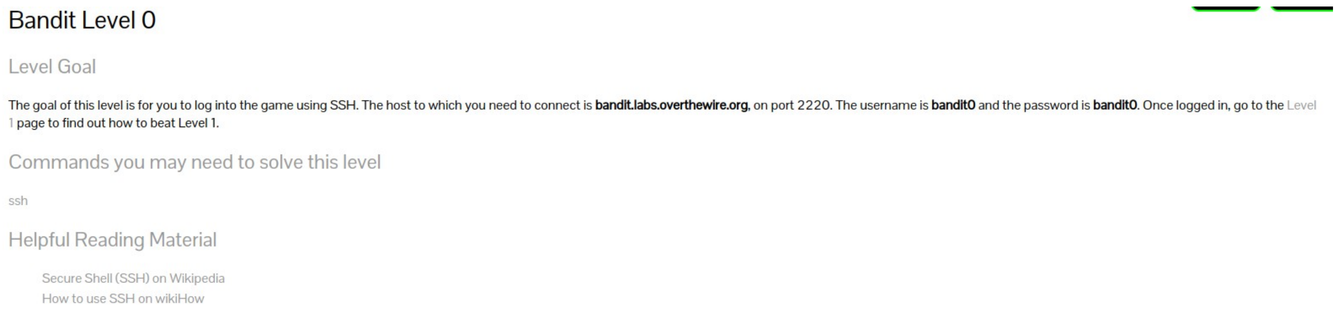
Over the Wire: Bandit

Level 0


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.

Commands you may need to solve this level

ssh

Helpful Reading Material

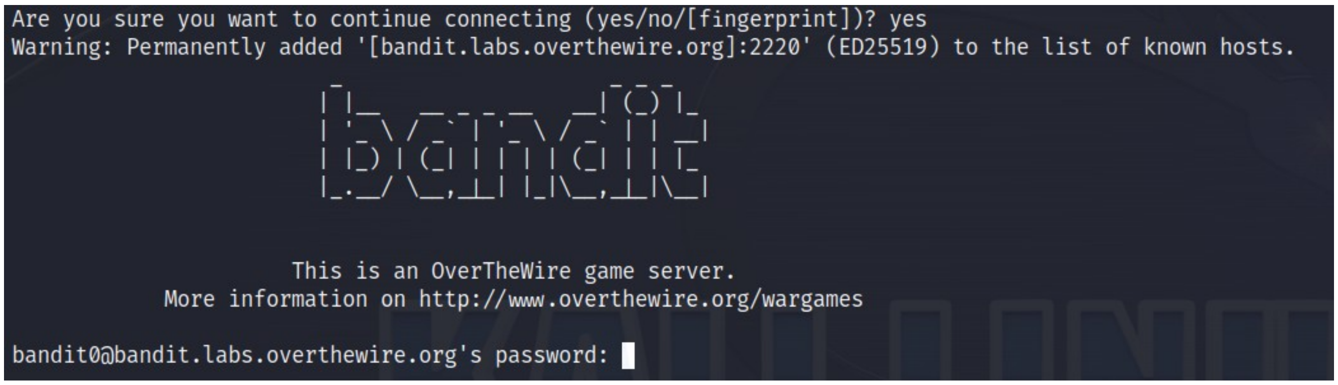
Secure Shell (SSH) on Wikipedia
How to use SSH on wikiHow

Looks like we need to SSH into the first level of the game, seems straight forward. We are provided a host to SSH into along with a port, username and password. If you are new to using SSH you can find more information at https://man7.org/linux/man-pages/man1/ssh.1.html.

Now lets use SSH to login to the first level. Referring to the man page for SSH we see the format is ssh user@host and using the “p” flag we can designate a port. Lets use this new found information to log into the next level.

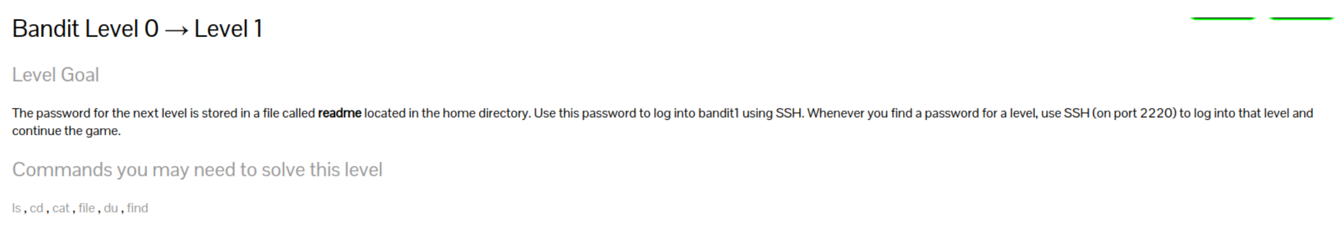
ssh bandit0@bandit.labs.overthewire.org -p 2220

When asked if you want to add “bandit.labs.overthewire.org” to your list of known hosts simply type “yes” and then enter the password you were provided.


```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[bandit.labs.overthewire.org]:2220' (ED25519) to the list of known hosts.
                         _                     _ _ _
                        | |__   __ _ _ __   __| (_) |_
                        | '_ \ / _` | '_ \ / _` | | __|
                        | |_) | (_| | | | | (_| | | |_
                        |_.__/ \__,_|_| |_|\__,_|_|\__|


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

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

Success! We are in and ready to proceed to level 1.

Level 1:



Bandit Level 0 → Level 1

Level Goal

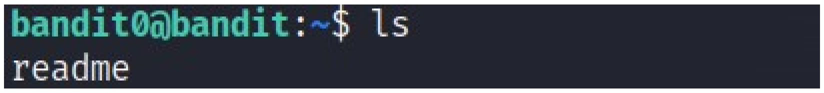
The password for the next level is stored in a file called **readme** located in the home directory. Use this password to log into bandit1 using SSH. Whenever you find a password for a level, use SSH (on port 2220) to log into that level and continue the game.

Commands you may need to solve this level

ls , cd , cat , file , du , find

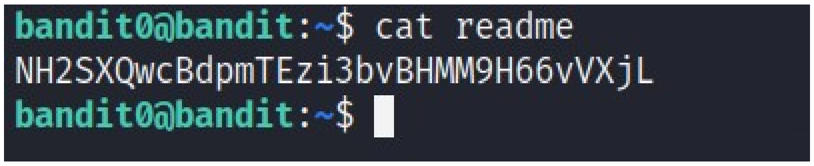
If you are not familiar with Linux I recommend you read the man pages for the listed “Command you may need to solve this level” section.

Lets list out what is in our current directory as the instructions state there should be a “readme” file located in our current working directory. We can do this using the “ls” command.



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

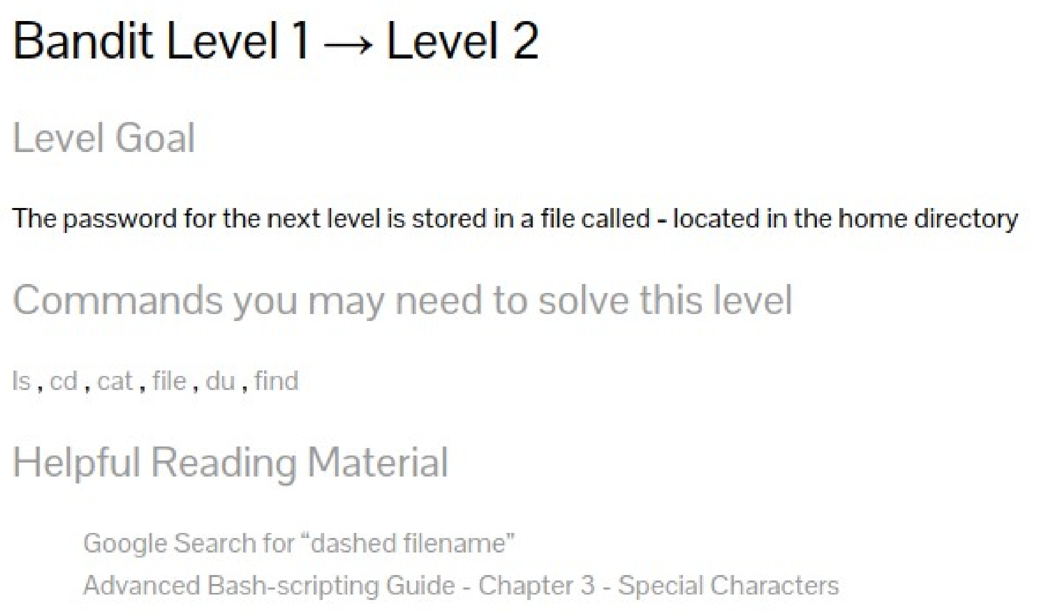
We found the “readme” file! Okay, lets get the flag. We can obtain the flag to the next level by simply reading the contents of the file using the “cat” command.



```
bandit0@bandit:~$ cat readme
NH2SXQwcBdpmTEzi3bvBHMM9H66vVXjL
bandit0@bandit:~$ 
```

Success! We now have the flag for logging into the next level.

level 2:



Bandit Level 1 → Level 2

Level Goal

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

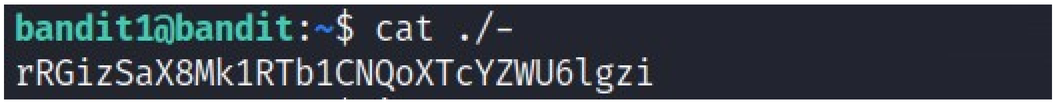
Commands you may need to solve this level

ls , cd , cat , file , du , find

Helpful Reading Material

- Google Search for "dashed filename"
- Advanced Bash-scripting Guide - Chapter 3 - Special Characters

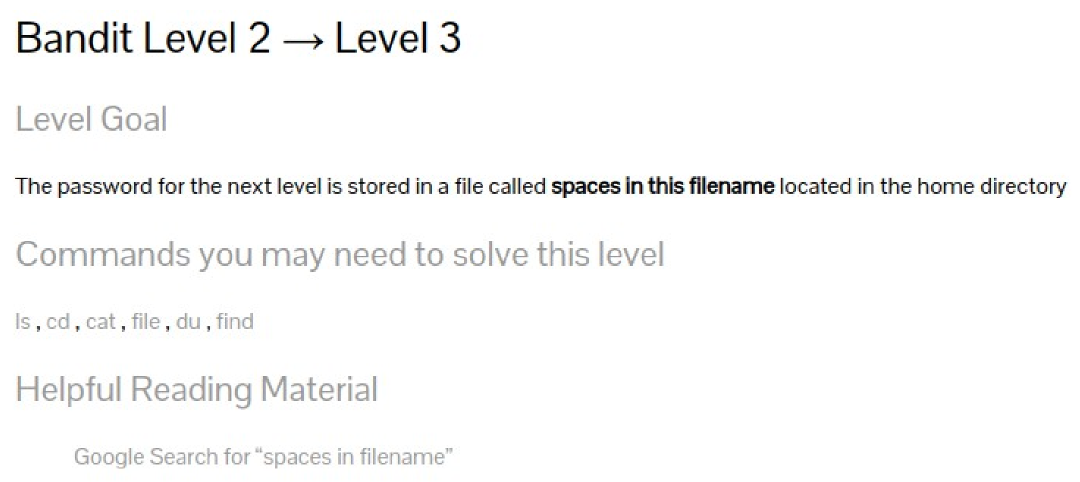
in order to see the contents of this file using "cat" we first need to tell "cat" that this is a file and we are not trying to use a flag. We can accomplish this by pre-pending ./ to the file name.



```
bandit1@bandit:~$ cat ./-
rRGizSaX8Mk1RTb1CNQoXTcYZWU6lgzi
```

success! Once again we can exit the connection and log into the next level using our new found flag.

Level 3:


Bandit Level 2 → Level 3

Level Goal

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

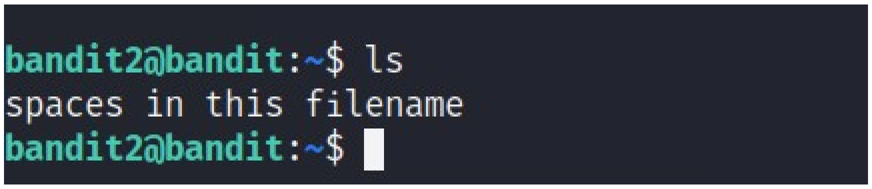
Commands you may need to solve this level

ls , cd , cat , file , du , find

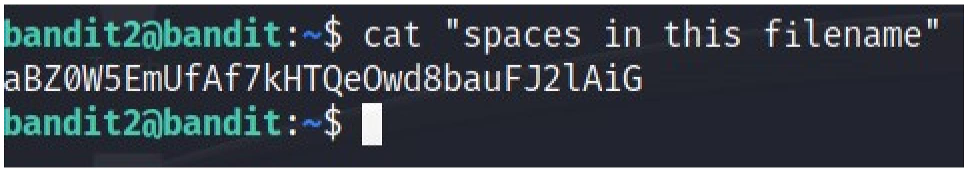
Helpful Reading Material

Google Search for "spaces in filename"

The instructions state that we have a file in our home directory. Lets use the "ls" command to list out what files we have.

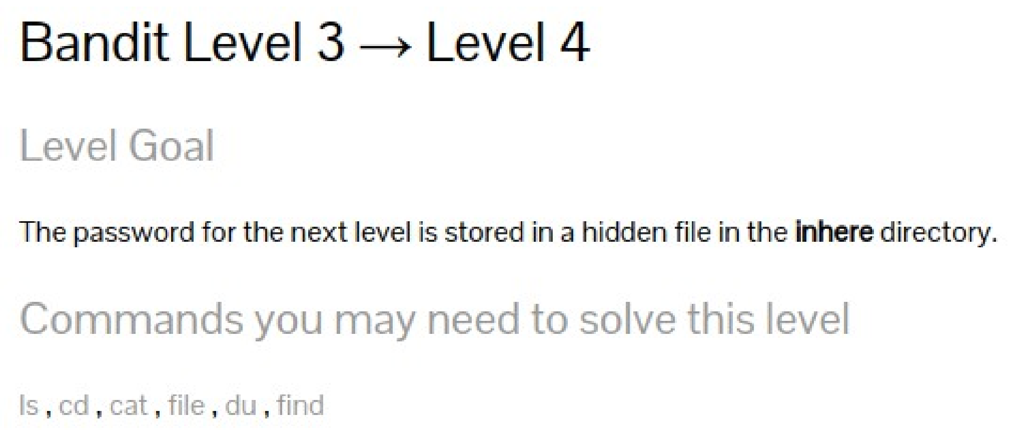

```
bandit2@bandit:~$ ls
spaces in this filename
bandit2@bandit:~$
```

in order to cat out this file we need to wrap the file name in double quotes as white space between the words will not be our friend here.


```
bandit2@bandit:~$ cat "spaces in this filename"
aBZ0W5EmUfAf7kHTQeOwd8bauFJ2lAiG
bandit2@bandit:~$
```

Success! We found our flag.

Level 3


Bandit Level 3 → Level 4

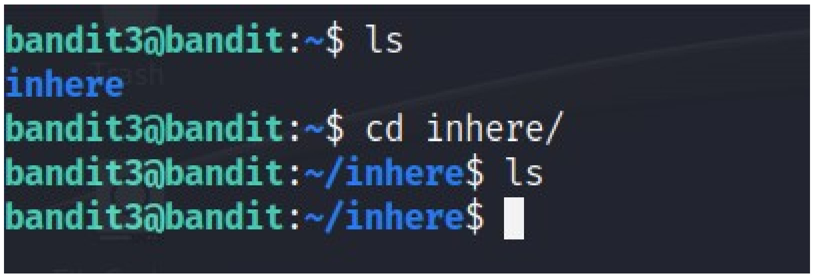
Level Goal

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

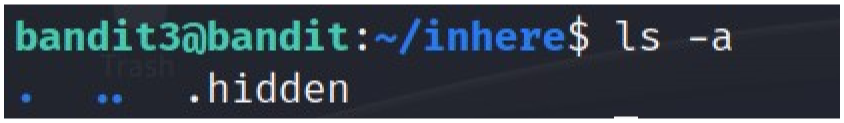
Commands you may need to solve this level

ls , cd , cat , file , du , find

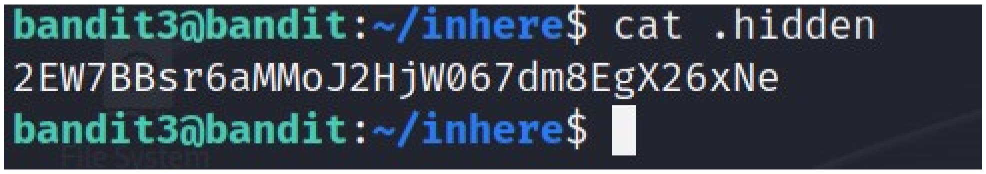
If you list out the contents of your current directory you will see there is a directory named “inhere”. Lets cd into the “inhere” directory. After changing directories you will notice if we use the ls command there is nothing listed.


```
bandit3@bandit:~$ ls
inhere
bandit3@bandit:~$ cd inhere/
bandit3@bandit:~/inhere$ ls
bandit3@bandit:~/inhere$ 
```

When we refer to the instructions that were provided we can see that it states this is a “hidden” directory. Lets use the “ls” command again, but this time append a flag to list all contents which will include hidden content. We can accomplish this using the -a flag. According to the man page for “ls” the -a flag will provide the ability to list all contents and will not ignore entries starting with a “.”.

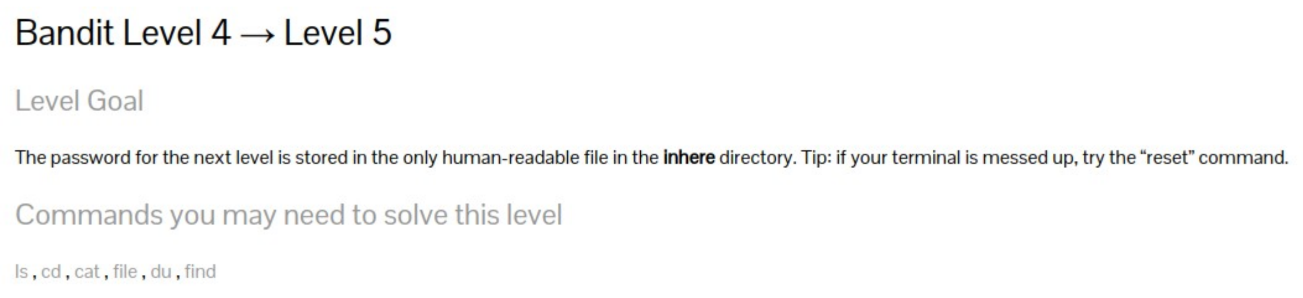

```
bandit3@bandit:~/inhere$ ls -a
.   ..   .hidden
```

We found the “hidden” file! Lets grab the flag now by using the “cat” command.


