Final Project for INF 385T - Applied Encryption II

Write-Up documentation for Bandit challenges on the platform `OverTheWire`

Student: Sifan Guo

Instructor: Prof. Walker Riley

This is the link for Bandit wargame.

Bandit0

> 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.

Bandit1

> 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.

`Input: ls`

`Output: readme`

`Input: cat readme`

`Output: boJ9jbbUNNfktd78OOpsqOltutMc3MY1`

Here we go, we get the password for `bandi1`. I know it's weird to type every characters of the password. In fact, we could copy and paste the password when we log in by using `Ctrl + Shift + V`. It's worth noting that my putty doesn't work but the `Secure Shell App` from Google extension works.

Bandit2

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

`Input: ls`

`Output: -`

`Input: cat ./-`

`Output: CV1DtqXWVFXTvM2F0k09SHz0YwRINYA9`

When "-" is the filename, we have to tell the system that this "-" is not the dash before parameters(for instance, `cat -n 7 filename`). Otherwise the system would wait for the input.

Bandit3

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

`Input: ls`

`Output: spaces in this filename`

`Input: cat "spaces in this filename"`

`Output: UmHadQclWmgdLOKQ3YNgjWxGoRMb5luK`

Just like you have to add the quotes when you `git add "xxx xxx.file"`, we need to do that to tell the system this is a name of a file rather than separated commands.

Bandit4

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

`Input: ls`

`Output: inhere`

`Input: cd inhere/`

`Input: ls -a`

`Output: . .. .hidden`

`Input: cat .hidden`

`Output: pIwrPrtPN36QITSp3EQaw936yaFoFgAB`

Basically, here we use the "-a" after ls to list all contents including the hidden files.

Bandit5

> 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.

`Input: ls`

`Output: inhere`

`Input: cd inhere/`

`Input: ls`

`Output: -file00 -file01 -file02 -file03 -file04 -file05 -file06 -file07 -file08 -file09`

`Input: cat ./-file07`

`Output: koReBOKuIDDepwhWk7jZC0RTdopnAYKh`

It says the file is human-readable only, I just check every file using `cat ./-file0x` and it turns out "-file07" has the password for bandit5.

Bandit6

> 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

`Input: ls`

`Output: inhere`

`Input: cd inhere/`

`Input: ls`

`Output: maybehere00 maybehere02 ..... maybehere19`

`Input: find -type f -size 1033c`

`Output: ./maybehere07/.file2`

`Input: cat ./maybehere07/.file2`

`Output: DXjZPULLxYr17uwoI01bNLQbtFemEgo7`

Using find command with "-type f", we can find file instead of directory (use "-type d"). Then with "-size 1033c", we get files whose size is 1033 Bytes.

Bandit7

> 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

`Input: cd /`
change directory to the root

`Input: ls -Rla | grep "bandit7 bandit6"`

It returns a lot of lines, but we can find this line.

`Output: -rw-r----- 1 bandit7 bandit6 33 Oct 16 2018 bandit7.password`

`Input: find -name bandit7.password`

It returns a lot of lines, but we can find this line.

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

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

`Output: HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs`

The `ls` command can be used with many parameters at once. The "R" means recursively find the sub-directories in addition to current work directory and the "l" means list the details of that file or folder.

Bandit8

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

`Input: cat data.txt | grep millionth`

`Output: millionth cvX2JJa4CFALtqS87jk27qwqGhBM9plV`

The password for bandit8 is cvX2JJa4CFALtqS87jk27qwqGhBM9plV.

Bandit9

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

`Input: sort data.txt | uniq -c | grep "1 "`

`Output: 1 UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhR`

Or

`Input: sort data.txt | uniq -u`

`Output: UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhR`

Bandit10

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

`Input: cat data.txt | grep ====`

`Output: Binary file (standard input) matches`

`Input: strings data.txt | grep ====`

`Output: 2========== the ========== password ========== isa ========== truKLdjsbJ5g7yyJ2X2R0o3a5HQJFuLk`

When I use the `cat` command, it prompts that the file is a Binary one. Then we should use `strings` command because it returns each string of printable characters in files. Its main uses are to determine the contents of and to extract text from binary files (i.e., non-text files). The introduction page is here

Bandit11

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

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

`Output: The password is IFukwKGsFW8MOq3IRFqrxE1hxTNEbUPR`

The `base64` command can encode a string without parameter and decode that with "-d".

Bandit12

> 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

It's basically Caesar Cipher in this challenge. However, I didn't use my python script from last semester, cuz I want to try linux solution.

The `tr` command in UNIX is a command line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace. It can be used with UNIX pipes to support more complex translation. `tr` stands for translate.

Using `tr` command, we need to know what is the 14th letter of the alphabet (n). So it translate a to n, b to o and so on.

