer argument k.  Defaults in brackets.
Star (*) indicates argument becomes new default.
---------------------------------------------------------------------------
<space>                 Display next k lines of text [current screen size]
z                       Display next k lines of text [current screen size]*
<return>                Display next k lines of text [1]*
d or ctrl-D             Scroll k lines [current scroll size, initially 11]*
q or Q or <interrupt>   Exit from more
s                       Skip forward k lines of text [1]
f                       Skip forward k screenfuls of text [1]
b or ctrl-B             Skip backwards k screenfuls of text [1]
'                       Go to place where previous search started
=                       Display current line number
/<regular expression>   Search for kth occurrence of regular expression [1]
n                       Search for kth occurrence of last r.e [1]
!<cmd> or :!<cmd>       Execute <cmd> in a subshell
v                       Start up '/usr/bin/vi' at current line
ctrl-L                  Redraw screen
:n                      Go to kth next file [1]
:p                      Go to kth previous file [1]
:f                      Display current file name and line number
.                       Repeat previous command
---------------------------------------------------------------------------
```

I can either try to spawn a shell from “more” or conduct an escape sequence from vim. So first I tried spawning a shell from more. (You must resize the shell to trigger “more”)

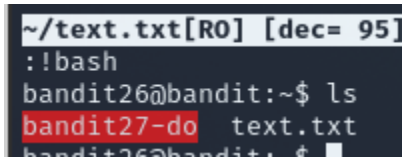


From more directly I couldn’t spawn a different shell or change default shell via “chsh”. So I moved on to vim. From vim I also tried spawning a shell:

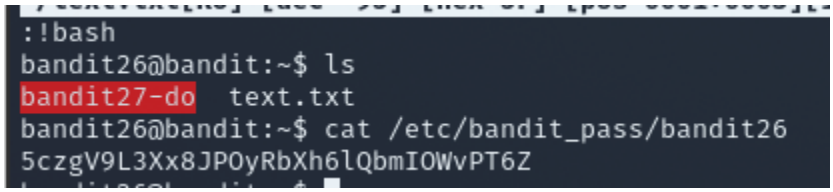
After further Investigating I found that you can set a shell that vim will spawn like so:

ls

And I'm in!



Here's the password (If I'll need it for future use):

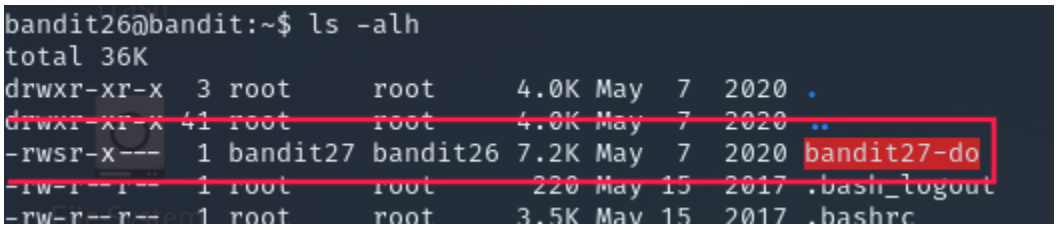


Level 26 -> level 27

Level Goal

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

This level was weirdly easy… What we get is a SUID file that let us run commands as bandit27



Like in a level I've done before I investigated the passwords directory for bandit27's password.

```
bandit26@bandit:~$ ./bandit27-do cat /etc/bandit_pass/bandit27
3ba3118a22e93127a4ed485be72ef5ea
bandit26@bandit:~$
```

Level 27 -> Level 28

Level Goal

There is a git repository at `ssh://bandit27-git@localhost/home/bandit27-git/repo`. The password for the user `bandit27-git` is the same as for the user `bandit27`.

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

I kind of laughed at myself at this level. I couldn't understand why I couldn't clone the git repo.

```
bandit27@bandit:~$ git clone ssh://bandit27-git@localhost/home/bandit27-git/repo
fatal: could not create work tree dir 'repo': Permission denied
```

I even tried to ssh into bandit-git and the connection would be closed because interactive git shell wasn't enabled. Then I decided to test a feeling I had:

```
bandit27@bandit:~$ touch test.txt
touch: cannot touch 'test.txt': Permission denied
```

I found the reason, I can't add files or directories to bandit27's home directory! So I created one in /tmp.

```
bandit27@bandit:~$ mkdir /tmp/mark-lvl27
bandit27@bandit:~$ cd /tmp/mark-lvl27
bandit27@bandit:/tmp/mark-lvl27$
```

Now just clone the repo.

```
bandit27@bandit:/tmp/mark-lvl27$ git clone ssh://bandit27-git@localhost/home/bandit27-git/repo
Cloning into 'repo'...
Could not create directory '/home/bandit27/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be established.
```

Here's the password!

```
bandit27@bandit:/tmp/mark-lvl27/repo$ cat README
The password to the next level is: 0ef186ac70e04ea33b4c1853d2526fa2
```

Level 28 -> level 29

Level Goal

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

When cloning the git repo you get a single file.

```
bandit28@bandit:/tmp/mark28/repo$ ls
README.md
```

After inspecting it I found something interesting.

```
bandit28@bandit:/tmp/mark28/repo$ cat README.md
# Bandit Notes
Some notes for level29 of bandit.