```
bandit3@bandit:~/inhere$ cat .hidden
2EW7BBsr6aMMoJ2HjW067dm8EgX26xNe
bandit3@bandit:~/inhere$ 
```

Success! We have obtained the flag for the next level! Lets move on.

level 4:



Bandit Level 4 → Level 5

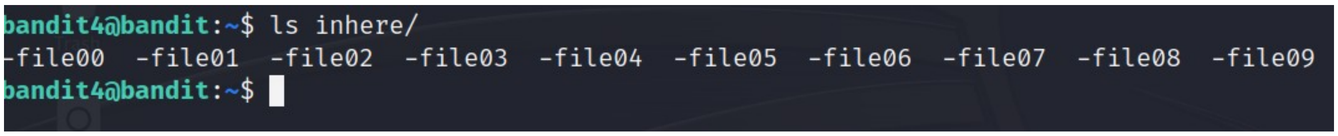
Level Goal

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.

Commands you may need to solve this level

ls , cd , cat , file , du , find

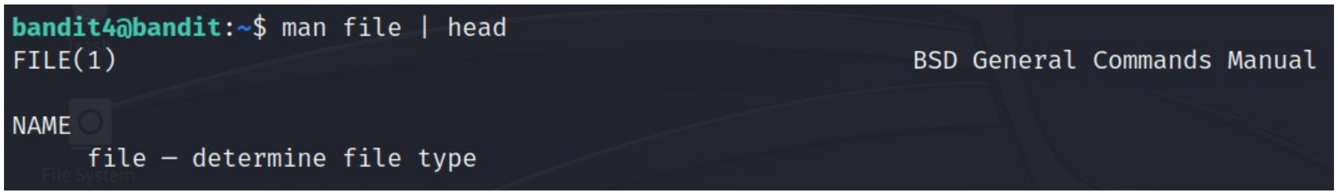
According to the instructions we are looking for the only “human-readable” file inside of the “inhere” directory. Lets go ahead and list out the contents of the “inhere” directory and see how many files we are looking at.



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

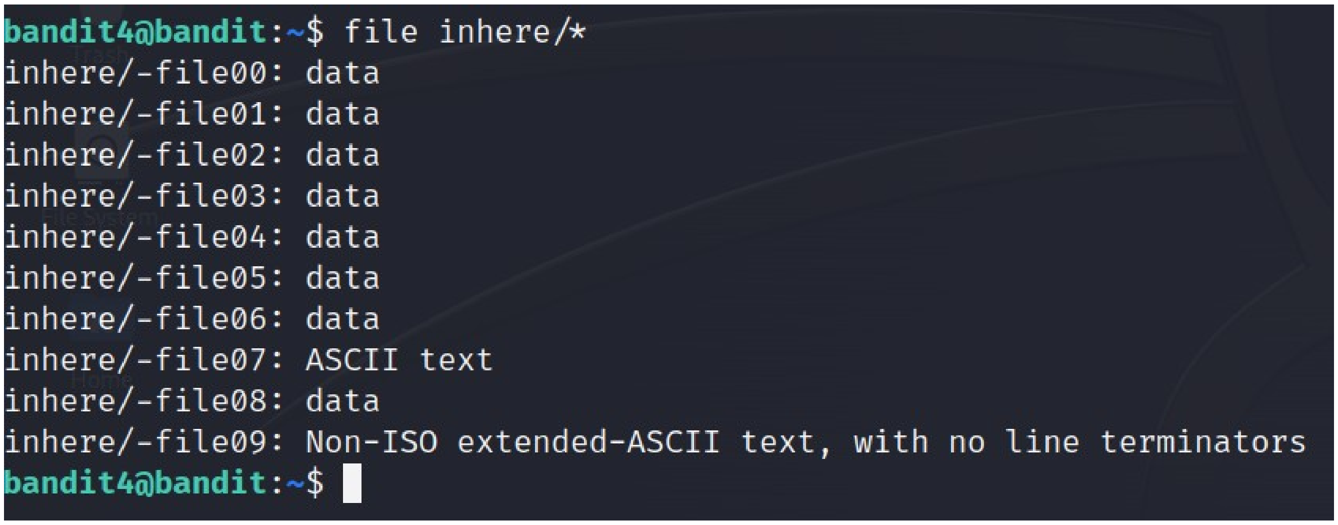
So, we have 10 files we are working with. We have several ways of short cutting this.. however, lets stick to the lesson goals for now. In the instructions we can see the listed commands that could be useful here.

One of the commands listed is the “file” command. If we take a look at the man page for the “file” command we will see that it says it determines the file type. That sounds just like what we need here.

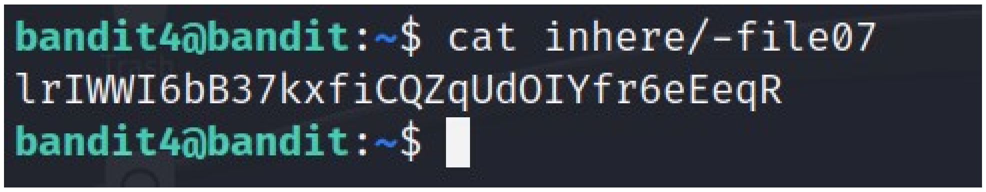


```
bandit4@bandit:~$ man file | head
FILE(1)                                          BSD General Commands Manual

NAME
     file — determine file type
```

Since this sounds like what we need lets run it on all ten files in the “inhere” directory and see what we get.

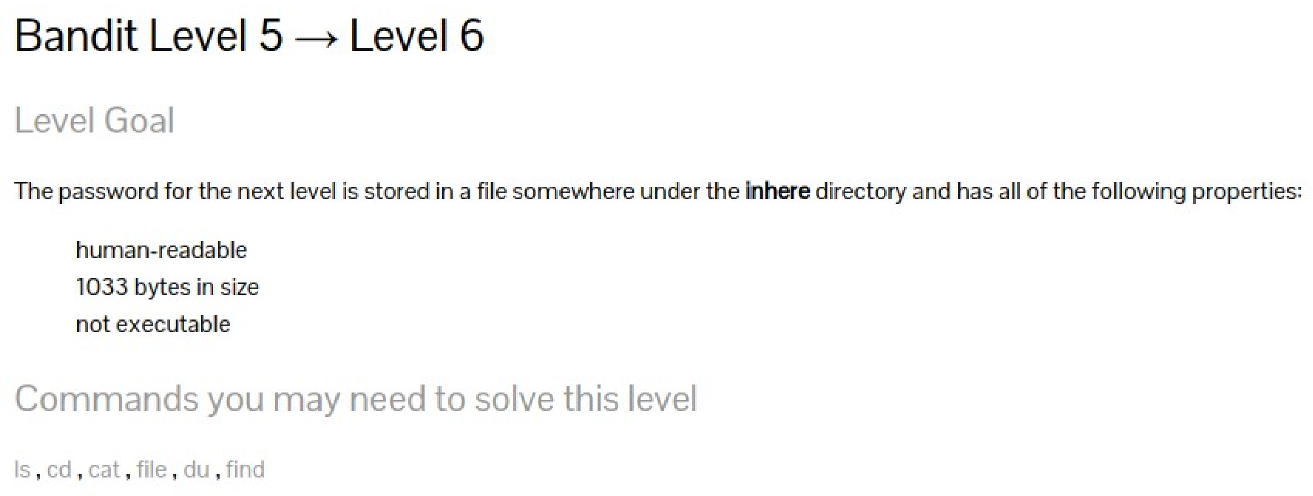

```
bandit4@bandit:~$ file inhere/*
inhere/-file00: data
inhere/-file01: data
inhere/-file02: data
inhere/-file03: data
inhere/-file04: data
inhere/-file05: data
inhere/-file06: data
inhere/-file07: ASCII text
inhere/-file08: data
inhere/-file09: Non-ISO extended-ASCII text, with no line terminators
bandit4@bandit:~$ 
```

Perfect! Looks like the only truly “human-readable” file is “file07”. Looks like it is time to grab our flag. Lets cat out the contents of the file.


```
bandit4@bandit:~$ cat inhere/-file07
lrIWWI6bB37kxfiCQZqUdOIYfr6eEeqR
bandit4@bandit:~$ 
```

success!

level 5:



Bandit Level 5 → Level 6

Level Goal

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

human-readable
1033 bytes in size
not executable

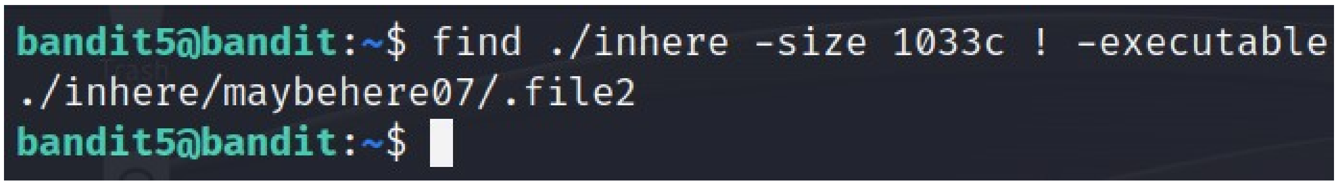
Commands you may need to solve this level

ls , cd , cat , file , du , find

Lets start with working with the tools we have here. In our list of commands that we way need we can see a command by the name of find. By reviewing the man page for the "find" command we can break down the needs for this case.

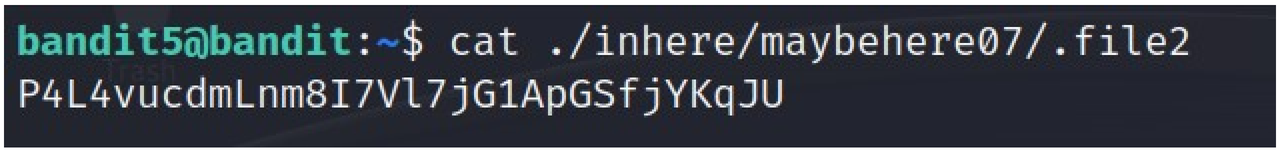
Find ./inhere -size 1033c ! -executable

using the -size flag we can append a 'c' to our size to specify we want bytes and prepend a '!' to our -executable flag to state we want non executable. This seems to be a good first run at our problem. Lets run this and see what we get for results.



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

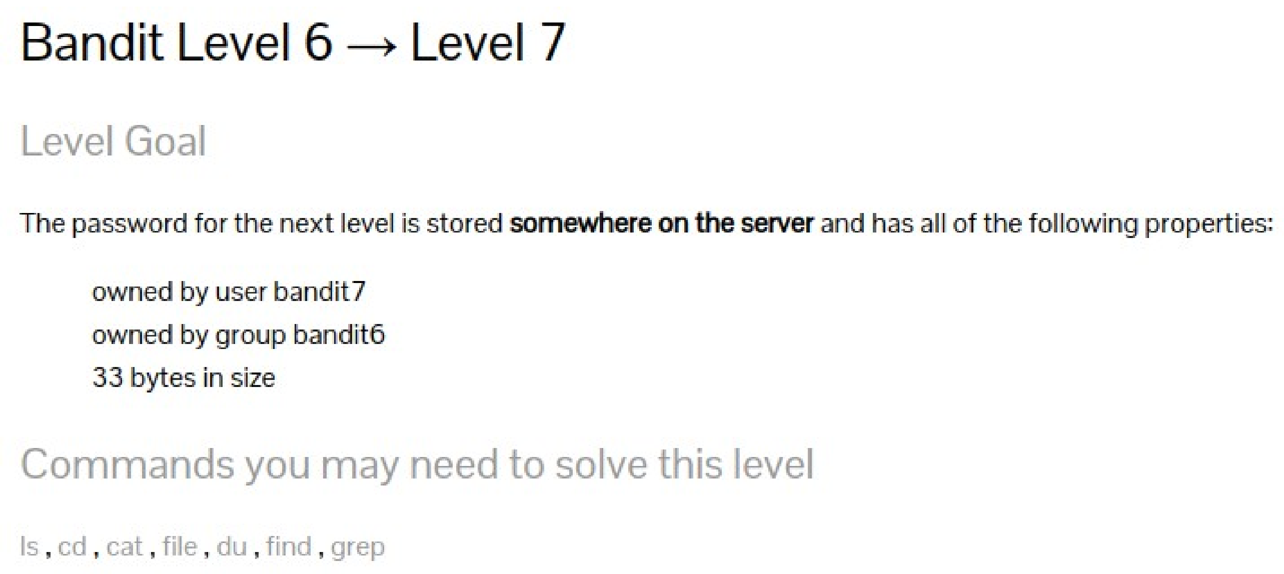
looks promising, lets cat the contents of the "./inhere/maybehere07/.file2" out.



```
bandit5@bandit:~$ cat ./inhere/maybehere07/.file2
P4L4vucdmLnm8I7Vl7jG1ApGSfjYKqJU
```

Success! Another one down, lets proceed to succeed.

level 6


Bandit Level 6 → Level 7

Level Goal

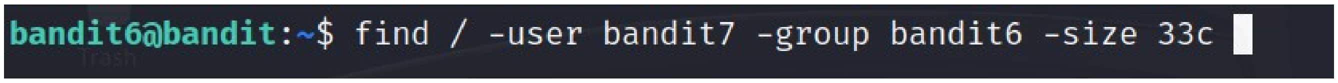
The password for the next level is stored **somewhere on the server** and has all of the following properties:

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

Commands you may need to solve this level

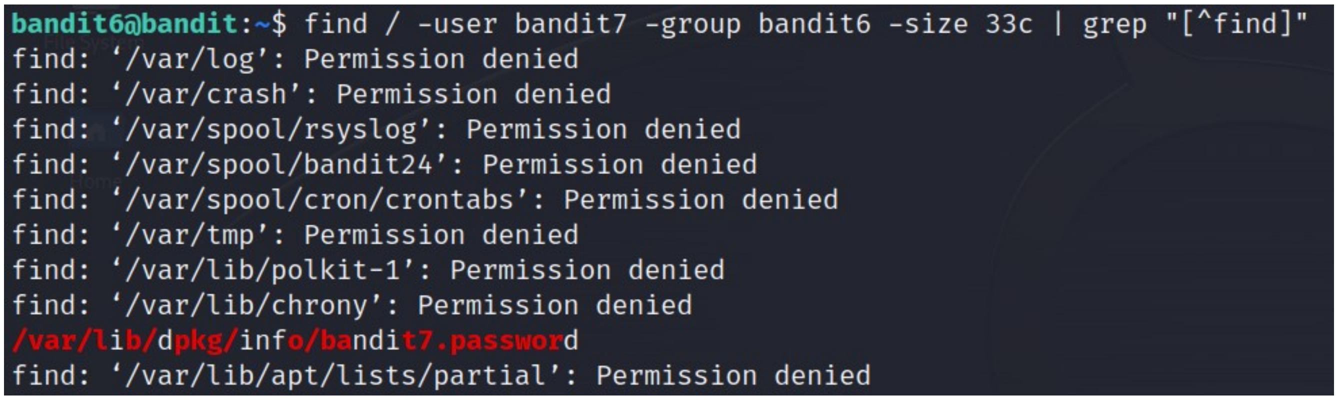
ls , cd , cat , file , du , find , grep

This challenge is very similar to the previous, but this time we do not get a location of the file. Using the “find” command we can search from the root directory up, so no problems there. We can also take a look at the man page once again to find (no pun intended) the flags for user and group.

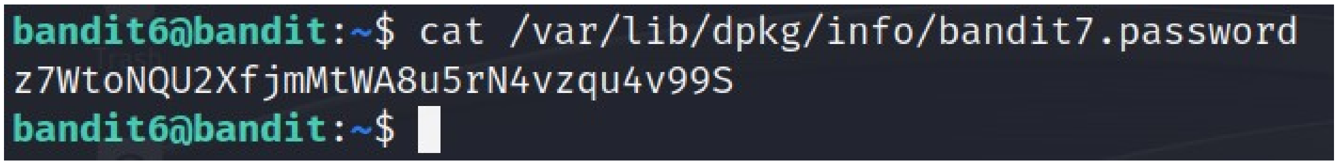

```
bandit6@bandit:~$ find / -user bandit7 -group bandit6 -size 33c
```

When we run this command with the group, size and user flags set we will get a lot of “permission denied” statements. If we look at the results close enough we can see that there is only one location listed that we have access to.

To make things easier on our eyes we can pipe the output into grep and specify lines that start with “find” and we will get a nice red color to look for.


```
bandit6@bandit:~$ find / -user bandit7 -group bandit6 -size 33c | grep "[^find]"
find: ‘/var/log’: Permission denied
find: ‘/var/crash’: Permission denied
find: ‘/var/spool/rsyslog’: Permission denied
find: ‘/var/spool/bandit24’: Permission denied
find: ‘/var/spool/cron/crontabs’: Permission denied
find: ‘/var/tmp’: Permission denied
find: ‘/var/lib/polkit-1’: Permission denied
find: ‘/var/lib/chrony’: Permission denied
/var/lib/dpkg/info/bandit7.password
find: ‘/var/lib/apt/lists/partial’: Permission denied
```

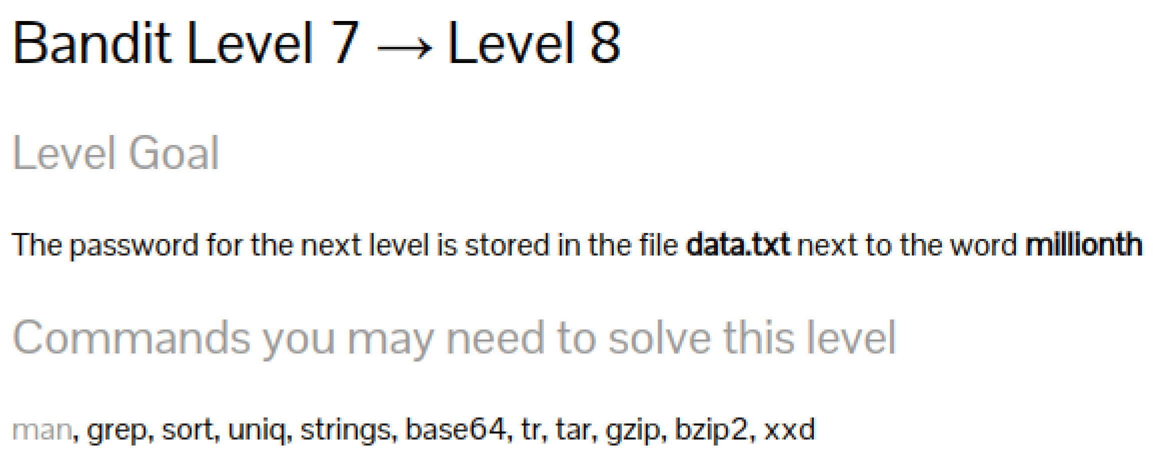
Lets go ahead and cat out the contents of the file.