`Input: cat data.txt | tr a-zA-Z n-za-mN-ZA-M`

`Output: The password is 5Te8Y4drgCRfCx8ugdwuEX8KFC6k2EUu`

Bandit13

> 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!)

The `file` command can determine file type for a given file.

`Input: cp data.txt /tmp/sf/sfile`

`Input: cd /tmp/sf`

`Input: ls`

`Output: sfile`

`Input: file sfile`

`Output: sfile: ASCII text`

This file is a hexdump, so I have to do revert that transformation.

`Input: xxd -r sfile reverted`

`Input: ls`

`Output: reverted sfile`

`Input: file reverted`

`Output: reverted: gzip compressed data, was "data2.bin", last modified: Tue Oct 16 12:00:23 2018, max compression, from Unix`

So we change the file name and unzip that 'gzip compressed data'.

`Input: mv reverted reverted.gz`

`Input: gzip -d reverted.gz`

`Input: ls`

`Output: reverted sfile`

`Input: file reverted`

`Output: reverted: bzip2 compressed data, block size = 900k`

So we change the file name to bzip2 compressed data and unzip it.

`Input: mv reverted reverted.b2`

`Input: bzip2 -d reverted.b2`

`Output: bzip2: Can't guess original name for reverted.b2 -- using reverted.b2.out`

`Input: file reverted.b2.out`

`Output: reverted.b2.out: gzip compressed data, was "data4.bin"......`

Again, we change the file name to gz cuz it's gzip compressed.

`Input: mv reverted.b2.out reverted.gz`

`Input: gzip -d reverted.gz`

`Input: file reverted`

`Output: reverted: POSIX tar archive (GNU)`

The following process is basically the same, so I just screenshot the codes here.

```
bandit12@bandit:/tmp/sf$ ls
data5.bin  reverted.tar  sfile
bandit12@bandit:/tmp/sf$ file data5.bin
data5.bin: POSIX tar archive (GNU)
bandit12@bandit:/tmp/sf$ tar -xf data5.bin
bandit12@bandit:/tmp/sf$ ls
data5.bin  data6.bin  reverted.tar  sfile
bandit12@bandit:/tmp/sf$ file data6.bin
data6.bin: bzip2 compressed data, block size = 900k
bandit12@bandit:/tmp/sf$ tar -xf data6.bin
bandit12@bandit:/tmp/sf$ ls
data5.bin  data6.bin  data8.bin  reverted.tar  sfile
bandit12@bandit:/tmp/sf$ file data8.bin
data8.bin: gzip compressed data, was "data9.bin", last modified: Tue Oct 16 12:00:23 2018, max compression, from Unix
bandit12@bandit:/tmp/sf$ tar -xf data8.bin
bandit12@bandit:/tmp/sf$ ls
data5.bin  data6.bin  data8.bin  reverted.tar  sfile
bandit12@bandit:/tmp/sf$ file data8.bin
data8.bin: gzip compressed data, was "data9.bin", last modified: Tue Oct 16 12:00:23 2018, max compression, from Unix
bandit12@bandit:/tmp/sf$ mv data8.bin data8.gz
bandit12@bandit:/tmp/sf$ gzip -d data8.gz
bandit12@bandit:/tmp/sf$ ls
data5.bin  data6.bin  data8  reverted.tar  sfile
bandit12@bandit:/tmp/sf$ file data8
data8: ASCII text
bandit12@bandit:/tmp/sf$ cat data8
The password is 8ZjyCRiBWFYkneahHwxCv3wb2a1ORpYL
```

`Input: cat data8`
`Output: The password is 8ZjyCRiBWFYkneahHwxCv3wb2a1ORpYL`

Bandit14

> 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

`Input: ssh -i sshkey.private bandit14@localhost`
`Input: cat /etc/bandit_pass/bandit14`
`Output: 4wcYUJFw0k0XLShlDzztnTBHiqxU3b3e`

Bandit15

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

`Input: cat /etc/bandit_pass/bandit14 | nc localhost 30000`

`Output: BfMYroe26WYalil77FoDi9qh59eK5xNr`

We can use `echo` or `cat` to talk to server:30000. Useful resource here.

Bandit16

> 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.

`Input: openssl s_client -connect localhost:30001`

`Output: CONNECTED(00000003).............---`

Then we enter the password for the current level.

`Input: BfMYroe26WYalil77FoDi9qh59eK5xNr`

`Output: Correct! cluFn7wTiGryunymYOu4RcffSxQluehd`

Bandit17

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

`Input: nmap localhost -p 31000-32000`

`Output: as follows`

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

Starting Nmap 7.40 ( https://nmap.org ) at 2019-04-19 18:10 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00019s latency).
Not shown: 1000 closed ports
PORT      STATE SERVICE
31790/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.07 seconds
```

As w e can see from the screenshot, the port 31790 is open and other 1000 ports are closed. So w e use openssl to connect to that port.

