Bandit – Stage by Stage explanation

Level 0

All you have to do here is ssh onto the bandit server. This can be done with the command `$ ssh -p 2220 bandit0@bandit.labs.overthewire.org` and then entering the password `bandit0`

Level 1

The password for the next level is stored in a file on the desktop named 'readme'.

This can be revealed with the `ls` command, and read with `cat readme`.

The password for bandit1 is `boJ9jbbUNNfktd78OOpsqOltutMc3MY1`

Level 2

The password here is stored in a file called '-'

This can be revealed with the `ls` command, but running `cat -` confuses the system, as `-` normally denotes a flag. We can get around this by using the complete file path, './-'. This gives us the command `cat ./-`

The password for bandit2 is `CV1DtqXWVFXTvM2F0k09SHz0YwRINYA9`

Level 3

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

This cannot be accessed by simply running `cat spaces in this file name`, as the system will register this as four separate files. Instead, we can either use the escape character, ", in front of the spaces, or specify the file name in quotation marks. This gives us either `cat spaces\ in\ this\ filename` or `cat "spaces in this file name"`

The password for bandit3 is `UmHadQclWmgdLOKQ3YNgjWxGoRMb5luK`

Level 4

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

We can change into this directory using `cd inhere`, and then reveal the hidden file, which is called '.hidden'. We can then read the contents of this file with the command `cat .hidden`.

The password for bandit4 is `pIwrPrtPN36QITSp3EQaw936yaFoFgAB`

Level 5

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.

Again, we can change to the inhere directory using `cd inhere`. Running `ls` then reveals that there are ten files in this directory. As we know that the password is contained in a human readable file, we need to somehow filter the files by filetype. This can be done with the command `find . -type f | xargs file`. This works by finding all the files in the current directory, and passing these as an argument into the file command, which then lists them by type. Doing so reveals to us that 9 of the 10 files are of type 'data', but one is of type 'ASCII text'. Therefore, the password must be in this file.

This works given the relatively small number of files in this directory, but what if it were larger? We could extend this command by piping the output into `grep text`, giving us a complete command of

```
find . -type f | xargs file | grep text
```

This would return only the file marked as ASCII text, omitting all others.

The password for bandit5 is koReBOKuIDDepwhWk7jZC0RTdopnAYKh

Level 6

In this level, we are given three properties of the file containing the password. It is human readable, 1033 bytes and not executable.

Therefore, we need to somehow string these conditions together into a search command. We can again start with `find . -type f`, which is going to recursively find all files of in this directory and subdirectories. However, this gives an unusably long output. At this point, one might think to apply `| xargs file | grep text`, as in the previous level.

This was tried, but still presented the issue of giving an unworkably large number of files. For that reason, I changed the command used, and introduced the `-exec` flag to my initial find statement. This allows us to also run ls with the appropriate flags (to list more information on the file, including file size and whether it is executable). The flags I used were -la; l is the alias of long listing, giving more information, and a is the alias for all, meaning even hidden files are included. From here, I just needed to add a filter so that only files 1033 bytes long were shown - this is done using the `grep 1033` command. By piping these together, we are left with

```
find . type -f -exec ls -la {} \; | grep 1033
```

The password for bandit6 is DXjZPULLxYr17uwoI01bNLQbtFemEgo7

Level 7

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

Immediately, we know that we wish to search the entire file system. This can be done with `find /`. As before, we know that we want to look for all files, so we can introduce the flag `-type f`. We can then make use of the `-user`, `-group` and `-size` flags for the find command, giving us:

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

This works, but we get a slew of permission denied errors. This makes it difficult to identify where the correct file is. We want to somehow suppress these - this can be done by introducing `2>/dev/null` to the command, giving us a final command of

```
$ find / -user bandit7 -group bandit6 -type f -size 33c 2>/dev/null
```

The bandit7 password is `HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs`

Level 8

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

As we know that the password is next to the word millionth, we want to return all lines in the file that contain the world 'millionth'. This can be done by piping the contents of the file (using the `cat` command) into grep, as we have seen before. This gives a command of

```
cat data.txt | grep millionth
```

There is only one line containing the word millionth, so we immediately find our password. It is `cvX2JJa4CFALtqS87jk27qwqGhBM9plV`

Level 9

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

Immediately, by first thought was to use the uniq command, with the -u flag - this should filter a document and only return lines that occur once. As such, I ran

```
cat data.txt | uniq -u
```

However, this caused the entire file to be printed. On reading the uniq man page, I discovered that uniq only compares adjacent lines, and that I therefore needed to sort the file first. A working solution then is

```
cat data.txt | sort | uniq -q
```

The password to bandit 9 is `UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhR`

Level 10

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.