```
bandit6@bandit:~$ cat /var/lib/dpkg/info/bandit7.password
z7WtoNQU2XfjmMtWA8u5rN4vzqu4v99S
bandit6@bandit:~$ 
```

Success! Lets proceed, shall we?

level 7:


Bandit Level 7 → Level 8

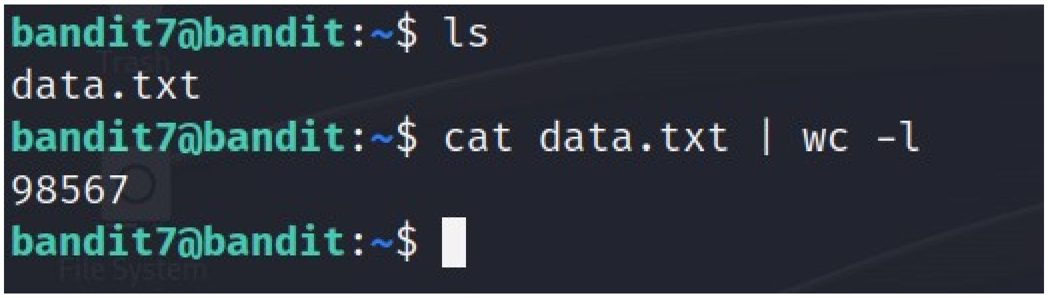
Level Goal

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

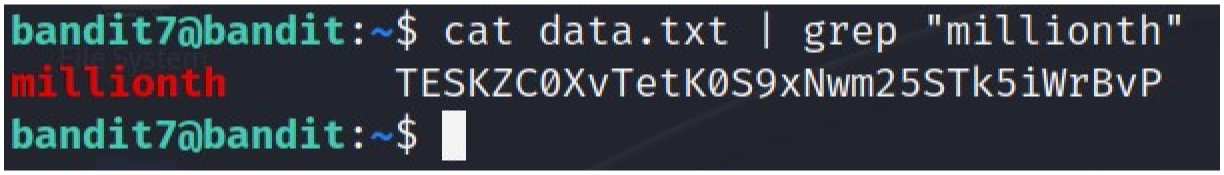
Commands you may need to solve this level

man, grep, sort, uniq, strings, base64, tr, tar, gzip, bzip2, xxd

looking at the file using the “wc” command with the -l flag (for lines) we can see it is pretty large.

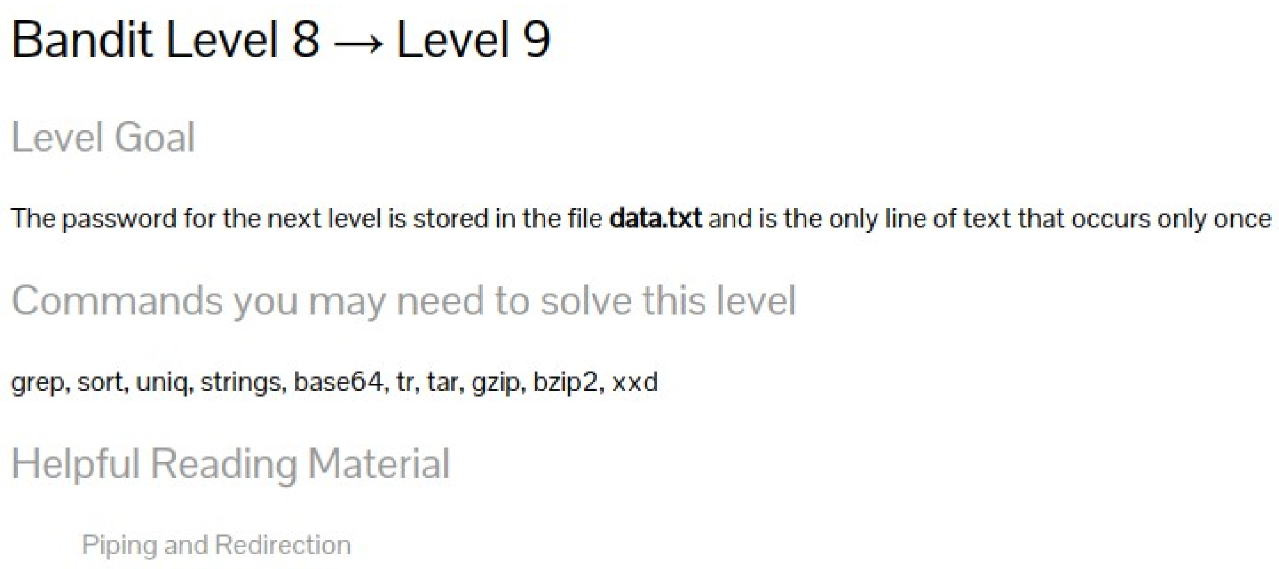

```
bandit7@bandit:~$ ls
data.txt
bandit7@bandit:~$ cat data.txt | wc -l
98567
bandit7@bandit:~$
```

98,567 lines in the file. That is a little to large to eyeball. Lets cat out the contents of the file and pipe the contents to grep looking for the word “millionth” as per the instructions.


```
bandit7@bandit:~$ cat data.txt | grep "millionth"
millionth       TESKZC0XvTetK0S9xNwm25STk5iWrBvP
bandit7@bandit:~$
```

Success!

level 8:



Bandit Level 8 → Level 9

Level Goal

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

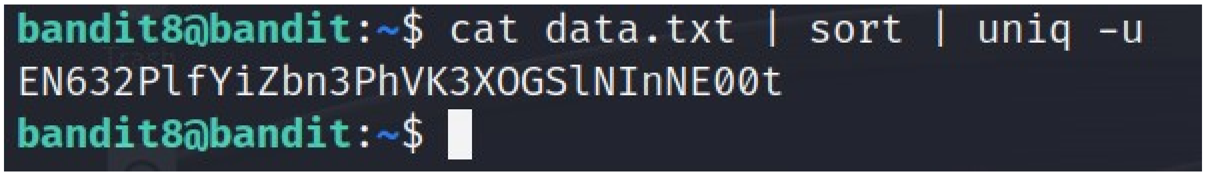
Commands you may need to solve this level

grep, sort, uniq, strings, base64, tr, tar, gzip, bzip2, xxd

Helpful Reading Material

Piping and Redirection

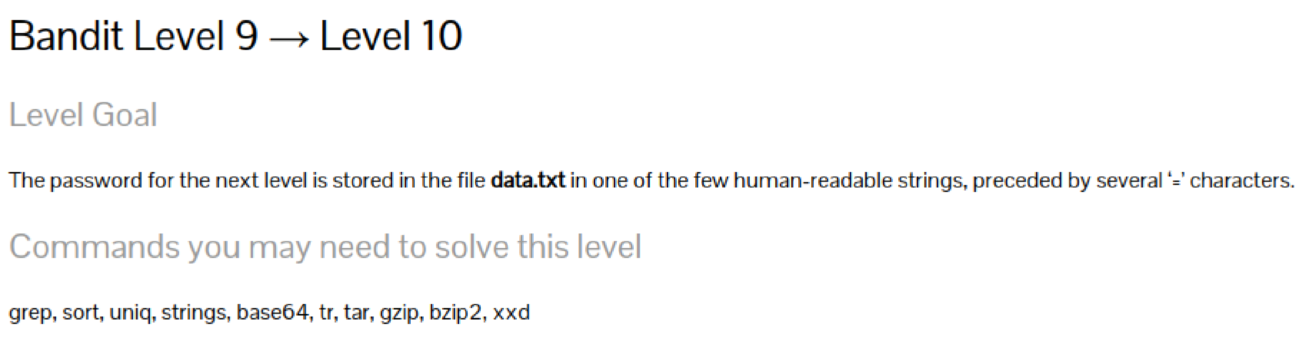
As we review the instructions provided we can see the key is “only line of text that occurs only once”. With this information we are pointed to using “sort” and “uniq”. Lets get to work. We will cat out the content of “data.txt” into “sort” and “uniq”, as we want to sort the data before running it through “uniq”



```
bandit8@bandit:~$ cat data.txt | sort | uniq -u
EN632PlfYiZbn3PhVK3XOGSlNInNE00t
bandit8@bandit:~$ 
```

Success!

level9:



Bandit Level 9 → Level 10

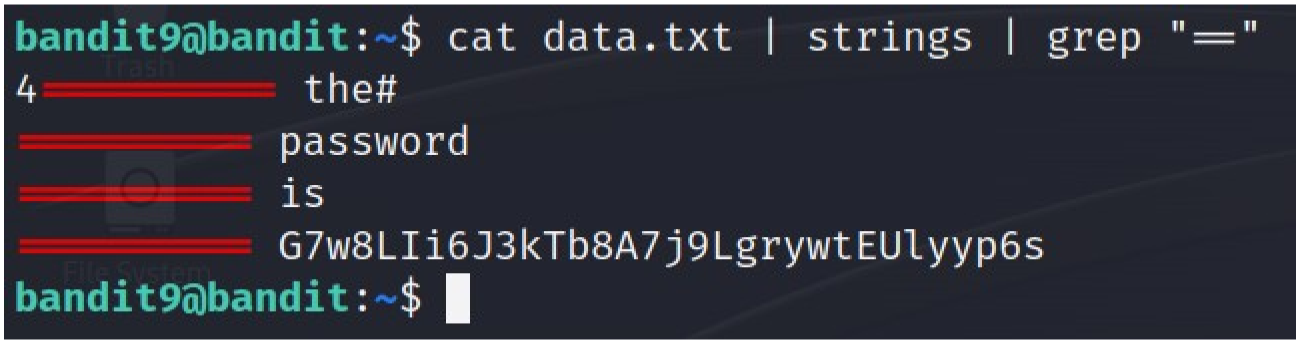
Level Goal

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

Commands you may need to solve this level

grep, sort, uniq, strings, base64, tr, tar, gzip, bzip2, xxd

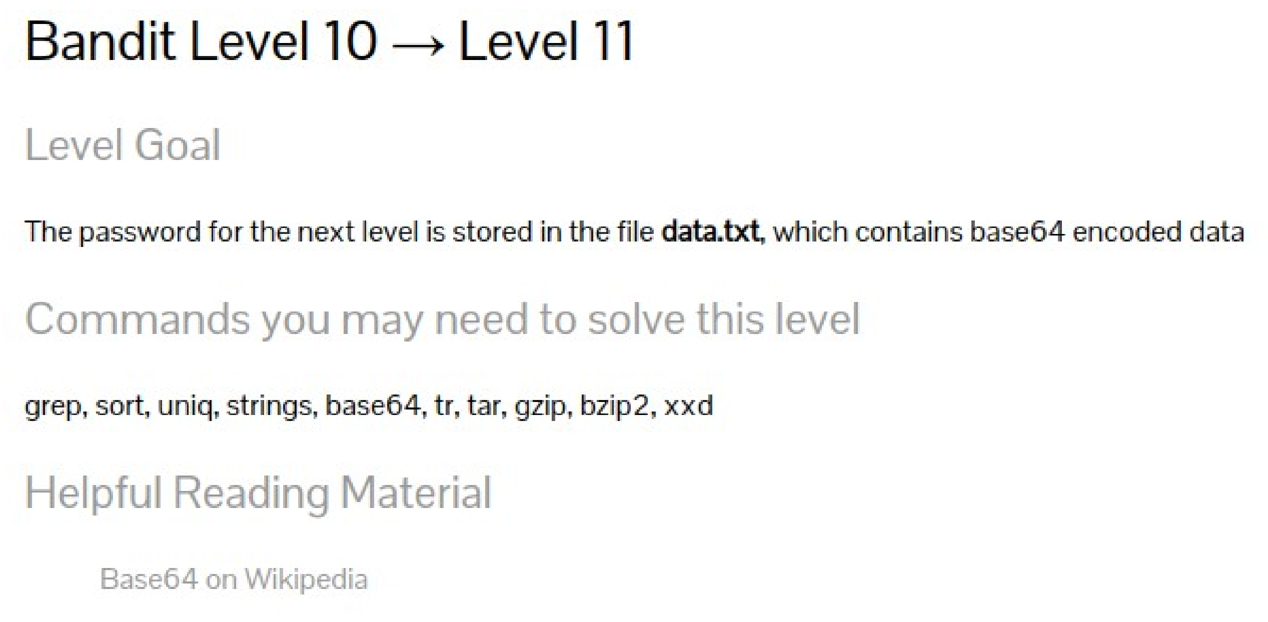
The key in our instructions looks like the wording of "strings", as we can see in our recommend commands we have a command called "strings". To obtain our flag we will cat out the content of "data.txt" and pipe that into "strings" and just to clean it up and save our eyes we will pipe that output into grep looking for "several = characters", as per the instructions.



```
bandit9@bandit:~$ cat data.txt | strings | grep "=="
4========== the#
========== password
========== is
========== G7w8LIi6J3kTb8A7j9LgrywtEUlyyp6s
bandit9@bandit:~$ 
```

Success!

level 10:



Bandit Level 10 → Level 11

Level Goal

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

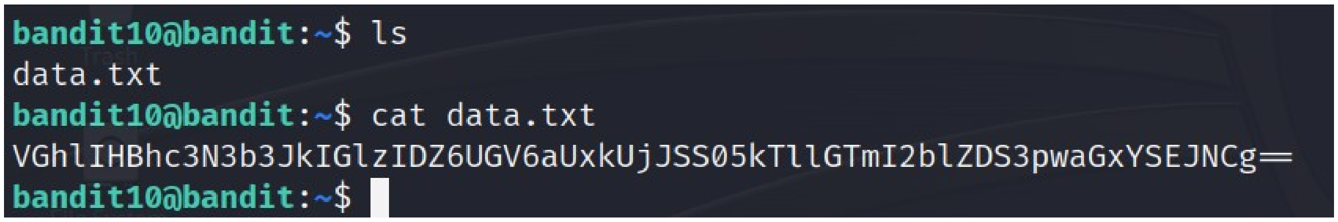
Commands you may need to solve this level

grep, sort, uniq, strings, base64, tr, tar, gzip, bzip2, xxd

Helpful Reading Material

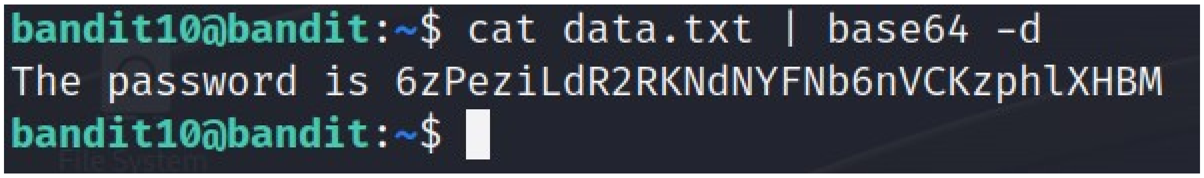
Base64 on Wikipedia

Per the instructions we have base64 encoded data in the data.txt. If we cat out the contents we can see it is worthless to us in its current state.



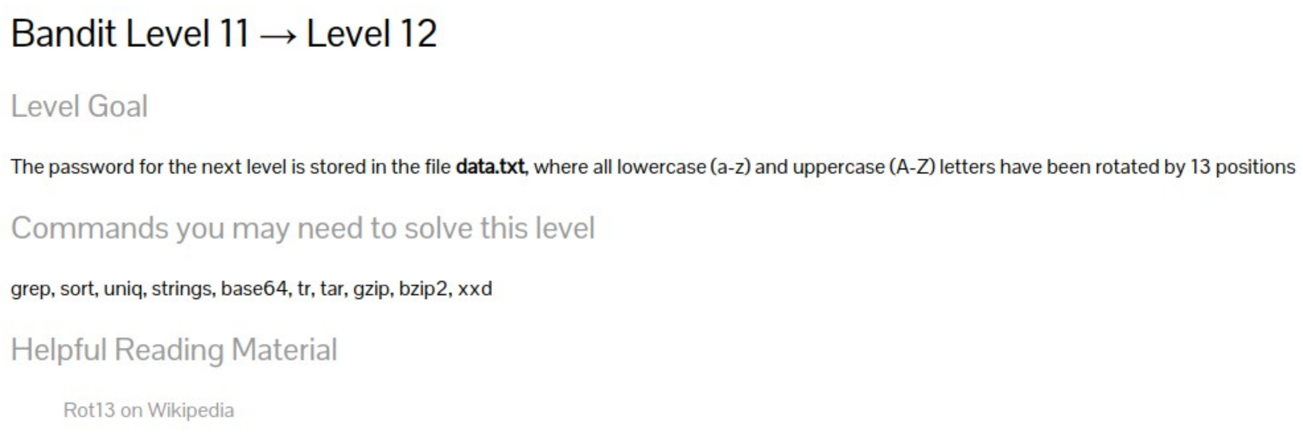
```
bandit10@bandit:~$ ls
data.txt
bandit10@bandit:~$ cat data.txt
VGhlIHBhc3N3b3JkIGlzIDZ6UGV6aUxkUjJSS05kTllGTmI2blZDS3pwaGxYSEJNCg==
bandit10@bandit:~$
```

We have a solution though, we will just pipe this output to base64 with the decode flag and we should be solid.



```
bandit10@bandit:~$ cat data.txt | base64 -d
The password is 6zPeziLdR2RKNdNYFNb6nVCKzphlXHBM
bandit10@bandit:~$
```

level 11:



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

Commands you may need to solve this level

grep, sort, uniq, strings, base64, tr, tar, gzip, bzip2, xxd

Helpful Reading Material

Rot13 on Wikipedia

We can get a better understanding of whats going on here by the "Helpful Reading Material", but the "rotated by 13 positions" points us to ROT-13/ROT13.

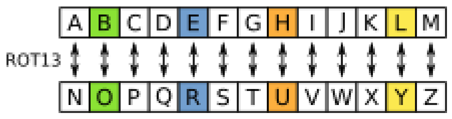
Rot13 acourding to the Wiki page "is a simple letter substitution cipher that replaces a letter with the 13th letter after it". Simple enough.. it is the same action to encode or decode ROT13.

The recommended commands section mentions the command "tr" (translate). Lets start by looking at the man page for "tr". We can see from the man page that tr converts/translates from one set to another.

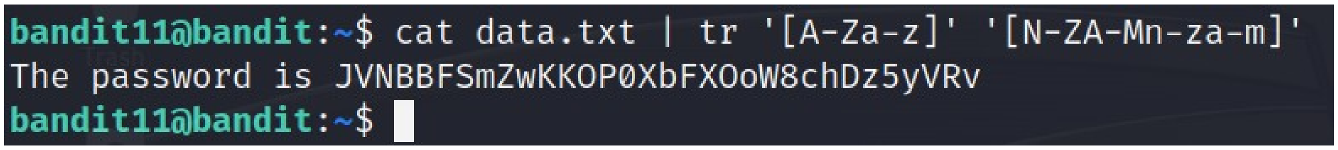
If we start slow this makes sense.. tr takes two sets, in our case we use the alphabet. So, tr '[A-Za-z]' → this is our first set if the letter is found in the input that we feed it, it will then translate it to our second set. With that being said we know we want to rotate by 13. Lets look at our second set: tr '[A-Za-z]' '[N-ZA-Mn-za-m]'.

we can think of this as follows: tr takes one input (data.txt for us) and two sets. The first set is like a filter looking for the characters in the first set coming in from the input → that character is then replaced in its mapped position in our second set.

The first set is the alphabet as we now it. The second set is our rotated alphabet (by 13). With our two sets being used it will equate to the following image from the ROT13 Wiki page:



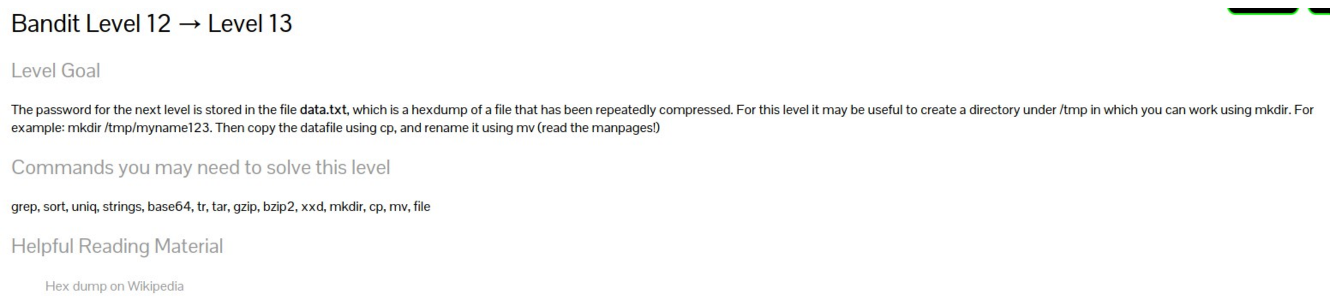
Lets apply this logic to our problem and see how it works out for us.