```
---
cluFn7wTiGryunymYOu4RcffSxQluehd
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
```

Because w e cannot save file in the current folder, w e need to change directory to /tmp/, and then w e copy and paste the RSA key into a private key file. How ever w hen I input `ssh -i rsakey.private bandit17@localhost`, it prompts:

```
Permissions 0644 for 'rsakey.private' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "rsakey.private": bad permissions
```

It says that we should not allow others to have right for this private key, with that saying, here is what I find useful for `chmod` command. I'd like to briefly introduce the `chmod` command: u for user (or the owner); g for group; o for others; a for all above; + adds the permission; - removes the permission; = makes the onley permission; r for read; w for write; x for execute.

`Input: ssh -i rsakey.private bandit17@localhost`

`Output: Enjoy your stay!`

Bandit18

> 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

In order to get the difference of two files, I referred to diff which is similar to `git diff`

`Input: diff passwords.new passwords.old`

`Output: 42c42`
`< kfBf3eYk5BPBRzwjqutbbfE887SVc5Yd`
`---`
`> hlbSBPAWJmL6WFDb06gpTx1pPButblOA`

The "<" symbol indicates that this line `kfBf3eYk5BPBRzwjqutbbfE887SVc5Yd` is from `passwords.new`, which means this string is the password for bandit18.

Bandit19

> 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.

When I tried to login into bandit18@localhost, it says "Byebye" just after I logged in. So I searched on the Google and find this answer. Then, I add the argument here:

`Input: cat readme`

`Output: IueksS7Ubh8G3DCwVzrTd8rAVOwq3M5x`

It's worth noting that the `-t` parameter opens a pseudo-tty(TeleTYpewriter, TTY), which might mean the .bashrc still runs when we open this session. However, we open the `shell` by typing `/bin/sh` as the input. I also tried to look for `-t` in the manual and I noticed that `-T` parameter can `Disable pseudo-terminal allocation`. I tried logged in using `-T` and I got the password as well, but I cannot see what I'm typing which looks like typing password in linux.

Bandit20

> 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.

```
bandit19@bandit:~$ ./bandit20-do
Run a command as another user.
  Example: ./bandit20-do id
bandit19@bandit:~$ ./bandit20-do cat /etc/bandit_pass/bandit20
GbKksEFF4yrVs6il55v6gwY5aVje5f0j
```

The password is `GbKksEFF4yrVs6il55v6gwY5aVje5f0j`.

Bandit21

> 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).

After I logged into the bandit20 server, I tested the `setuid` binary file at first. It says **This program will connect to the given port on localhost using TCP. If it receives the correct password from the other side, the next password is transmitted back**. As it indicates, we need a port to listen to this `setuid` binary file. Then I reopen a ssh connection and open a port with `-l` which means listening. Also the `-p 9999` means that the `echo` command will run on localhost:9999. There is no output for that, it's like a service which keeps running until someone connect and say something to this port.

```
Input: echo "GbKksEFF4yrVs6il55v6gwY5aVje5f0j" | nc -l -p 9999
```

Then I went back to former ssh connection and run the binary file with port set to 9999.

```
Input: ./suconnect 9999

Output:
Read: GbKksEFF4yrVs6il55v6gwY5aVje5f0j
Password matches, sending next password
```

It looks like the binary file is sending next password, then I check the other ssh connection which runs port 9999 listening.

```
Output: gE269g2h3mw3pwgrj0Ha9Uoqen1c9DGr
```

```
bandit20@bandit:~$ echo "GbKksEFF4yrVs6il55v6gwY5aVje5f0j" | nc -l -p 9999
gE269g2h3mw3pwgrj0Ha9Uoqen1c9DGr
bandit20@bandit:~$ nmap localhost -p9999

Starting Nmap 7.40 ( https://nmap.org ) at 2019-04-20 21:03 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000092s latency).
PORT     STATE  SERVICE
9999/tcp closed abyss

Nmap done: 1 IP address (1 host up) scanned in 0.05 seconds
```

As we can see from this connection session, the port was closed after receiving the password.

Bandit22

> 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.

As the goal indicates, I looked into the `/etc/cron.d/` folder and list the contents. Because We're in bandit22, let's see what's inside this file `cronjob_bandit22`.

```
Input: cd /etc/cron.d/
Input: ls

Output: cronjob_bandit22 cronjob_bandit23 cronjob_bandit24

Input: cat cronjob_bandit22

Output:
@reboot bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
* * * * * bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
```

It mentions `/usr/bin/cronjob_bandit22.sh` so I'm going to check it out.

```
Input: cat /usr/bin/cronjob_bandit22.sh

Output:
#!/bin/bash
chmod 644 /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
cat /etc/bandit_pass/bandit22 > /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
```

The `/usr/bin/cronjob_bandit22.sh` file save the password of bandit22 into temp folder.