## credentials

- username: bandit29
- password: xxxxxxxxxx
```

This file has credentials but they are missing! I've decided to see older commits:

```
bandit28@bandit:/tmp/mark28/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
```

You can see an interesting commit; "fix info leak". This means that a commit before it had that issue. Lets see that commit.

```
bandit28@bandit:/tmp/mark28/repo$ git checkout c086d11a00c0648d095d04c089786efef5e01264
Note: checking out 'c086d11a00c0648d095d04c089786efef5e01264'.
```

And here's the password in cleartext:

```
bandit28@bandit:/tmp/mark28/repo$ cat README.md
# Bandit Notes
Some notes for level29 of bandit.

## credentials

- username: bandit29
- password: bbc96594b4e001778eee9975372716b2
```

Level 29 -> level 30

Level Goal

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

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

When you clone the repository you get the README.md file:

```
bandit29@bandit:/tmp/mark29/repo$ ls
README.md
bandit29@bandit:/tmp/mark29/repo$ cat README.md
# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

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

From what is written I can understand that the password is not located on production. Meaning, not on the master branch.

```
bandit29@bandit:/tmp/mark29/repo$ git branch -a
  dev
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev
  remotes/origin/master
  remotes/origin/sploits-dev
bandit29@bandit:/tmp/mark29/repo$
```

I decided to check out the “dev” branch first. And this is what I’ve seen.

```
bandit29@bandit:/tmp/mark29/repo$ git checkout dev
Switched to branch 'dev'
```

```
bandit29@bandit:/tmp/mark29/repo$ cat README.md
# Bandit Notes
Some notes for bandit30 of bandit.

## credentials

- username: bandit30
- password: 5b90576bedb2cc04c86a9e924ce42faf
```

That’s it!

Level 30 -> level 31

Level Goal

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

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

While going over the git documentation and trying the “tag” option I found something interesting:

```
bandit30@bandit:/tmp/mark30/repo$ git tag
secret
```

So I’ve researched a bit about tags, you can actually use the “show” option on a tag name in order to get more information about it!
Source: https://www.freecodecamp.org/news/git-tag-explained-how-to-add-remove/

```
bandit30@bandit:/tmp/mark30/repo$ git show secret
47e603bb428404d265f59c42920d81e5
```

That’s the password!

Level 31 -> level 32

Level Goal

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

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

When inspecting the "README.md" file this time I saw this:

```
bandit31@bandit:/tmp/mark31/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
```

Basically I get an objective to push a specific file with specific content.

```
bandit31@bandit:/tmp/mark31/repo$ echo 'May I come in?' > key.txt
```

So then I tried to push it to the repository and saw this:

```
bandit31@bandit:/tmp/mark31/repo$ git add key.txt
The following paths are ignored by one of your .gitignore files:
key.txt
Use -f if you really want to add them.
```

After running it with the -f switch I managed to push it (it's also possible to just delete ".gitignore").

```
Total 3 (delta 0), reused 0 (delta 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: 56a9bf19c63d650ce78e6ec0354ee45e
remote:
remote: .oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.
remote:
```

And that's the password!

Level 32 -> level 33

Level Goal

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

Commands you may need to solve this level

When i first connected I got a weird shell:

```
WELCOME TO THE UPPERCASE SHELL
>>
```

The problem is that all chars get converted to uppercase!

```
>> ls
sh: 1: LS: not found
>> id
sh: 1: ID: not found
>> whoami
sh: 1: WHOAMI: not found
```

So I decided to play around a bit and got to the realization that I can use special characters.

```
>> ./*
WELCOME TO THE UPPERCASE SHELL
```

So I decided to try and get to bash.

```
>> /???/????
/bin/chvt: /bin/chvt: cannot execute binary file
```

I realized that I need to find a different approach. So i opened another ssh instance and snooped around with bandit31. I realized that I can refer to python with missing chars and numbers:

```
bandit31@bandit:~$ ls /usr/bin
[                                grub-editenv                     rake
 python2
 python2.7
 python2.7-config
 python2-config
 python3
 python3.5
 python3.5-config
 python3.5m
 python3.5m-config
 python3-config
 python3m
```

I've tried running these but they all gave errors except "/usr/bin/python3.5m". I've done it like so:

```
/bin/chvt: /bin/chvt: cannot execute binary file
>> /???/???/??????3.5?
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('test')
test
```

Great now I have a python shell! Now let's spawn a bash shell like so:

```
>>> import os;os.system("/bin/bash")
bandit33@bandit:~$
```

Nice! And we're bandit33?? This is why:

```
-rwsr-x--- 1 bandit33 bandit32 7556 May  7  2020 uppershell
```

"uppershell" is a SUID binary. When we ran it, we actually ran it with bandit33's privileges. Therefore we are treated as bandit33. And now with these permissions lets see bandit33's password:

```
bandit33@bandit:~$ cat /etc/bandit_pass/bandit33
c9c3199ddf4121b10cf581a98d51caee
```

On to the next level :)

Level 33 -> level 34

For this level we get a single "README.txt" file.

```
bandit33@bandit:~$ ls
README.txt
bandit33@bandit:~$ cat README.txt
Congratulations on solving the last level of this game!

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

If you have an idea for an awesome new level, please let us know!
```

We reached the finish line!