```
bandit11@bandit:~$ cat data.txt | tr '[A-Za-z]' '[N-ZA-Mn-za-m]'
The password is JVNBBFSmZwKKOP0XbFXOoW8chDz5yVRv
bandit11@bandit:~$ 
```

Hey, look at that.. Success!

level 12:



Bandit Level 12 → Level 13

Level Goal

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

Commands you may need to solve this level

grep, sort, uniq, strings, base64, tr, tar, gzip, bzip2, xxd, mkdir, cp, mv, file

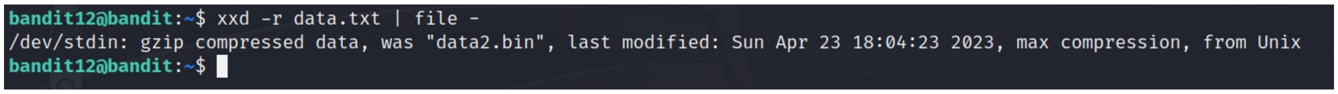
Helpful Reading Material

Hex dump on Wikipedia

According to the provided instructions we have a hexdump in our file. Looking at the recommended commands we will be using xxd to start off. According to the man page for xxd we need to use the “-r” flag for a reverse operation, converting the hexdump into a binary.

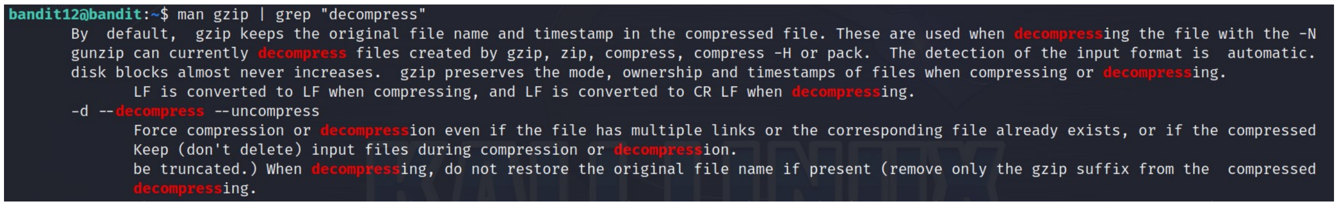
The instructions state we can use the /tmp directory to work out of.. we are going to use a shortcut method. Instead of using files to dump our output into we will tell our file command to use stdin as its input using the “-” character.

Lets start with our xxd command and see what type of compression we are working with, as the instructions state that we will have multiple levels of compression.



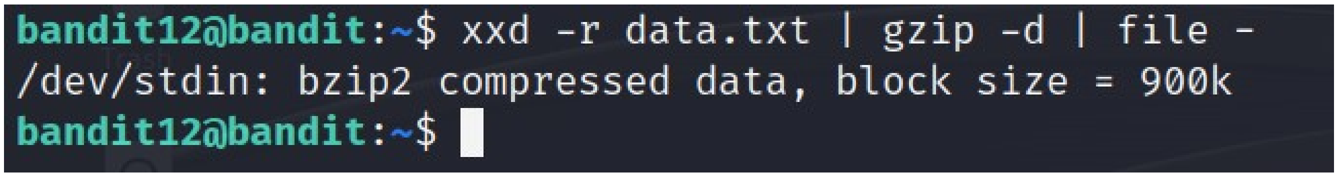
```
bandit12@bandit:~$ xxd -r data.txt | file -
/dev/stdin: gzip compressed data, was "data2.bin", last modified: Sun Apr 23 18:04:23 2023, max compression, from Unix
bandit12@bandit:~$
```

We can see we are dealing with gzip compression. Looking at the man page for gzip we can see to decompress we need to use the -d flag.



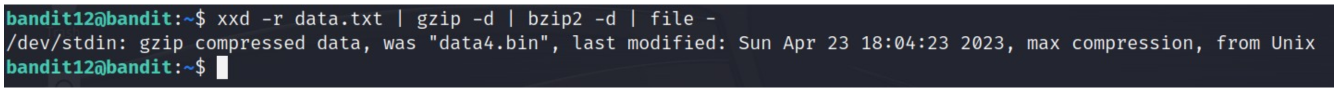
```
bandit12@bandit:~$ man gzip | grep "decompress"
       By  default,  gzip keeps the original file name and timestamp in the compressed file. These are used when decompressing the file with the -N
       gunzip can currently decompress files created by gzip, zip, compress, compress -H or pack.  The detection of the input format is  automatic.
       disk blocks almost never increases.  gzip preserves the mode, ownership and timestamps of files when compressing or decompressing.
              LF is converted to LF when compressing, and LF is converted to CR LF when decompressing.
       -d --decompress --uncompress
              Force compression or decompression even if the file has multiple links or the corresponding file already exists, or if the compressed
              Keep (don't delete) input files during compression or decompression.
              be truncated.) When decompressing, do not restore the original file name if present (remove only the gzip suffix from the  compressed
              decompressing.
```

Replacing our file command we will pipe our output from xxd into the standard input for gzip using the -d flag and then pipe that output into our file command.



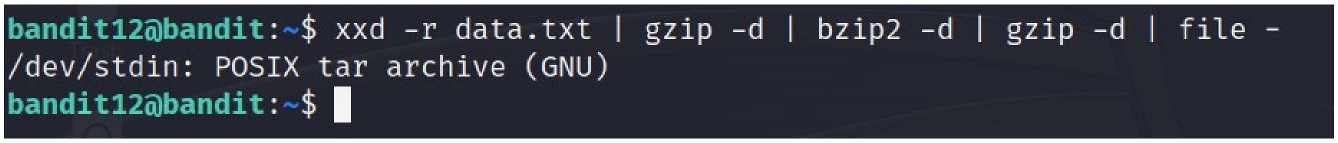
```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | file -
/dev/stdin: bzip2 compressed data, block size = 900k
bandit12@bandit:~$
```

As we can see from the output we are now presented with bzip2 compression. A look at the man page for bzip2 reveals that we once again will use the -d flag.



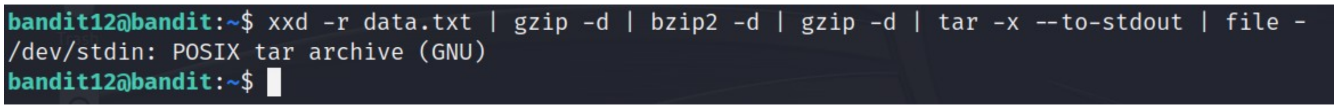
```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | bzip2 -d | file -
/dev/stdin: gzip compressed data, was "data4.bin", last modified: Sun Apr 23 18:04:23 2023, max compression, from Unix
bandit12@bandit:~$
```

Moving onto the next layer we can see we are back to another layer of gzip compression. Lets follow the same structure we have been using, as we know from previous experience we will use the -d flag with gzip.



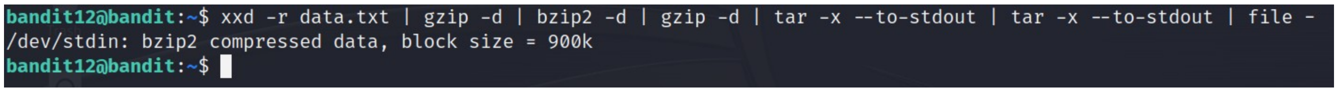
```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | bzip2 -d | gzip -d | file -
/dev/stdin: POSIX tar archive (GNU)
bandit12@bandit:~$
```

Our file command shows we are now dealing with a tar ball. After review of the man page we will need to use the “-x” flag and inform the command we want this sent to standard out, so we can pipe the output into our file command.



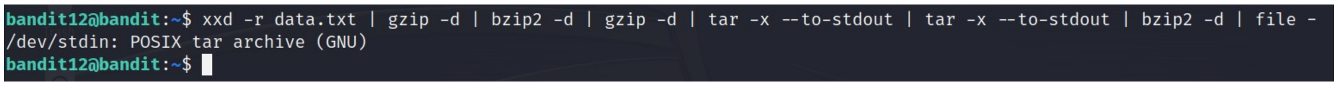
```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | bzip2 -d | gzip -d | tar -x --to-stdout | file -
/dev/stdin: POSIX tar archive (GNU)
bandit12@bandit:~$
```

Looks like another layer using tar. Lets repeat our last command and continue.



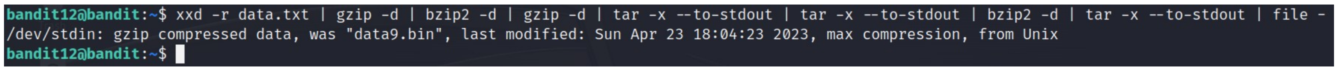
```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | bzip2 -d | gzip -d | tar -x --to-stdout | tar -x --to-stdout | file -
/dev/stdin: bzip2 compressed data, block size = 900k
bandit12@bandit:~$
```

We are now faced with another layer of bzip2. We have navigated this previously, so lets decompress this with the -d flag again.



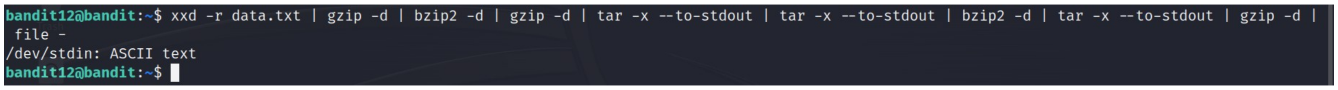
```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | bzip2 -d | gzip -d | tar -x --to-stdout | tar -x --to-stdout | bzip2 -d | file -
/dev/stdin: POSIX tar archive (GNU)
bandit12@bandit:~$
```

Once again we are working with tar based compression. Lets repeat our steps with tar using the -x flag and directing our output with –to-stdout.



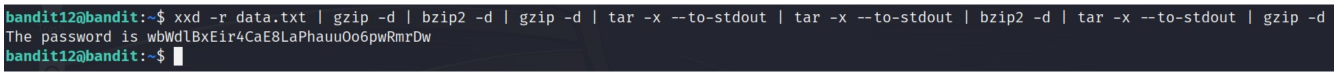
```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | bzip2 -d | gzip -d | tar -x --to-stdout | tar -x --to-stdout | bzip2 -d | tar -x --to-stdout | file -
/dev/stdin: gzip compressed data, was "data9.bin", last modified: Sun Apr 23 18:04:23 2023, max compression, from Unix
bandit12@bandit:~$
```

Reviewing the output we see we now have gzip compression, lets continue.



```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | bzip2 -d | gzip -d | tar -x --to-stdout | tar -x --to-stdout | bzip2 -d | tar -x --to-stdout | gzip -d |
 file -
/dev/stdin: ASCII text
bandit12@bandit:~$ 
```

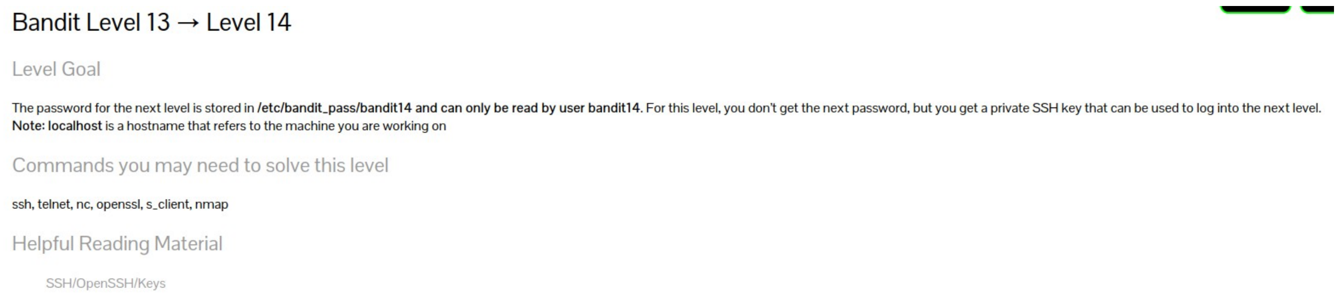
Okay, looks like we have made it. Instead of another layer of compression it looks like we have ASCII. Lets drop the file command and see what the output is.



```
bandit12@bandit:~$ xxd -r data.txt | gzip -d | bzip2 -d | gzip -d | tar -x --to-stdout | tar -x --to-stdout | bzip2 -d | tar -x --to-stdout | gzip -d
The password is wbWdlBxEir4CaE8LaPhauuOo6pwRmrDw
bandit12@bandit:~$ 
```

Success! It took a minute to work through, but we have obtained our flag.

level 13:


Bandit Level 13 → Level 14

Level Goal

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

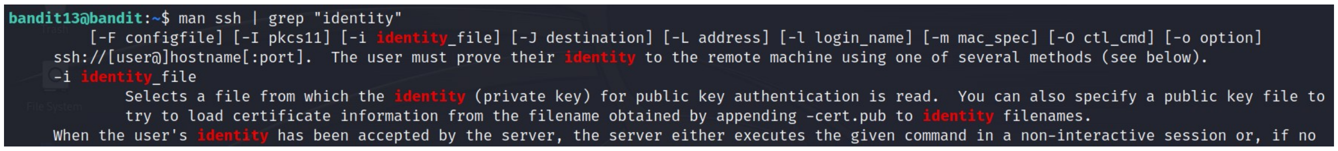
Commands you may need to solve this level

ssh, telnet, nc, openssl, s_client, nmap

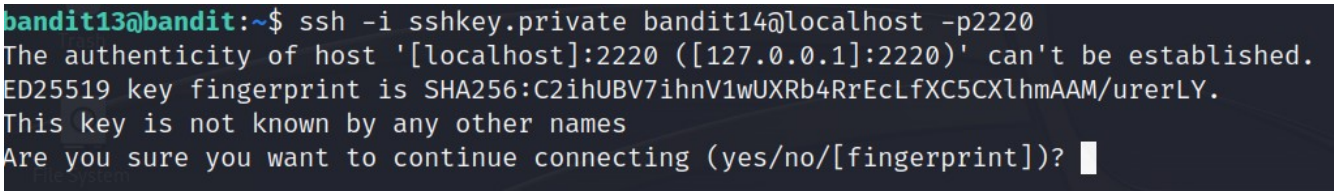
Helpful Reading Material

SSH/OpenSSH/Keys

As per the instructions we need to ssh into the next level. We can see that we have been provided a private sshkey in our current working directory. If we take a look at the man page for ssh we can see we need to use the -i flag to use an identity file.

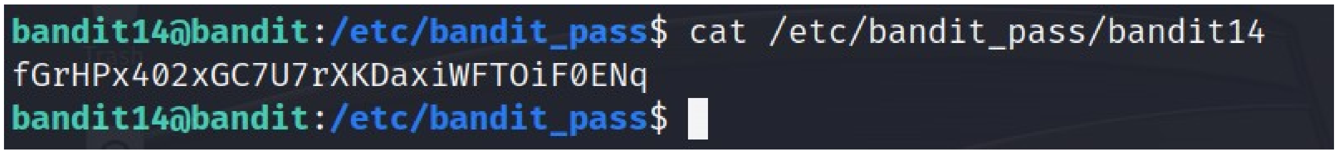

```
bandit13@bandit:~$ man ssh | grep "identity"
          [-F configfile] [-I pkcs11] [-i identity_file] [-J destination] [-L address] [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option]
     ssh://[user@]hostname[:port].  The user must prove their identity to the remote machine using one of several methods (see below).
     -i identity_file
             Selects a file from which the identity (private key) for public key authentication is read.  You can also specify a public key file to
             try to load certificate information from the filename obtained by appending -cert.pub to identity filenames.
     When the user's identity has been accepted by the server, the server either executes the given command in a non-interactive session or, if no
```

Lets go ahead and use the ssh private key to ssh into the next level. Unlike our other connections we are going to connect to bandit14 from bandit13. As we put this together we need to be aware of the wording in the instructions advising us to connect to localhost.


```
bandit13@bandit:~$ ssh -i sshkey.private bandit14@localhost -p2220
The authenticity of host '[localhost]:2220 ([127.0.0.1]:2220)' can't be established.
ED25519 key fingerprint is SHA256:C2ihUBV7ihnV1wUXRb4RrEcLfXC5CXlhmAAM/urerLY.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

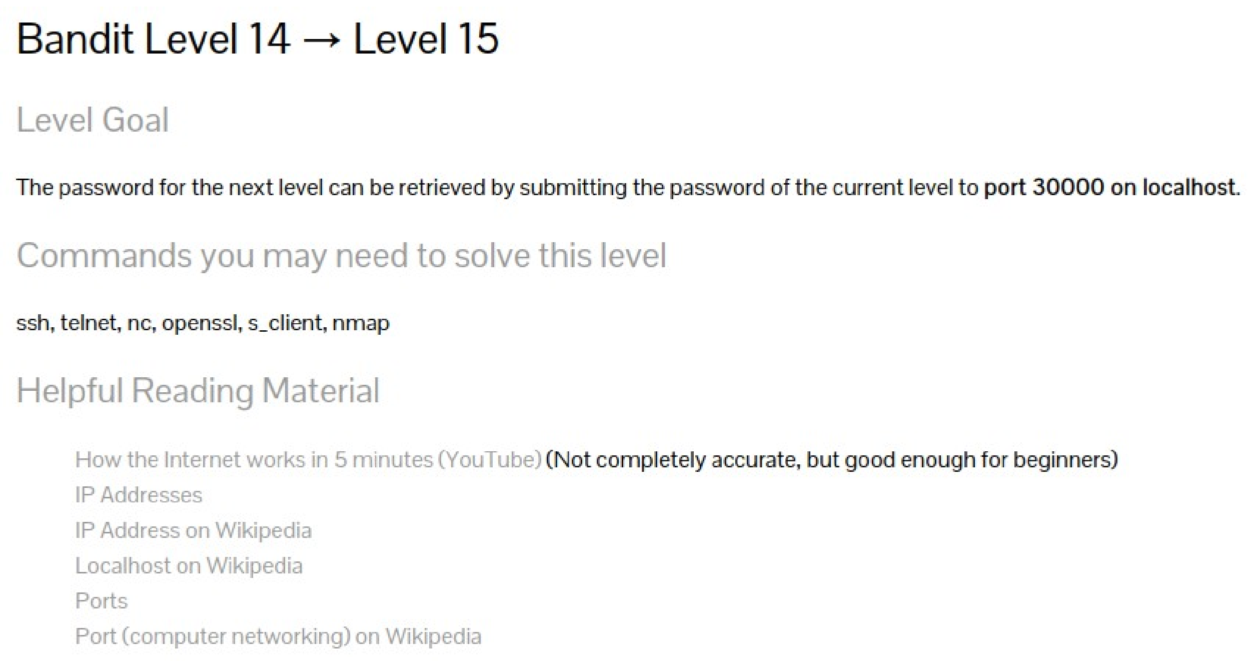
After executing our ssh command enter "yes" to add the connection to our known hosts.