```
Input: cat /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
```

`Output: Yk7owGAcWjwMVRwrTesJEwB7WVOiILLI`

So the password for bandit22 is Yk7owGAcWjwMVRwrTesJEwB7WVOiILLI.

Bandit23

> 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.

Just like `bandit22`, I tried looking into the `/etc/cron.d/` folder and list the contents.

```
Input: cd /etc/cron.d/
Input: ls

Output: cronjob_bandit22 cronjob_bandit23 cronjob_bandit24

Input: cat cronjob_bandit23

Output:
@reboot bandit23 /usr/bin/cronjob_bandit23.sh &> /dev/null
* * * * * bandit23 /usr/bin/cronjob_bandit23.sh &> /dev/null

Input: cat /usr/bin/cronjob_bandit23.sh

Output: #!/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
```

The file looks more complicated than bandit22, so I tried to run it first.

```
Input: /usr/bin/cronjob_bandit23.sh

Output: Copying passwordfile /etc/bandit_pass/bandit22 to /tmp/8169b67bd894ddbb4412f91573b38db3
```

As we can see from the output, it copies the bandit22 password file to `/tmp` folder. So if we want the bandit23's password, we can tweak the `myname` in the script. That's why I tried `echo I am user bandit23 | md5sum | cut -d ' ' -f 1`

```
Output: 8ca319486bfbbc3663ea0fbe81326349
```

As long as bandit23 had run this script, there should be that password file.

```
Input: cat /tmp/8ca319486bfbbc3663ea0fbe81326349
```

Here it is:
`Output: jc1udXuA1tiHqjIsL8yaapX5XIAI6i0n`

Bandit24

> 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.

Like the previous two bandits, we explore the script for bandit24.

```
bandit23@bandit:/etc/cron.d$ 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 $i"
        timeout -s 9 60 ./$i
        rm -f ./$i
    fi
done
```

This level is interesting because we can actually see others creating their shell files. And more surprisingly, the scripts will be deleted once the automatic program runs at regular intervals from cron.

Basically, I copied the command from last level:

```
#!/bin/bash
cat /etc/bandit_pass/bandit24 > /tmp/bandit24sf/bandit24
```

After I paste the command into a shell file, I wait for the system to automatically run that script. However, nothing happened after 5 minutes. Then I tried to run this script under the current level to see what's going on here and it prompts that "permission denied". So I Googled the `chmod` details and find this blog very useful to help me memorize the linux permission commands. Hence, I changed the access code to `chmod 777`.

At first, I only tried to change the permission of the current shell file. But it still doesn't create new file under the temp folder. Then I searched for other's solution. So I noticed that I should also change the current folder to ensure that user `bandit24` can write file here.

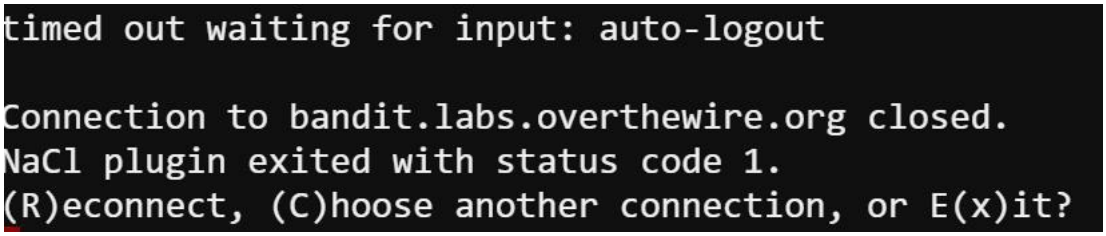
The password is UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ for bandit24.

Bandit25

> 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.

In this challenge, we have to brute-force the ping at most 10000 times like it asked. In order to write a bash file, we have to change directory to `/tmp`. Because I didn't have any experience writing a loop function in a bash file. I searched for "the equivalent python `range` function in bash". This link shows we can do that using `seq`.
``

I waited for the script to brute force. The begin time is 10:54 a.m. The process looks very slow, I recommend an interesting Italian movie `Quo vado? (2016)`. However, I found that the server disconnected when I was cooking for lunch. So I look for solutions to prevent disconnecting, but most of the solutions require permission to `/etc/ssh/sshd_config`. This method offers a good option for using `nohup` or `screen` command, and I re-run the script using `screen`, now the time is 2:20 pm.



```
bandit24@bandit:/tmp$ ./sifan.sh
0000
0001
0002
0003
0004
0005
0006
0007
0008
0009
0010
```

I had no idea what happened to that server, I still didn't get the script running in the background after the ssh disconnected. So I write this python script to simulate the keyboard pressing the "Up" key and the "Enter" key to check the status of brute-forcing.

SimulateKeyPress.py - Notepad

File Edit Format View Help

```
import time
from pynput.keyboard import Key, Controller


keyboard = Controller()

while True:
    time.sleep(20)

    keyboard.press(Key.up)
    keyboard.release(Key.up)
    keyboard.press(Key.enter)
    keyboard.release(Key.enter)
```

```
bandit24@bandit:/tmp$ wc -l result25.txt
9023 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9082 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9136 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9191 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9250 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9305 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9364 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9424 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9484 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9546 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9609 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9672 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9733 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9793 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9855 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
9916 result25.txt
bandit24@bandit:/tmp$ wc -l result25.txt
```

Even though I got the right pincode to access bandit25, I realize that I ignore the pincodes start with 0 (e.g. 0013). When I searched for how to implement that in Linux bash, I learned the usage of `seq` command and I found that we can actually use one line to get this job done.

After I wrote the script as follows:

```
seq -f "UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ %04g" 0000 10000 | nc localhost 30002 >> seq25.txt
```

The screen started flashing, then I realized that the old script was running slow because I only `echo` one line during every connection. But actually, we can input much more lines than that. I thought I could've got the password this time. However, this one-time connection only allows us to try 7178 times. I know it because I run this "count line" command `wc -l seq25.txt | uniq -c`. So I added another try `seq -f "UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ %04g" 7000 10000 | nc localhost 30002 >> seq25.txt` which gives me the right password for bandit25:
`uNG9O58gUE7snukf3bvZ0rxhtnjzSGzG`.

```
Wrong! Please enter the correct pincode. Try again.
Timeout. Exiting.
bandit24@bandit:/tmp$ wc -l seq25.txt | uniq -c
      1 7178 seq25.txt
bandit24@bandit:/tmp$ seq -f "UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ %04g" 7000 10000 | nc localhost 30002 >> seq25.txt
bandit24@bandit:/tmp$ wc -l seq25.txt | uniq -c
      1 10047 seq25.txt
bandit24@bandit:/tmp$ wc -l seq25.txt | uniq -u
10047 seq25.txt
bandit24@bandit:/tmp$ cat seq25.txt | uniq -u
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.
Timeout. Exiting.
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$
```

Bandit26

> 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.

Using `ssh -i bandit26.sshkey bandit26@localhost`, we can easily log in to bandit26 like the goal describes. But we'll be logged out, so we need to check what shell is used in bandit26 instead of /bin/bash.

`Input: cat /etc/passwd`

Output:

```
bandit0:x:11000:11000:bandit level 0:/home/bandit0:/bin/bash
bandit1:x:11001:11001:bandit level 1:/home/bandit1:/bin/bash
bandit10:x:11010:11010:bandit level 10:/home/bandit10:/bin/bash
bandit11:x:11011:11011:bandit level 11:/home/bandit11:/bin/bash
bandit12:x:11012:11012:bandit level 12:/home/bandit12:/bin/bash
bandit13:x:11013:11013:bandit level 13:/home/bandit13:/bin/bash
bandit14:x:11014:11014:bandit level 14:/home/bandit14:/bin/bash
bandit15:x:11015:11015:bandit level 15:/home/bandit15:/bin/bash
bandit16:x:11016:11016:bandit level 16:/home/bandit16:/bin/bash
bandit17:x:11017:11017:bandit level 17:/home/bandit17:/bin/bash
bandit18:x:11018:11018:bandit level 18:/home/bandit18:/bin/bash
bandit19:x:11019:11019:bandit level 19:/home/bandit19:/bin/bash
bandit2:x:11002:11002:bandit level 2:/home/bandit2:/bin/bash
bandit20:x:11020:11020:bandit level 20:/home/bandit20:/bin/bash
bandit21:x:11021:11021:bandit level 21:/home/bandit21:/bin/bash
bandit22:x:11022:11022:bandit level 22:/home/bandit22:/bin/bash
bandit23:x:11023:11023:bandit level 23:/home/bandit23:/bin/bash
bandit24:x:11024:11024:bandit level 24:/home/bandit24:/bin/bash
bandit25:x:11025:11025:bandit level 25:/home/bandit25:/bin/bash
bandit26:x:11026:11026:bandit level 26:/home/bandit26:/usr/bin/showtext
bandit27:x:11027:11027:bandit level 27:/home/bandit27:/bin/bash
bandit28:x:11028:11028:bandit level 28:/home/bandit28:/bin/bash
bandit29:x:11029:11029:bandit level 29:/home/bandit29:/bin/bash
```

Then we check what's inside that file:
`Input: cat /usr/bin/showtext`

`Output:#!/bin/sh`
`more ~/text.txt`
`exit 0`

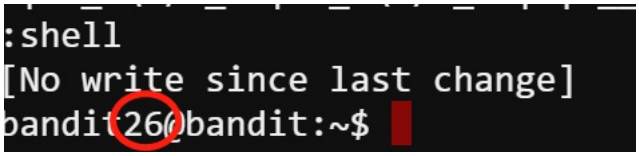
It's worth noting that I also tried the same method as bandit19, but it doesn't work due to the change of shell.