Okay we are in bandit14. Lets go ahead and grab the password for the bandit14 login. We can obtain the password if we cat out the bandit14 password file located at "/etc/bandit_pass/bandit14".


```
bandit14@bandit:/etc/bandit_pass$ cat /etc/bandit_pass/bandit14
fGrHPx402xGC7U7rXKDaxiWFTOiF0ENq
bandit14@bandit:/etc/bandit_pass$
```

Success! Lets hop out of this connection and login with a password and continue on.

level 14:



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**.

Commands you may need to solve this level

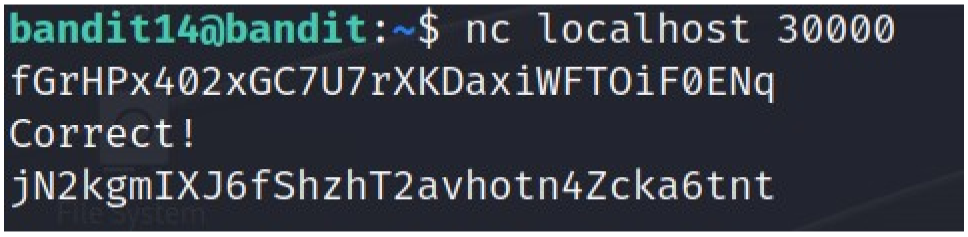
ssh, telnet, nc, openssl, s_client, nmap

Helpful Reading Material

- How the Internet works in 5 minutes (YouTube) (Not completely accurate, but good enough for beginners)
- IP Addresses
- IP Address on Wikipedia
- Localhost on Wikipedia
- Ports
- Port (computer networking) on Wikipedia

When we review the instructions and recommended commands for this level we can conclude we have a couple options to succeed here. If you do not have much experience with basic networking I would recommend reviewing the “Helpful Reading Material”.

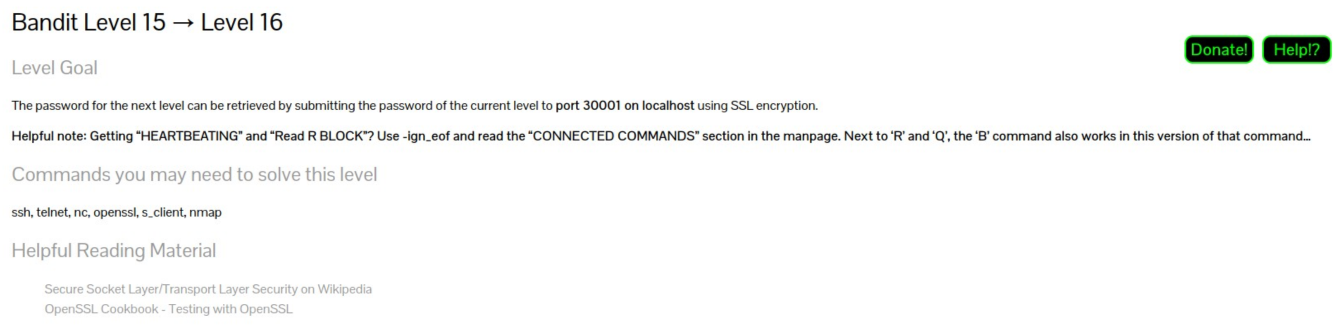
I am going to use the “nc” command in this scenerio, although this is not the only way to accomplish the goal at hand. Using netcat/nc we can use “localhost” as our host and port 30000. After we obtain a succesesful connection we will simply paste in the current password and hit enter.



```
bandit14@bandit:~$ nc localhost 30000
fGrHPx402xGC7U7rXKDaxiWFTOiF0ENq
Correct!
jN2kgmIXJ6fShzhT2avhotn4Zcka6tnt
```

Success! Lets move on.

level 15:



Bandit Level 15 → Level 16

Level Goal

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

Commands you may need to solve this level

ssh, telnet, nc, openssl, s_client, nmap

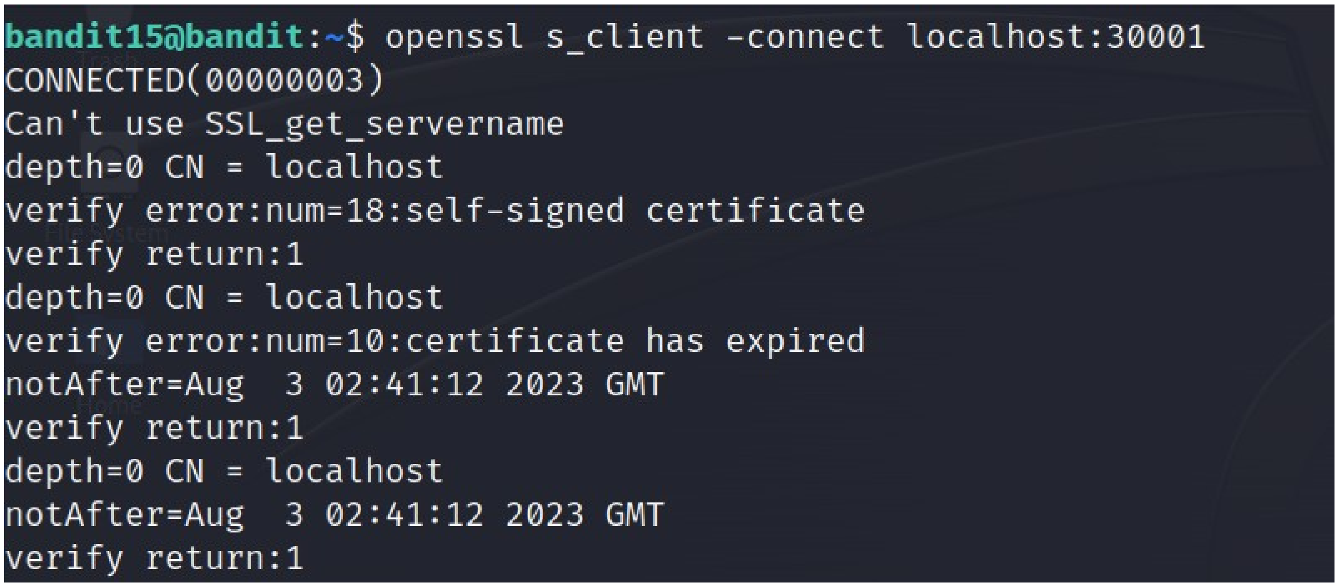
Helpful Reading Material

- Secure Socket Layer/Transport Layer Security on Wikipedia
- OpenSSL Cookbook - Testing with OpenSSL

This challenge seems to be pretty similar to the bandit14, difference is we need to use SSL to connect this time. Working off the recommended commands we are going to use the openssl "s_client".

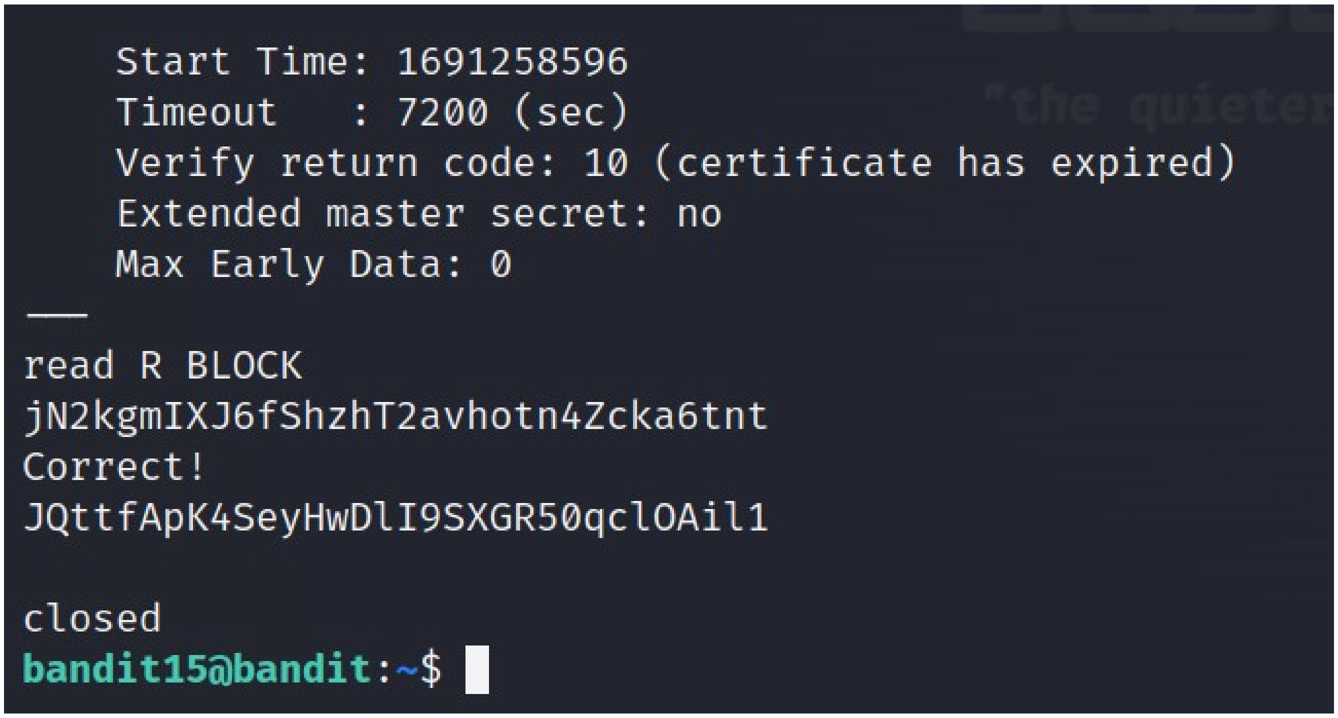
When we review the man page for openssl s_client it shows that we can use the "-connect" flag. When using the flag we can provide the host and port in the following format: host:port.

As per our last challenge when a connection is made we will provide the current password and we should get the flag we are looking for. Lets get started.



```
bandit15@bandit:~$ openssl s_client -connect localhost:30001
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 CN = localhost
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN = localhost
verify error:num=10:certificate has expired
notAfter=Aug  3 02:41:12 2023 GMT
verify return:1
depth=0 CN = localhost
notAfter=Aug  3 02:41:12 2023 GMT
verify return:1
```

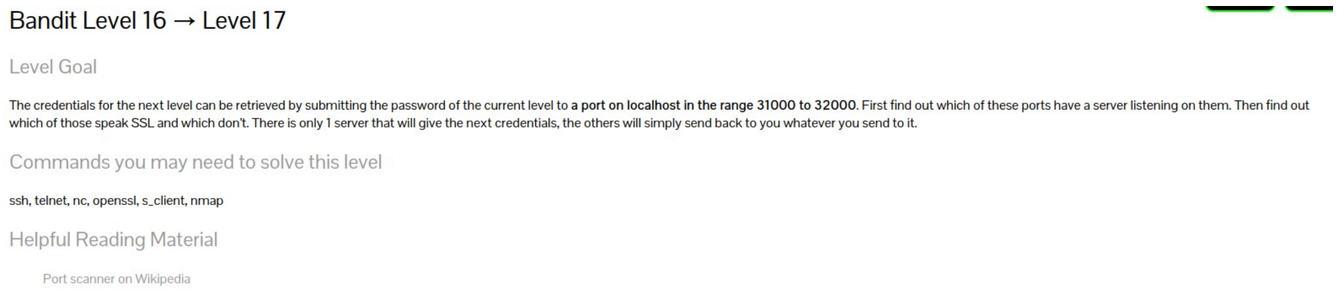
At the end of the response we can enter the current password.



```
    Start Time: 1691258596
    Timeout   : 7200 (sec)
    Verify return code: 10 (certificate has expired)
    Extended master secret: no
    Max Early Data: 0
---
read R BLOCK
jN2kgmIXJ6fShzhT2avhotn4Zcka6tnt
Correct!
JQttfApK4SeyHwDlI9SXGR50qclOAil1

closed
bandit15@bandit:~$ 
```

success!

level 16:



Bandit Level 16 → Level 17

Level Goal

The credentials for the next level can be retrieved by submitting the password of the current level to **a port on localhost in the range 31000 to 32000**. First find out which of these ports have a server listening on them. Then find out which of those speak SSL 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.

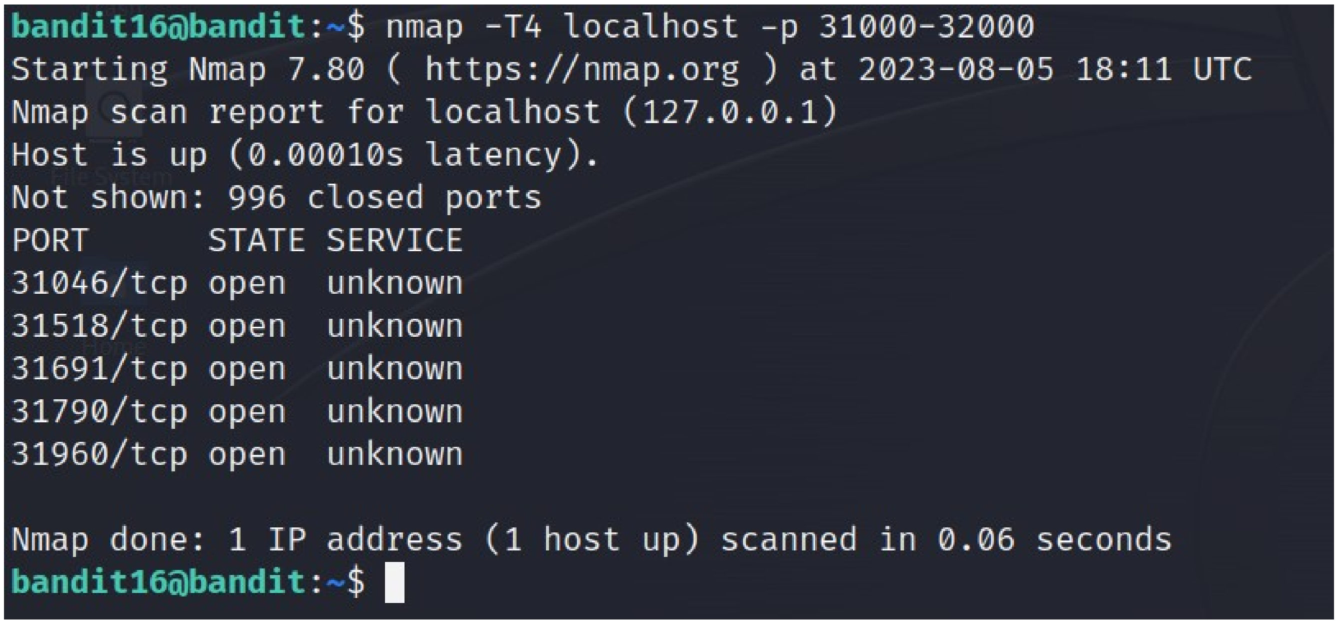
Commands you may need to solve this level

ssh, telnet, nc, openssl, s_client, nmap

Helpful Reading Material

Port scanner on Wikipedia

After reviewing the instructions we can break this down into a couple of steps. First we need to find the active listening port. Looking at the recommended commands we are clearly being pointed to nmap.