```
bandit25@bandit:~$ ssh -i bandit26.sshkey bandit26@localhost -t cat ~/text.txt
Could not create directory '/home/bandit25/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:98UL0ZWr85496EtCRkKlo20X3OPnyPSB5tB5RPbhczc.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/bandit25/.ssh/known_hosts).
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames


  _                     _ _ _   ___   __
 | |                   | (_) | |__ \ / /
 | |__   __ _ _ __   __| |_| |_   ) / /_
 | '_ \ / _` | '_ \ / _` | | __| / / '_ \
 | |_) | (_| | | | | (_| | | |_ / /| (_) |
 |_.__/ \__,_|_| |_|\__,_|_|\__|____\___/
Connection to localhost closed.
```

Since it uses command `more` which takes effect only when our window size is smaller than the full page. Hence, we resize the window and here it is.

As long as the `more` function is running, that's to say, we're still user bandit26 within the `more` function. Looking at the manual of `more` command, I found that we could read other files in the vim mode by pressing `v` and enter `:e /etc/bandit_pass/bandit26` we get the password `5czgV9L3Xx8JPOyRbXh6lQbmIOWvPT6Z` for bandit 26. However, getting this password does not allow us to normally log in to bandit26. Then I recalled the goal of this level is finding the shell. I find the usage of setting shell inside a vim editor here. Hence, after entering the vim mode, we could set the shell back to `/bin/bash` by running `:set shell=/bin/bash` and run `:shell`. Finally, we enter the shell as bandit26.



Bandit27

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

This must be the easiest level other than bandit0 for that we can literally grab the password for bandit 27. I just list the file in the current folder and then follow the my intuition.

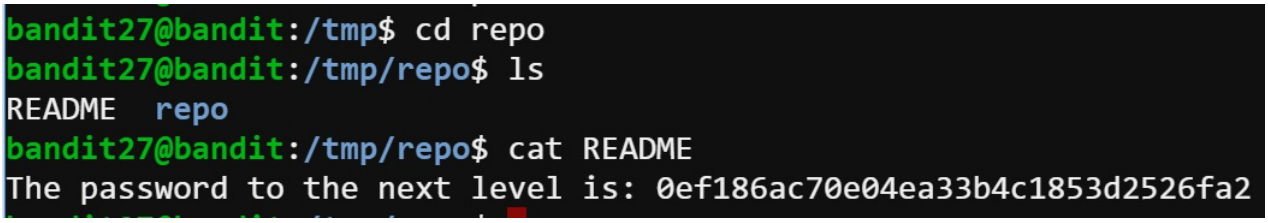
```
bandit26@bandit:~$ ls
bandit27-do  text.txt
bandit26@bandit:~$ ./bandit27-do
Run a command as another user.
  Example: ./bandit27-do id
bandit26@bandit:~$ ./bandit27-do id
uid=11026(bandit26) gid=11026(bandit26) euid=11027(bandit27) groups=11026(bandit26)
bandit26@bandit:~$ ./bandit27-do cat /etc/bandit_pass/bandit27
3ba3118a22e93127a4ed485be72ef5ea
```