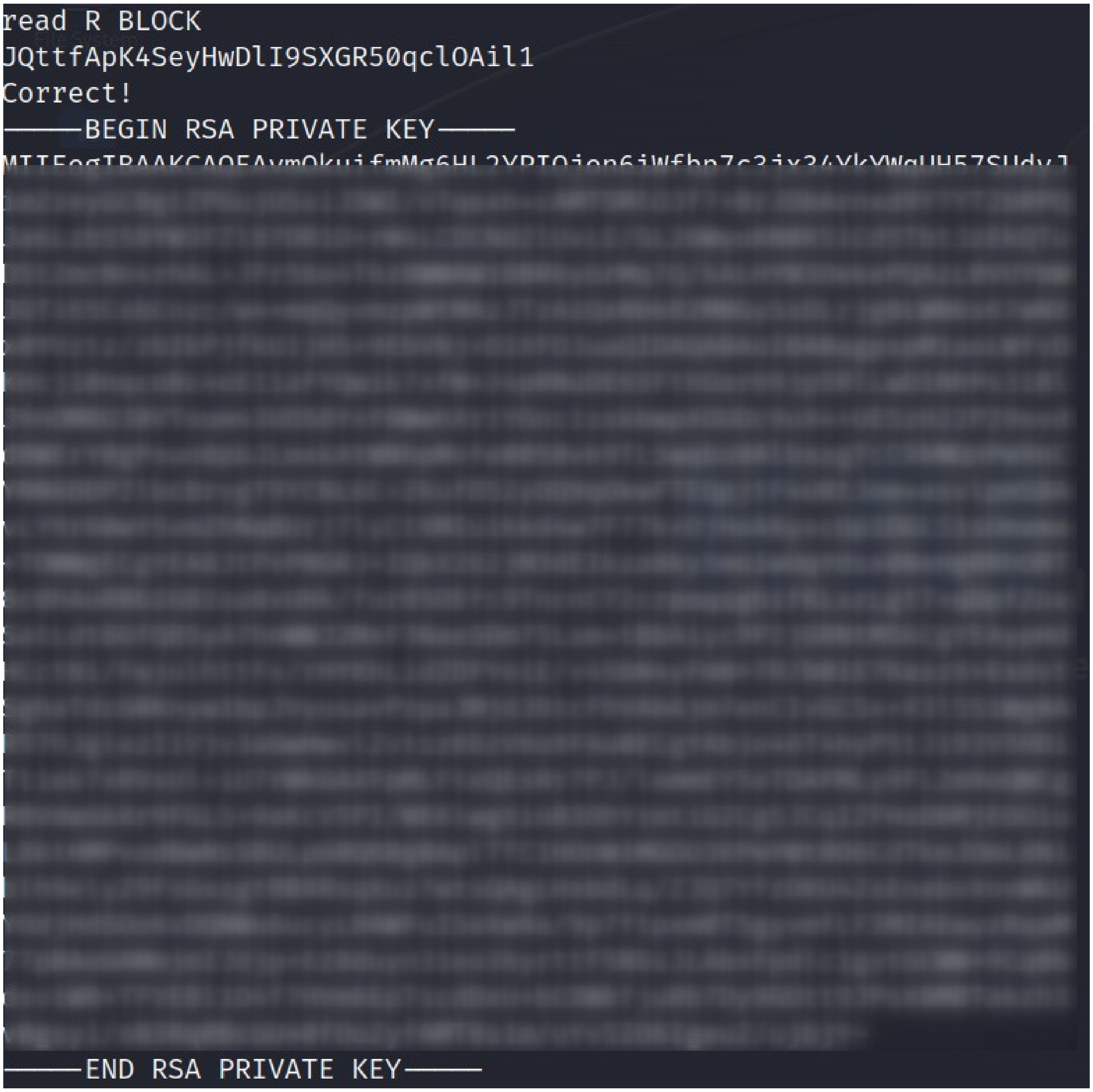


```
bandit16@bandit:~$ nmap -T4 localhost -p 31000-32000
Starting Nmap 7.80 ( https://nmap.org ) at 2023-08-05 18:11 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00010s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
31046/tcp open  unknown
31518/tcp open  unknown
31691/tcp open  unknown
31790/tcp open  unknown
31960/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
bandit16@bandit:~$ 
```

A breakdown of this nmap command is as follows:

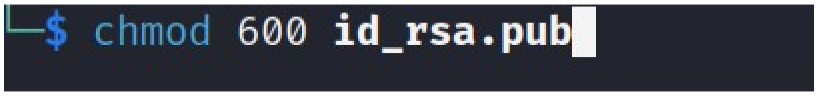
-T4	Setting the speed at which nmap will scan the ports
-p	Giving the port range to scan

In our nmap scan results we can see that we have 5 open ports to target. If we use openssl s_client to make an ssl connection (as per the instructions) we can see the only port that responds properly is port 31790. When we provide the password for the current level the response is “Correct!” and we get a private ssh key!



```
read R BLOCK
JQttfApK4SeyHwDlI9SXGR50qclOAil1
Correct!
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAymOkuifmMg6HL2YPIQjon6iWfbn7c3jx34YkYWqUH57SUdyJ
[redacted]
-----END RSA PRIVATE KEY-----
```

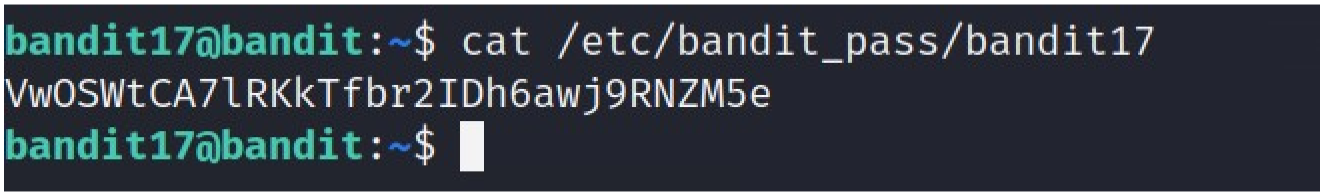
Lets copy the the private RSA key and exit out of our connection. In our attack box we will create a file and drop the copied RSA private key into it. After we have our key saved in the file we will modify the permisions with chmod.



```
└─$ chmod 600 id_rsa.pub
```

Now we are ready to login with our new RSA key. We have done this before and know that this process takes the -i flag to SSH in with an include file.

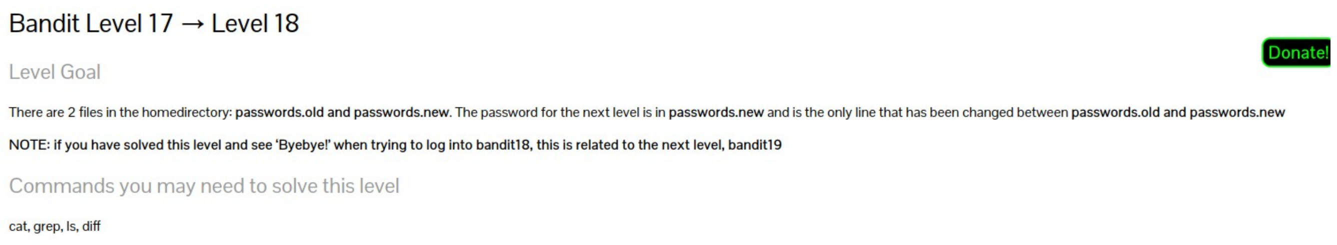
Okay, we have succesfully gained access to bandit17 with our SSH key. Lets go ahead and grab the password for bandit17.



```
bandit17@bandit:~$ cat /etc/bandit_pass/bandit17
VwOSWtCA7lRKkTfbr2IDh6awj9RNZM5e
bandit17@bandit:~$ 
```

success!

level 17:



Bandit Level 17 → Level 18

Donate!

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**

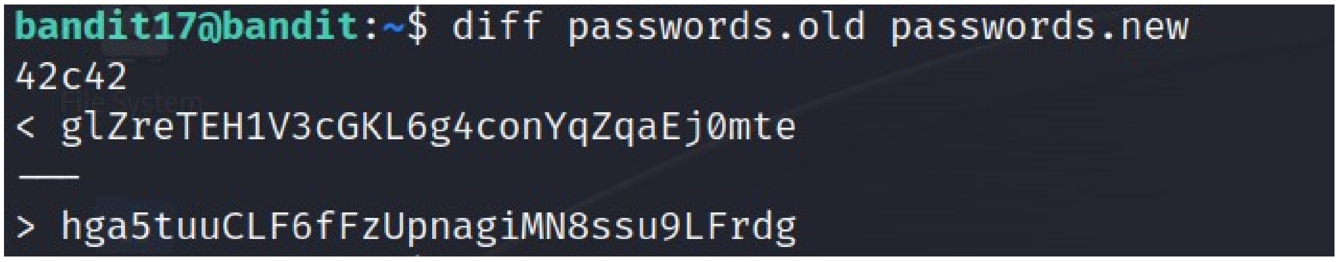
NOTE: if you have solved this level and see 'Byebye!' when trying to log into bandit18, this is related to the next level, bandit19

Commands you may need to solve this level

cat, grep, ls, diff

When we list out the contents of our current working directory we can see that we have two password files. It seems that our flag is the only line that has been changed in the passwords.new when compared to the passwords.old file. This sounds like a job for the "diff" command.

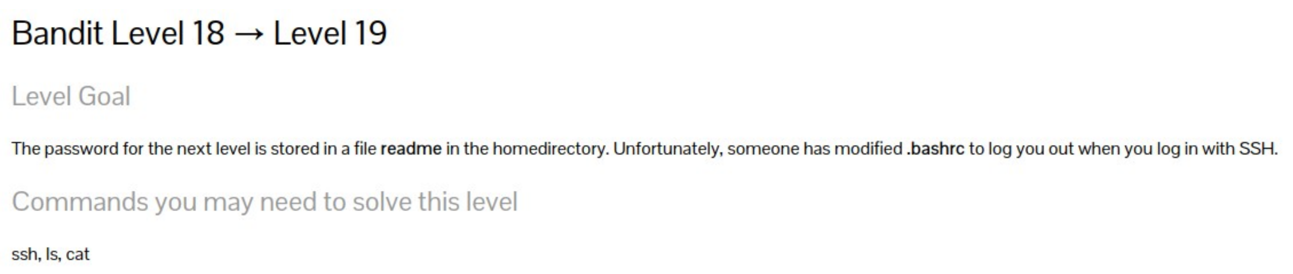
It is worth noting that when using the "diff" command the output is shown accourding to the order in which you executed the command. This means if you enter old new, the old will be listed first and the new listed second.



```
bandit17@bandit:~$ diff passwords.old passwords.new
42c42
< glZreTEH1V3cGKL6g4conYqZqaEj0mte
---
> hga5tuuCLF6fFzUpnagiMN8ssu9LFrdg
```

Success! The Second listed password (in this case) would be our flag.

level 18:


Bandit Level 18 → Level 19

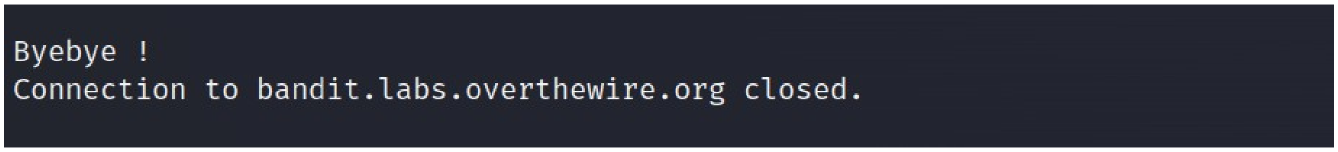
Level Goal

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.

Commands you may need to solve this level

ssh, ls, cat

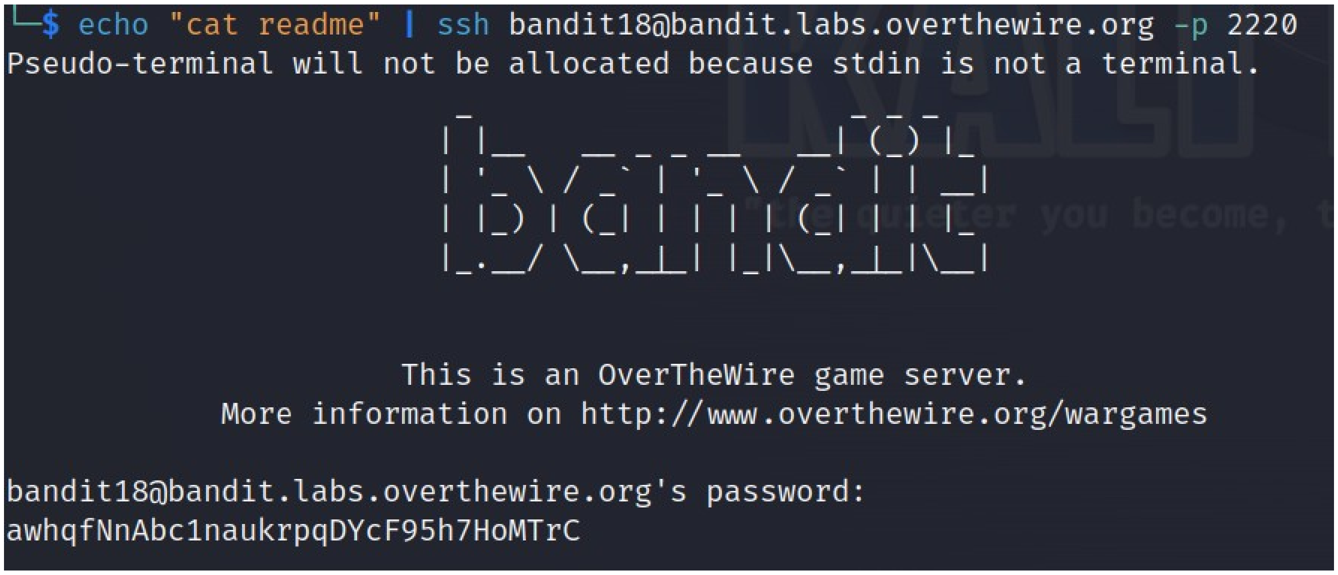
When we try to login to bandit18 the connection is promptly killed and we are back on our box.


```
Byebye !
Connection to bandit.labs.overthewire.org closed.
```

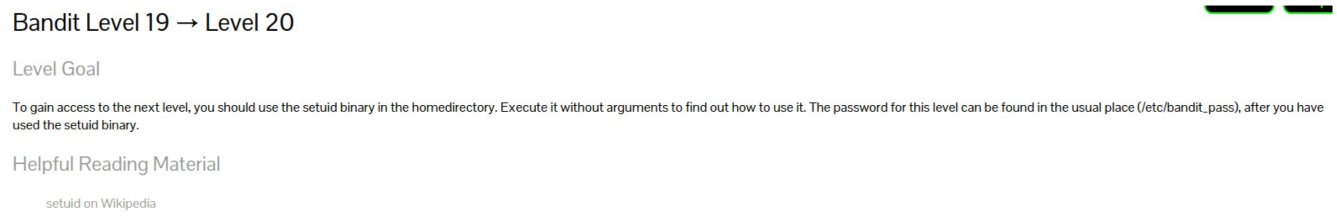
Our instructions tell us that the password for the next level are stored in a file named “readme” in the home directory. We are able to login even though we are kicked off right away, maybe we can run a command at the same time as logging in.

lets try to cat out the contents of the “readme” file during login.

echo "cat readme" | ssh bandit18@bandit.labs.overthewire.org -p 2220


```
└─$ echo "cat readme" | ssh bandit18@bandit.labs.overthewire.org -p 2220
Pseudo-terminal will not be allocated because stdin is not a terminal.
                        _                     _ _ _
                       | |__   __ _ _ __   __| (_) |_
                       | '_ \ / _` | '_ \ / _` | | __|
                       | |_) | (_| | | | | (_| | | |_
                       |_.__/ \__,_|_| |_|\__,_|_|\__|


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

bandit18@bandit.labs.overthewire.org's password:
awhqfNnAbc1naukrpqDYcF95h7HoMTrC
```
success!

level 19:


Bandit Level 19 → Level 20

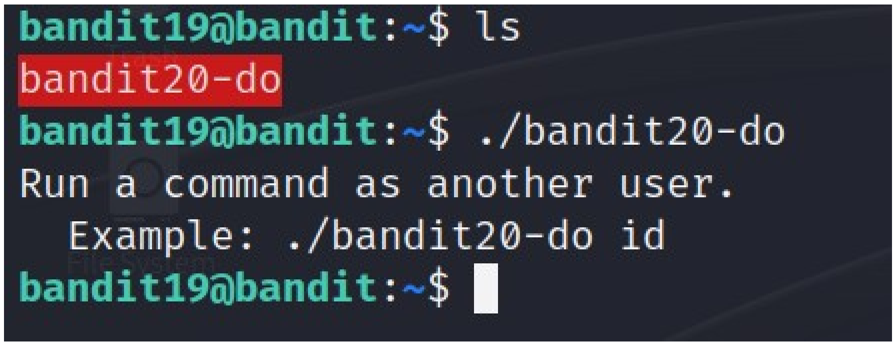
Level Goal

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.

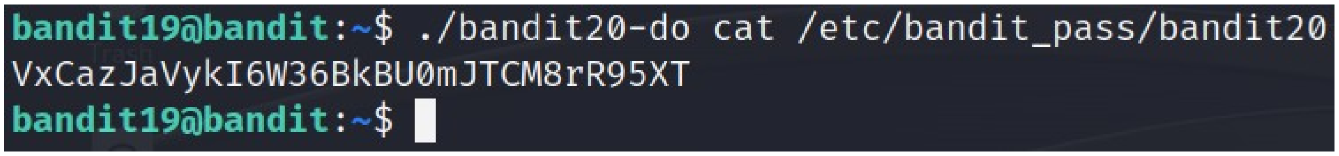
Helpful Reading Material

setuid on Wikipedia

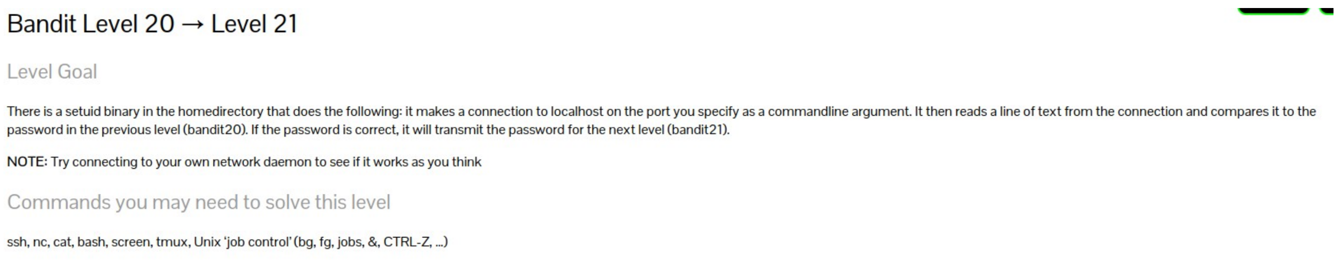
Just as the instructions stated we can see the setuid binary in our home directory. If we run it with no arguments it tells us to provide a ID.


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

Accourding to the output this binary allows us to run commands as another user (safe to assume we should be able to run as bandit20). If we can run this binary as bandit20 lets just cat out the file for the bandit20 password.


```
bandit19@bandit:~$ ./bandit20-do cat /etc/bandit_pass/bandit20
VxCazJaVykI6W36BkBU0mJTCM8rR95XT
bandit19@bandit:~$ 
```

Level 20:


Bandit 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).

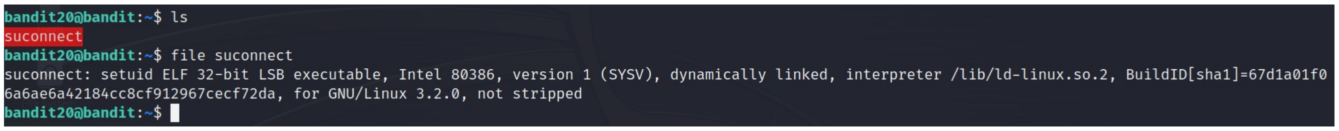
NOTE: Try connecting to your own network daemon to see if it works as you think

Commands you may need to solve this level

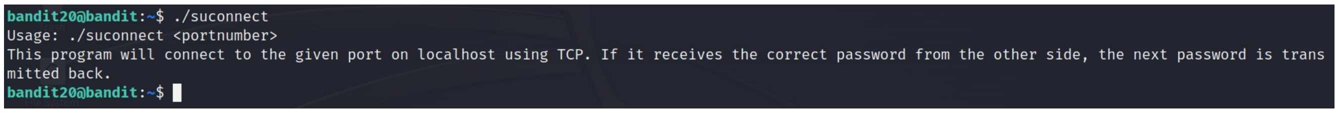
ssh, nc, cat, bash, screen, tmux, Unix 'job control' (bg, fg, jobs, &, CTRL-Z, ...)

Reviewing the instructions for this level we find out we have another setuid binary to play with. From the information we have reviewed this binary expects a port number, the binary then checks that port number on localhost and expects to receive the password for bandit20. If the binary recieves the correct password from the provided port we should get the flag we are looking for. Sounds like a job for a nc listener. Lets check it out.

After logging in to bandit20 we can see the binary and if we check the file type we can also see that it is a setuid 32-bit elf, which is what we expected anyway.