The password is `3ba3118a22e93127a4ed485be72ef5ea`.

Bandit28

> 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.

I tried `git clone ssh://bandit27-git@localhost/home/bandit27-git/repo` but I get the "Permission Denied" error, which made me think that unlike https link, `git clone` command works differently in ssh link. But after I read many git documentations, I realized that my fault was attempting to write file in the `~` directory. I should've went to `/tmp` folder in order to write files. So I changed to `/tmp` folder, and I guess that someone may have already cloned the repository. Then I tried this.



It turns out that someone dow nloaded the repository but didn't delete it. That's actually annoying because it w ill cause a fatal error if others try to git clone in the same directory. Besides, lazy people like me w ill pass the `git clone` process. So I deleted this folder after get the passw ord. :-) hahahahaha!

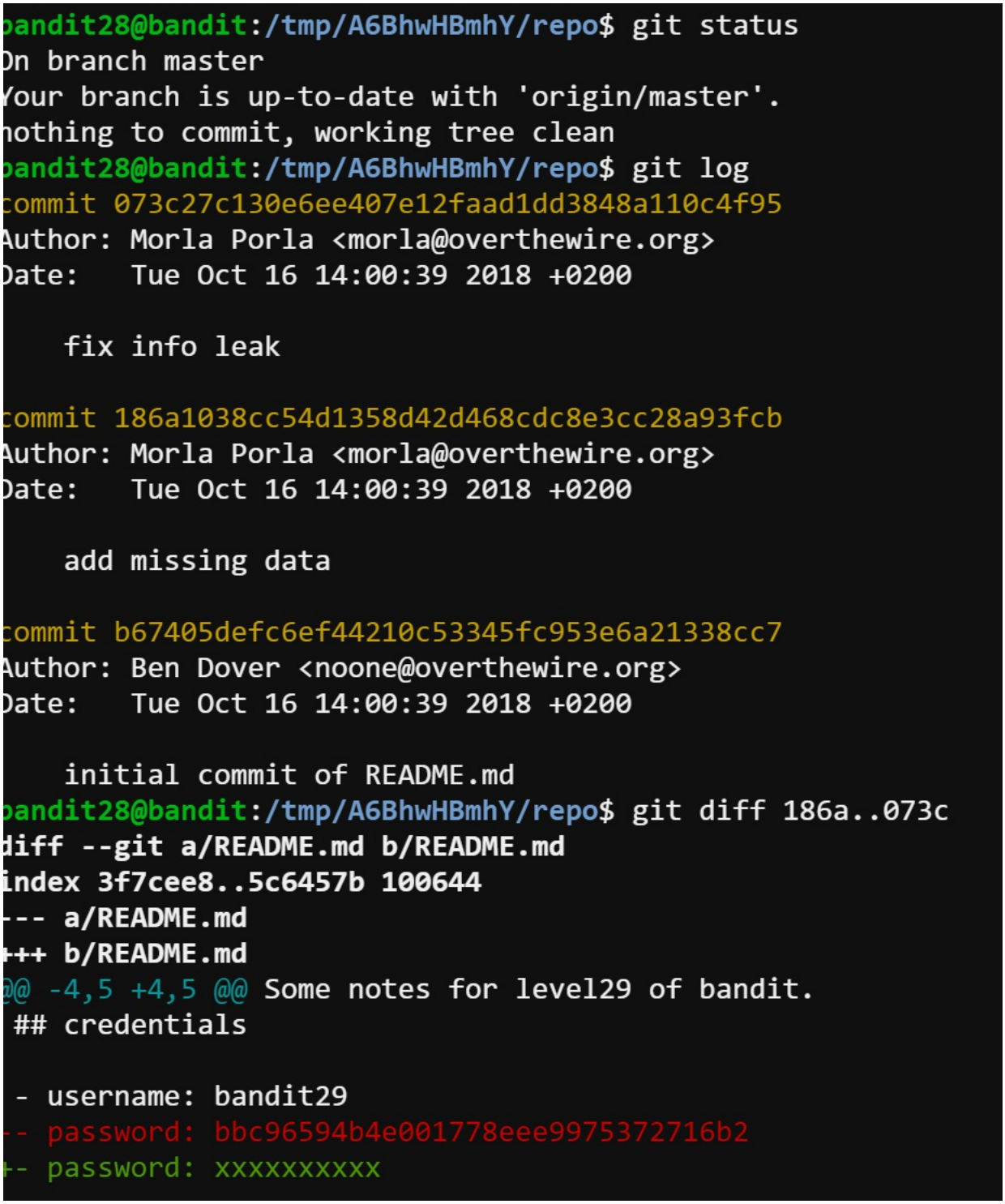
`Input: cat README`

`Output: The password to the next level is: 0ef186ac70e04ea33b4c1853d2526fa2`

Bandit29

> There is a git repository at ssh://bandit28-git@localhost/home/bandit28-git/repo. The passw ord for the user bandit28-git is the same as for the user bandit28.

At first, I really thought the passw ord w as that 10 "x". How ever, the server refused to serve that passw ord. Then I checked the `git log` and noticed that "fix info leak" commit message, so I compare the difference betw een these tw o commits using `git diff` . Eventually, the passw ord `bbc96594b4e001778eee9975372716b2` show s up.



Bandit30

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

This time the README file looks different, but I still tried the former solution and it didn't w ork. But this sentence `password: <no passwords in production!>` looks like a hint w hich might suggest that w e can check other branches.

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

## credentials

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

After `git remote show origin` command shows all the branches, I guess the password could exist in the `dev` branch. Hence, I `git pull` the `dev` branch and check it out.

```
bandit29@bandit:/tmp/ddds/repo$ git remote show origin
Could not create directory '/home/bandit29/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:98UL0ZWr85496EtCRkKlo20X3OPnyPSB5tB5RPbhczc.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/bandit29/.ssh/known_hosts).
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames

bandit29-git@localhost's password:
Permission denied, please try again.
bandit29-git@localhost's password:
Permission denied, please try again.
bandit29-git@localhost's password:
* remote origin
  Fetch URL: ssh://bandit29-git@localhost/home/bandit29-git/repo
  Push  URL: ssh://bandit29-git@localhost/home/bandit29-git/repo
  HEAD branch: master
  Remote branches:
    dev         tracked
    master      tracked
    sploits-dev tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
bandit29@bandit:/tmp/ddds/repo$ git pull origin dev
Could not create directory '/home/bandit29/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:98UL0ZWr85496EtCRkKlo20X3OPnyPSB5tB5RPbhczc.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/bandit29/.ssh/known_hosts).
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames

bandit29-git@localhost's password:
From ssh://localhost/home/bandit29-git/repo
 * branch            dev        -> FETCH_HEAD
Updating 84abedc..33ce2e9
Fast-forward
 README.md         | 2 +-
 code/gif2ascii.py | 1 +
 2 files changed, 2 insertions(+), 1 deletion(-)
```

Finally, we found the password in the `dev` branch. The password is `5b90576bedb2cc04c86a9e924ce42faf`.

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

## credentials

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

Bandit31

> 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.

This time, I tried the same process as last one, but they didn't bring me the password. Since the password must be inside this git repository and there is not so much space to hide. I tried `git tag` to show the tags and one tag called `secret` came up. Then I use `git show` to display the content. I'm not sure this is the password, but why not check it out?

`Input: git show secret`

`Output: 47e603bb428404d265f59c42920d81e5`

```
bandit30@bandit:/tmp/jjjd/repo/.git/refs/tags$ git tag
secret
bandit30@bandit:/tmp/jjjd/repo/.git/refs/tags$ git show secret
47e603bb428404d265f59c42920d81e5
```

That password works!

Bandit32

> 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.

```
bandit31@bandit:/tmp/31ban$ git clone ssh://bandit31-git@localhost/home/bandit31-git/repo
Cloning into 'repo'...
Could not create directory '/home/bandit31/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:98UL0ZWr85496EtCRkKlo20X3OPnyPSB5tB5RPbhczc.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/bandit31/.ssh/known_hosts).
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames

bandit31-git@localhost's password:
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (4/4), done.
bandit31@bandit:/tmp/31ban$ ls
repo
bandit31@bandit:/tmp/31ban$ cd repo
bandit31@bandit:/tmp/31ban/repo$ ls
README.md
bandit31@bandit:/tmp/31ban/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
```

After I `git clone` the repository, I find this time it's actually easier than the previous level. All it asked is to push a file to the remote repository. So I created a new file named `key.txt` with one line `May I come in?` and then `git add` with the parameter `-f`. After that, I commit the changes and push the stage to the remote by `git commit` and `git push`.

```
bandit31@bandit:/tmp/31ban/repo$ vim key.txt
bandit31@bandit:/tmp/31ban/repo$ cat key.txt
May I come in?
bandit31@bandit:/tmp/31ban/repo$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
bandit31@bandit:/tmp/31ban/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.
bandit31@bandit:/tmp/31ban/repo$ git add key.txt =f
fatal: pathspec '=f' did not match any files
bandit31@bandit:/tmp/31ban/repo$ git add key.txt -f
bandit31@bandit:/tmp/31ban/repo$ git commit -m "may I come in please?"
[master f196c5a] may I come in please?
 1 file changed, 1 insertion(+)
 create mode 100644 key.txt
bandit31@bandit:/tmp/31ban/repo$ git push
Could not create directory '/home/bandit31/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:98UL0ZWr85496EtCRkKlo20X3OPnyPSB5tB5RPbhczc.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/bandit31/.ssh/known_hosts).
This is a OverTheWire game server. More information on http://www.overthewire.org/wargames

bandit31-git@localhost's password:
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 331 bytes | 0 bytes/s, done.
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:
To ssh://localhost/home/bandit31-git/repo
 ! [remote rejected] master -> master (pre-receive hook declined)
error: failed to push some refs to 'ssh://bandit31-git@localhost/home/bandit31-git/repo'
```

As the remote prompts, `56a9bf19c63d650ce78e6ec0354ee45e` is the password.

bandit33

> After all this git stuff its time for another escape. Good luck!

In the beginning, I thought there was something wrong with my client, because everything I typed became upper case like this.

```
WELCOME TO THE UPPERCASE SHELL
>> ls
sh: 1: LS: not found
>> pwd
sh: 1: PWD: not found
>>
```

By doing some research (link), I found the `$0` positional parameter command enable me to run commands as the sub-process of that upper case shell. That means we can get the `cat` command run inside this.

```
WELCOME TO THE UPPERCASE SHELL
>> ls
sh: 1: LS: not found
>> $0
$ cat /etc/bandit_pass/bandit33
c9c3199ddf4121b10cf581a98d51caee
$
```

So the password for bandit33 is `c9c3199ddf4121b10cf581a98d51caee`.

bandit34

> At this moment, level 34 does not exist yet.

```
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 worki
ng
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!
```

Oh, yeah, all done! :-)