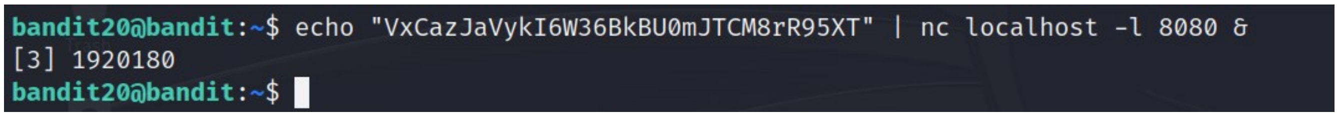

```
bandit20@bandit:~$ ls
suconnect
bandit20@bandit:~$ file suconnect
suconnect: setuid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=67d1a01f0
6a6ae6a42184cc8cf912967cecf72da, for GNU/Linux 3.2.0, not stripped
bandit20@bandit:~$
```

If we run the binary with no args we can get a little more information.

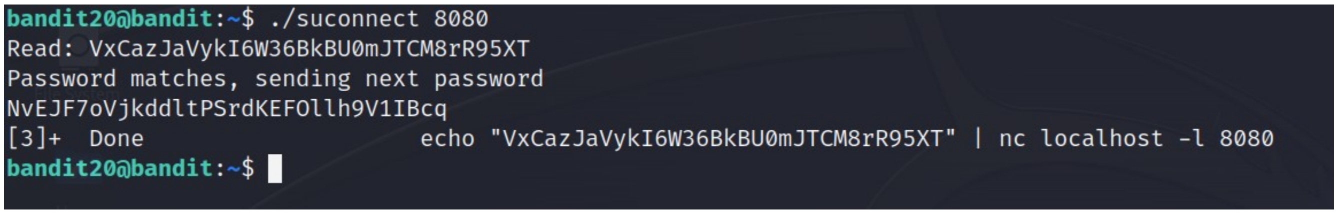

```
bandit20@bandit:~$ ./suconnect
Usage: ./suconnect <portnumber>
This program will connect to the given port on localhost using TCP. If it receives the correct password from the other side, the next password is trans
mitted back.
bandit20@bandit:~$
```

This clarifies what we knew. The program will connect to the port we provide on localhost using a TCP connection. If the program recieves the correct password (bandit20 password) it will then give us the password to the next level.

Lets start with running a netcat listener that we pipe the bandit20 password into and run it as a background job listening on localhost port 8080.


```
bandit20@bandit:~$ echo "VxCazJaVykI6W36BkBU0mJTCM8rR95XT" | nc localhost -l 8080 &
[3] 1920180
bandit20@bandit:~$
```

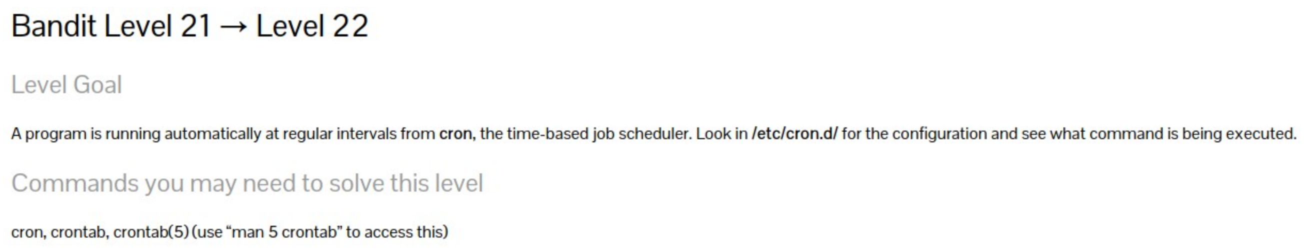
Now that we have the background job running and providing the information the program is expecting lets run the program providing the port number nc is listening on.



```
bandit20@bandit:~$ ./suconnect 8080
Read: VxCazJaVykI6W36BkBU0mJTCM8rR95XT
Password matches, sending next password
NvEJF7oVjkddltPSrdKEFOllh9V1IBcq
[3]+  Done                    echo "VxCazJaVykI6W36BkBU0mJTCM8rR95XT" | nc localhost -l 8080
bandit20@bandit:~$ 
```

Success!

level 21:



Bandit Level 21 → Level 22

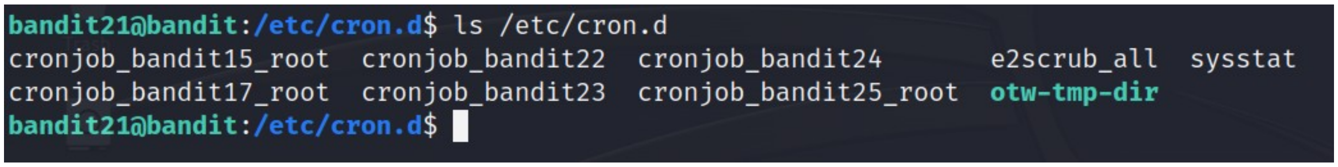
Level Goal

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

Commands you may need to solve this level

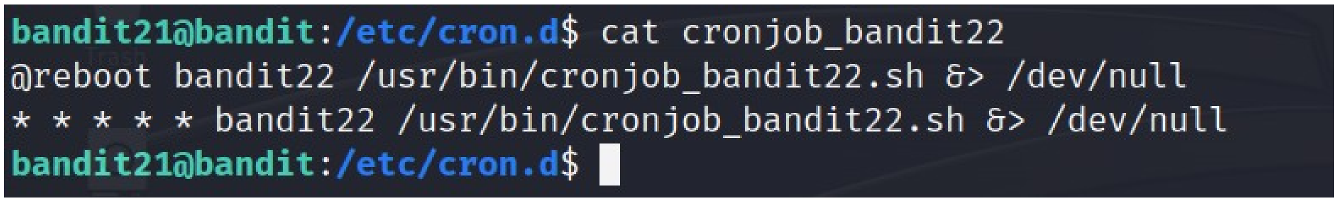
cron, crontab, crontab(5) (use "man 5 crontab" to access this)

After reading the instructions lets go check out "cron.d" and see what we are looking at.



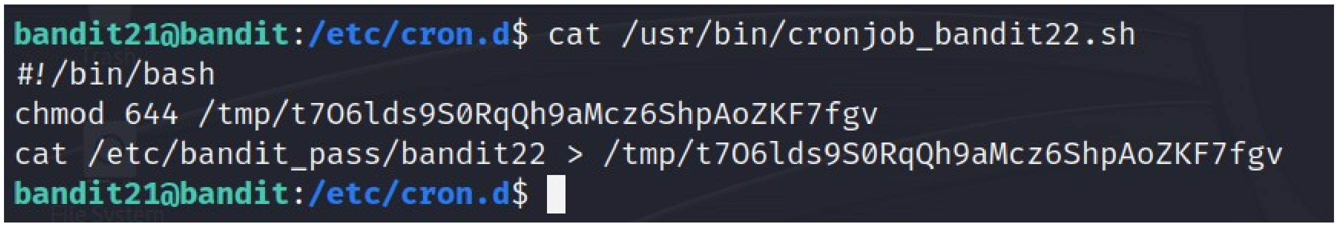
```
bandit21@bandit:/etc/cron.d$ ls /etc/cron.d
cronjob_bandit15_root  cronjob_bandit22  cronjob_bandit24       e2scrub_all  sysstat
cronjob_bandit17_root  cronjob_bandit23  cronjob_bandit25_root  otw-tmp-dir
bandit21@bandit:/etc/cron.d$ 
```

After reviewing the directory the next logical step for us would be to see what is in "cronjob_bandit22".



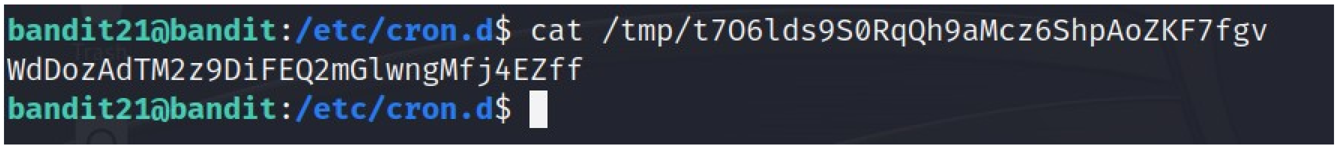
```
bandit21@bandit:/etc/cron.d$ cat cronjob_bandit22
@reboot bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
* * * * * bandit22 /usr/bin/cronjob_bandit22.sh &> /dev/null
bandit21@bandit:/etc/cron.d$ 
```

So the cronjob shell script located at "/usr/bin/cronjob_bandit22.sh" is running on the regular. Lets see what the script is doing.



```
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 we step through the process it looks like the password for bandit22 should be located at "/tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv". Lets go collect our flag!

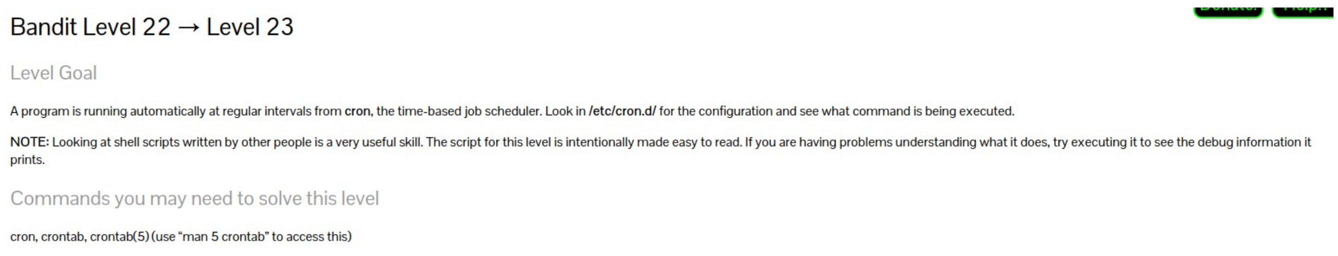


```
bandit21@bandit:/etc/cron.d$ cat /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
WdDozAdTM2z9DiFEQ2mGlwngMfj4EZff
bandit21@bandit:/etc/cron.d$ 
```

Success!

Level 22:

level 22 → level 23



Bandit Level 22 → Level 23

Level Goal

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

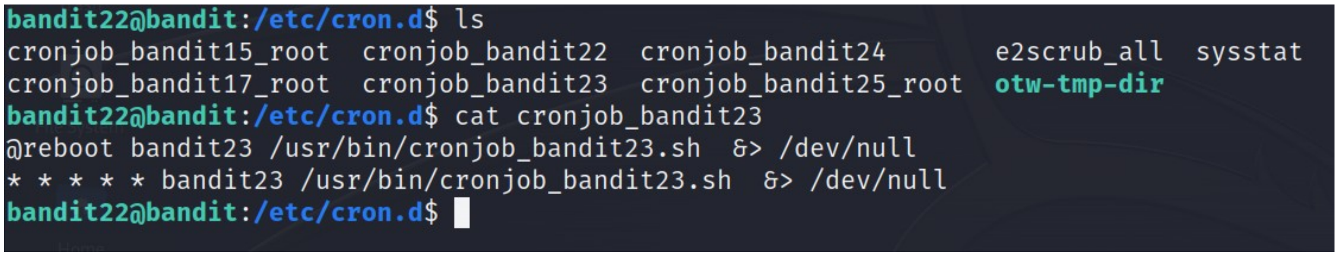
NOTE: Looking at shell scripts written by other people is a very useful skill. The script for this level is intentionally made easy to read. If you are having problems understanding what it does, try executing it to see the debug information it prints.

Commands you may need to solve this level

cron, crontab, crontab(5) (use "man 5 crontab" to access this)

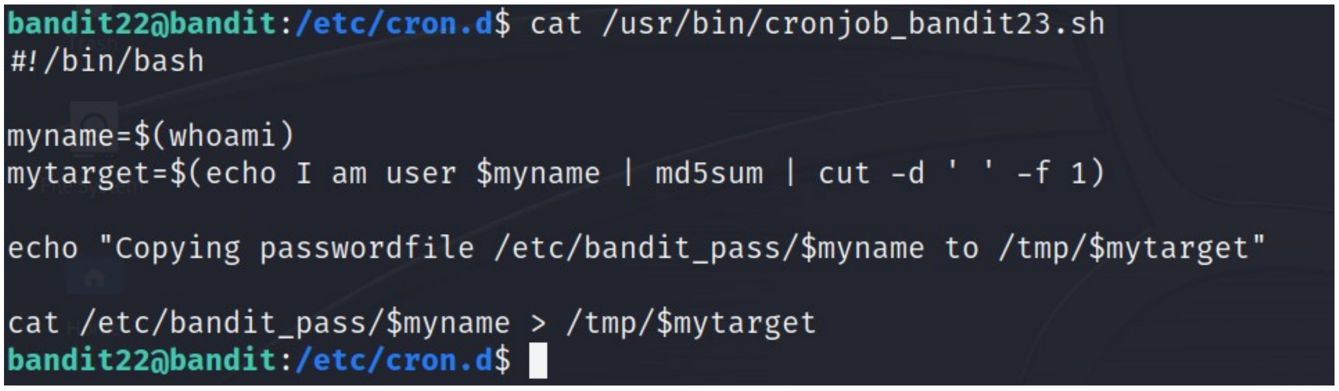
In Bandit22 it seems we will be revisiting the cron directory. Lets see what we are working with.

After logging into bandit22 lets revisit the "/etc/cron.d" directory and cat out the contents of the "cronjob_bandit23" file.



```
bandit22@bandit:/etc/cron.d$ ls
cronjob_bandit15_root  cronjob_bandit22  cronjob_bandit24       e2scrub_all  sysstat
cronjob_bandit17_root  cronjob_bandit23  cronjob_bandit25_root  otw-tmp-dir
bandit22@bandit:/etc/cron.d$ cat cronjob_bandit23
@reboot bandit23 /usr/bin/cronjob_bandit23.sh  &> /dev/null
* * * * * bandit23 /usr/bin/cronjob_bandit23.sh  &> /dev/null
bandit22@bandit:/etc/cron.d$
```

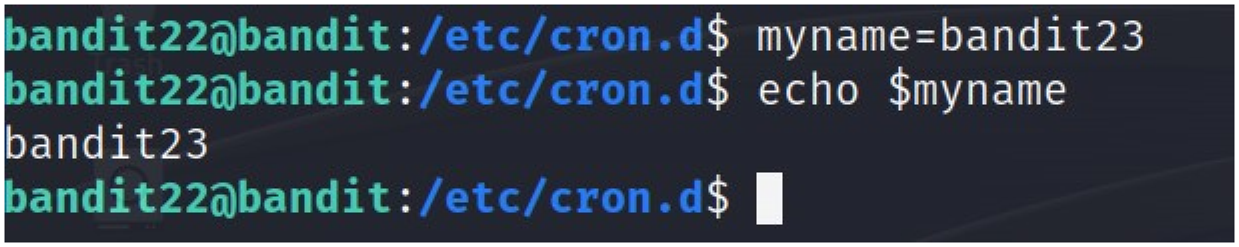
From the contents of the file we can see a cronjob is running a script at reboot and on the regular. The script is located at "/usr/bin/cronjob_bandit23.sh". We will go ahead a take a look at what the script is doing.



```
bandit22@bandit:/etc/cron.d$ cat /usr/bin/cronjob_bandit23.sh
#!/bin/bash

myname=$(whoami)
mytarget=$(echo I am user $myname | md5sum | cut -d ' ' -f 1)

echo "Copying passwordfile /etc/bandit_pass/$myname to /tmp/$mytarget"

cat /etc/bandit_pass/$myname > /tmp/$mytarget
bandit22@bandit:/etc/cron.d$
```

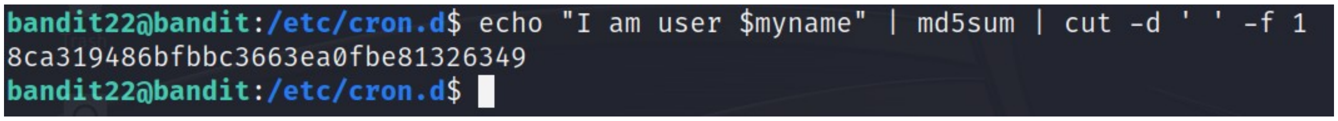
Lets breakdown this shell script. A variable of "myname" is set to the user by means of the "whoami" command, then a string using that variable is piped into the "md5sum" command. It is ultimatley cleaned up by using "cut" to remove the dash and white space from the end of the hash value and stored in the variable "mytarget". After the hash is created the script then stores the password for the username stored in "myname" in "/tmp/$mytarget"

Lets declare our own variable and store bandit23 in it.



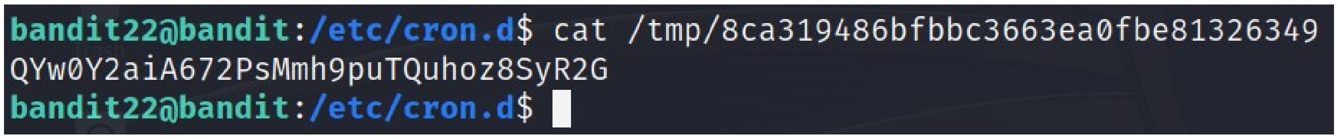
```
bandit22@bandit:/etc/cron.d$ myname=bandit23
bandit22@bandit:/etc/cron.d$ echo $myname
bandit23
bandit22@bandit:/etc/cron.d$ 
```

Now that we have “bandit23” stored in the “myname” variable lets see what the output of the bash one liner in the script would output. Taking this step should give us the location of the password for the next level.



```
bandit22@bandit:/etc/cron.d$ echo "I am user $myname" | md5sum | cut -d ' ' -f 1
8ca319486bfbbc3663ea0fbe81326349
bandit22@bandit:/etc/cron.d$ 
```

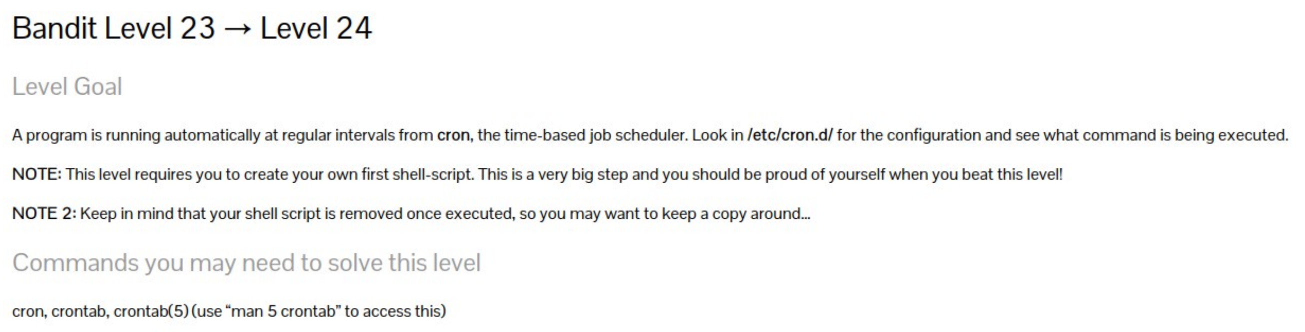
That looks promising.. If our theory is correct, we should have the location of the flag. Lets check.



```
bandit22@bandit:/etc/cron.d$ cat /tmp/8ca319486bfbbc3663ea0fbe81326349
QYw0Y2aiA672PsMmh9puTQuhoz8SyR2G
bandit22@bandit:/etc/cron.d$ 
```

success!

level 23:



Bandit Level 23 → Level 24

Level Goal

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

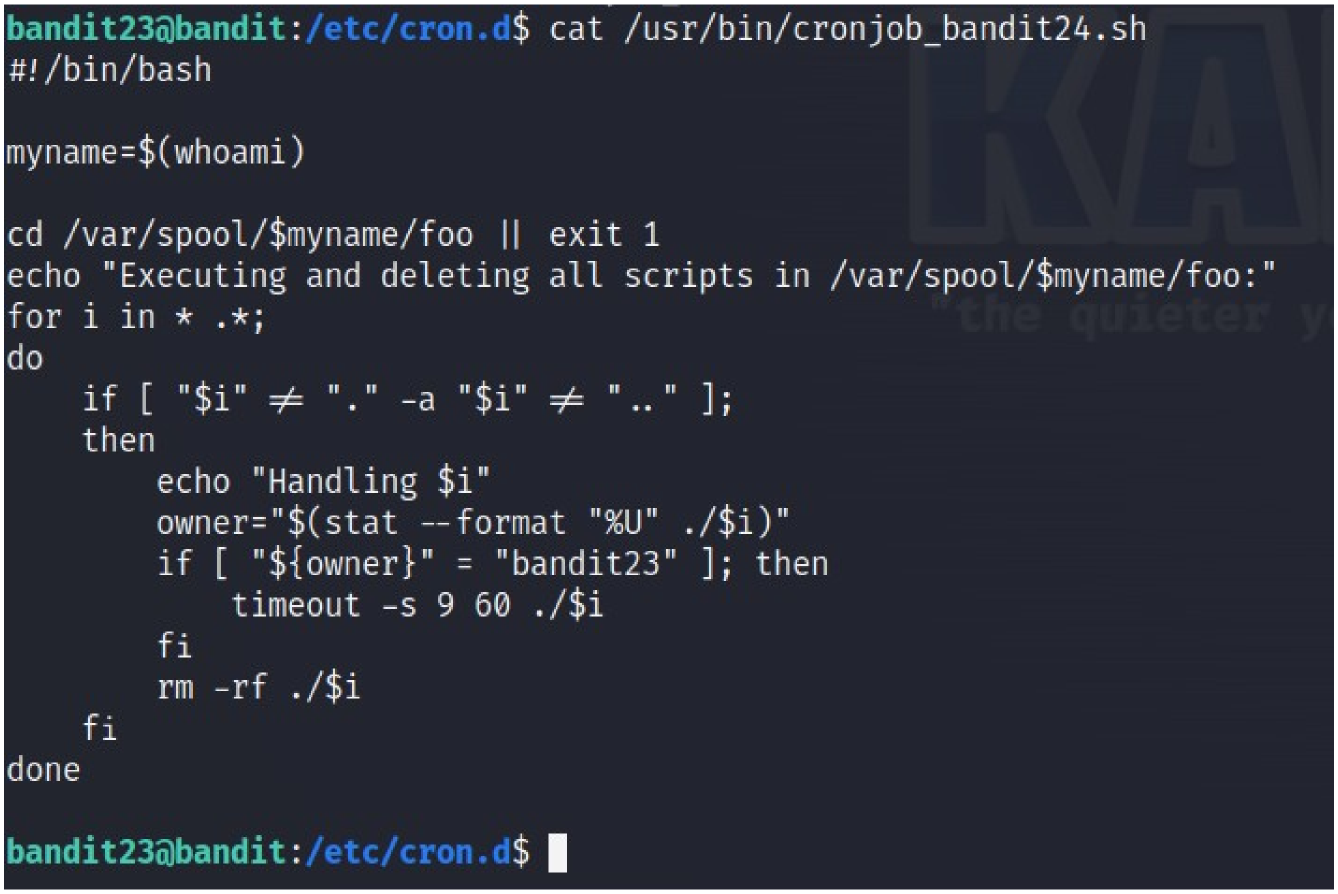
NOTE: This level requires you to create your own first shell-script. This is a very big step and you should be proud of yourself when you beat this level!

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

Commands you may need to solve this level

cron, crontab, crontab(5) (use "man 5 crontab" to access this)

Following the normal flow here we will check "/etc/cron.d/cronjob_bandit24" and see that our script is located in "/usr/bin/cronjob_bandit24.sh". Lets see what is going on with this script. We just need to cat out the contents of the script.



```
bandit23@bandit:/etc/cron.d$ cat /usr/bin/cronjob_bandit24.sh
#!/bin/bash

myname=$(whoami)

cd /var/spool/$myname/foo || exit 1
echo "Executing and deleting all scripts in /var/spool/$myname/foo:"
for i in * .*;
do
    if [ "$i" ≠ "." -a "$i" ≠ ".." ];
    then
        echo "Handling $i"
        owner="$(stat --format "%U" ./$i)"
        if [ "${owner}" = "bandit23" ]; then
            timeout -s 9 60 ./$i
        fi
        rm -rf ./$i
    fi
done

bandit23@bandit:/etc/cron.d$ 
```

Lets breakdown this script step by step.

myname =$(whoami) : *The myname variable is populated with the username of the user running the script.*

cd var/spool/$myname/foo || exit 1 : *the script is going to cd into listed directory, if not it will exit with an error code.*

echo "Executing and deleting all scripts in /var/spool/$myname/foo:" : *Looks like the use case for the script is to run all scripts in the listed directory then wipes the scripts.*

for i in * .*; : *The script enters into a for loop.*

if ["$i" != "." -a "$i" != ".."]; : *Checking to make sure the name does not start with a "." or "..".*

  echo "Handling $i

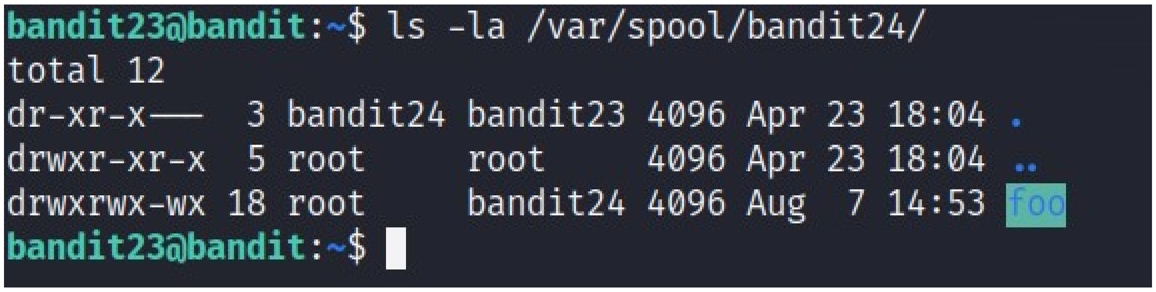
  owner="$(stat --format "%U" ./$i)"

  if ["${owner}" = "bandit23"]; then

   timeout -s 9 60 ./$i" : *Script has some formating, but the big takeaway here is that it is checking to see if the owner of the script is bandit23. If the owner of the script is bandit23 it is going to delay and after 60 seconds the process with end and delete the script.*

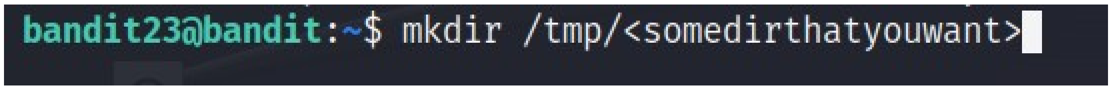
The script ends by wiping every file/script in the directory.

With all this information lets check out the privileges set on the "foo" directory.



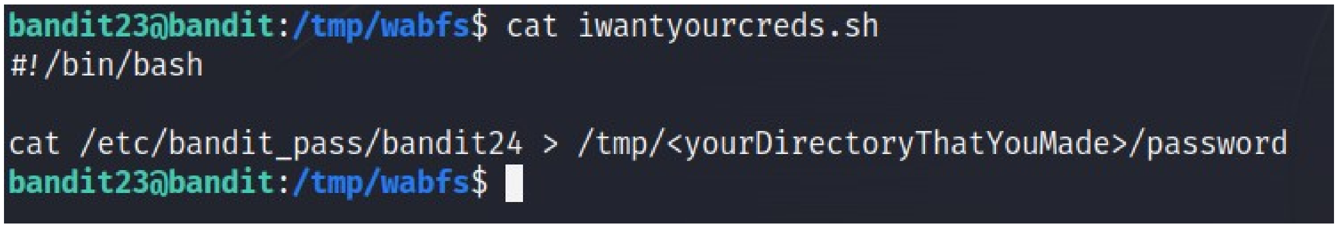
```
bandit23@bandit:~$ ls -la /var/spool/bandit24/
total 12
dr-xr-x---  3 bandit24 bandit23 4096 Apr 23 18:04 .
drwxr-xr-x  5 root     root     4096 Apr 23 18:04 ..
drwxrwx-wx 18 root     bandit24 4096 Aug  7 14:53 foo
bandit23@bandit:~$
```

We can see that the scripts in this directory are going to run as bandit24, which is nice to know. In even better news we have write access to this directory! We do not have read access, but the instructions did say we will need to write a shell script and we will need to keep a "copy" around. Lets create a tmp directory and write a simple script.



```
bandit23@bandit:~$ mkdir /tmp/<somedirthatyouwant>
```

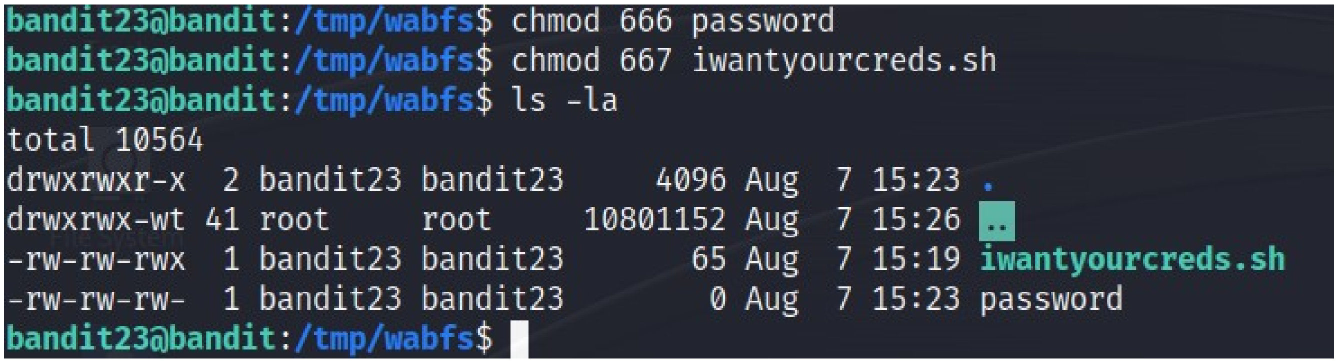
Now that we are in our "/tmp" directory lets write up our script. We know that we need to get the password for the next level and we know where that is located (/etc/bandit_pass/bandit24), we also know that this script will be ran as bandit24. Lets get to work, I am going to open up a blank file with vim using the .sh extension.



```
bandit23@bandit:/tmp/wabfs$ cat iwantyourcreds.sh
#!/bin/bash

cat /etc/bandit_pass/bandit24 > /tmp/<yourDirectoryThatYouMade>/password
bandit23@bandit:/tmp/wabfs$
```

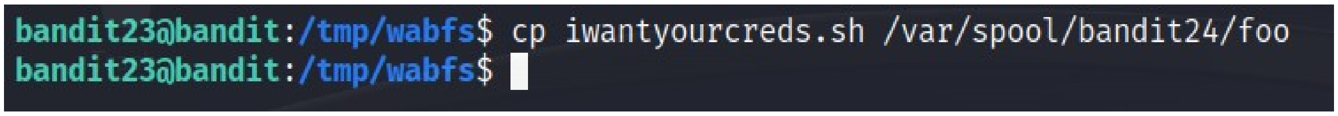
This is a very basic script, but it does what we need it to do. With this script we simply cat out the contents of the bandit24 file and put that output into a file by the name of password in our temp directory.

Next we need to create the password file using the touch command and adjust the privileges on our files. We need to allow the script to be executable and allow write access to our password file.



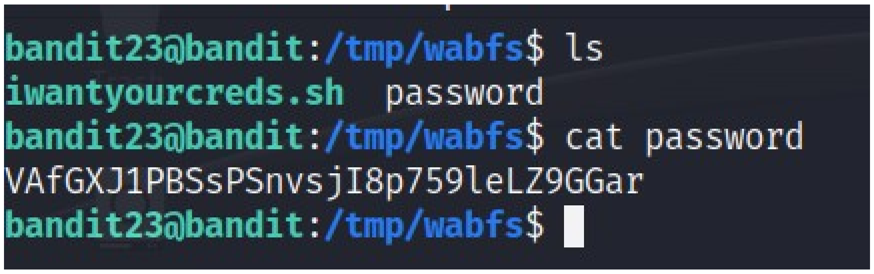
```
bandit23@bandit:/tmp/wabfs$ chmod 666 password
bandit23@bandit:/tmp/wabfs$ chmod 667 iwantyourcreds.sh
bandit23@bandit:/tmp/wabfs$ ls -la
total 10564
drwxrwxr-x  2 bandit23 bandit23     4096 Aug  7 15:23 .
drwxrwx-wt 41 root     root     10801152 Aug  7 15:26 ..
-rw-rw-rwx  1 bandit23 bandit23       65 Aug  7 15:19 iwantyourcreds.sh
-rw-rw-rw-  1 bandit23 bandit23        0 Aug  7 15:23 password
bandit23@bandit:/tmp/wabfs$
```

Looks good, now lets use the cp command to copy our file from the tmp directory to the foo directory. If all has been done correctly we should get the password for the next level in our password file in about 60 seconds or less.



```
bandit23@bandit:/tmp/wabfs$ cp iwantyourcreds.sh /var/spool/bandit24/foo
bandit23@bandit:/tmp/wabfs$
```

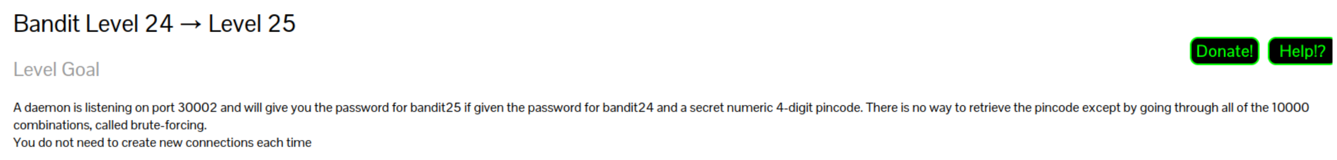
Now lets check our password file and grab our password. All we should have to do is simply cat out the contents of our password file.



```
bandit23@bandit:/tmp/wabfs$ ls
iwantyourcreds.sh  password
bandit23@bandit:/tmp/wabfs$ cat password
VAfGXJ1PBSsPSnvsjI8p759leLZ9GGar
bandit23@bandit:/tmp/wabfs$
```

Success!

Level 24:

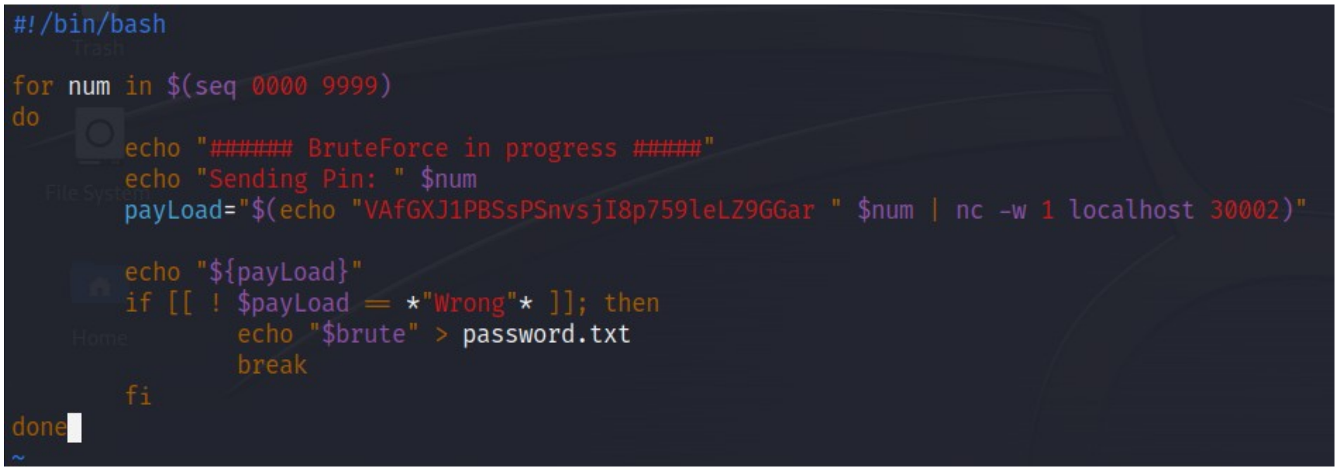


Bandit Level 24 → Level 25

Level Goal

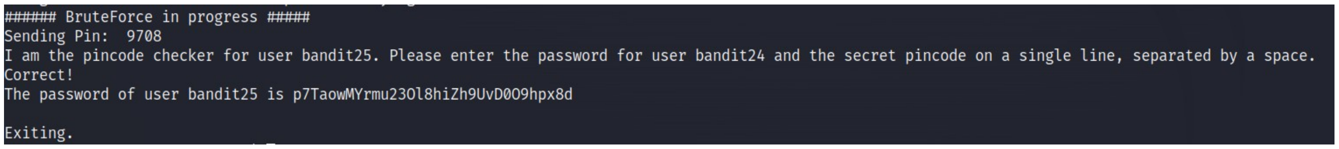
A daemon is listening on port 30002 and will give you the password for bandit25 if given the password for bandit24 and a secret numeric 4-digit pincode. There is no way to retrieve the pincode except by going through all of the 10000 combinations, called brute-forcing.
You do not need to create new connections each time

In this level we are going to create a basic script using bash that will send the password for bandit24 along with a pin code ranging from 0000 – 9999. We need to send this to port 30002. Lets put together a little bruteforcer to accomplish this. We will make our connection utilizing the nc command in our script.



```
#!/bin/bash

for num in $(seq 0000 9999)
do
        echo "###### BruteForce in progress #####"
        echo "Sending Pin: " $num
        payLoad="$(echo "VAfGXJ1PBSsPSnvsjI8p759leLZ9GGar " $num | nc -w 1 localhost 30002)"

        echo "${payLoad}"
        if [[ ! $payLoad == *"Wrong"* ]]; then
                echo "$brute" > password.txt
                break
        fi
done
~
```

Success!



```
###### BruteForce in progress #####
Sending Pin:  9708
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 p7TaowMYrmu23Ol8hiZh9UvD0O9hpx8d

Exiting.
```