As we are told that the password is 'one of the few human-readable strings' we can use the command `strings`, which will return all human-readable strings in the document. We also know that the password is preceded by a string of equals signs, we can pipe the output of this command into a grep search again. This reveals the password. The final command was

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

This reveals that the password to bandit 10 is `truKLdjsbJ5g7yyJ2X2R0o3a5HQJFuLk`

Level 11

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

In order to be able to read the contents of the file, we therefore need to decode it. This can be done with the command `base64 -d data.txt`. Running this reveals the password.

The password to bandit 11 is `IFukwKGsFW8MOq3IRFqrxE1hxTNEbUPR`

Level 12

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

To decode this message, we need to shift every letter back by 13 places. This can be done using the `tr` (transliterate) command, and an appropriate filter. Again, we can use `cat data.txt |` to pipe the file contents in. Our final command is

```
$ cat data.txt | tr [n-za-mN-ZA-M] [a-zA-Z]
```

The password for bandit 12 is `5Te8Y4drgCRfCx8ugdwuEX8KFC6k2EUu`

Level 13

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!)

I started off here by following their suggestion, and creating / navigating to /tmp/lloyd. As we know that data.txt is a hexdump, the first thing to do is apply a reverse hexdump to this file. This is done using the following command

```
xxd -r > data
```

This pipes the output of the reverse hexdump command into a new file, 'data'. We know that this file has been compressed, but we don't know how. We can find this out by running `file data`. The output of this tells us that 'data' contains gzip compressed data. Therefore, we want to decompress it using gzip. Simply running `gzip -d data` throws an error, as it complanes that the suffix is unknown, so we can instead use

```
zcat -d data > data2.bin
```

This decompresses the file and stores the output in data2.bin, which was the original file name. Again, we can run the `file` command to find out how this has been compressed. This informs us that the file was compressed with bzip2. We can decompress it with

```
bzip2 -d data2.bin
```

Running this decompresses the output into a new file called 'data2.bin.out' but this is not a particularly nice name, so I renamed it to 'data3.bin' using the `mv` command. Again, I checked how it can been compressed using `file`.

These steps were repeated several times, before I eventually reached the original ASCII text file containing the password.

Password to bandit 13 is `8ZjyCRiBWFYkneahHwxCv3wb2a1ORpYL`

Level 14

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

Here, we are given bandit 14's private ssh key, which will allow us to log in as them. This is contained in a file called sshkey.private, in bandit 13's home directory. To pass a key in, we use the `-i` flag with our ssh command, like so:

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

This logs us in as bandit 14. From here, we can navigate to the directory given to us and simply read the password.

The password to bandit 14 is `4wcYUJFw0k0XLShlDzztnTBHiqxU3b3e`

Level 15

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

The first thing we need to do is open a connection to localhost, on port 30000. This can be done with the command

```
nc localhost 30000
```

Once our connection is opened, we simply need to input the password to bandit 14, and the server gives back the password to bandit 15. This is `BfMYroe26WYalil77FoDi9qh59eK5xNr`

Level 16

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.

Here we need to follow a similar process to on the previous level, but here we need to use an ssl connection ratherthan simple TCP. This can be done with the following command:

```
openssl s_client -connect localhost:30001
```

As before, we then simply input the previous password to get the next one. The password for bandit 16 is `cluFn7wTiGryunymYOu4RcffSxQluehd`

Level 17

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.

The first thing that we need to do here is to find out which ports are open. This can be done using netcat, see below

```
nc -zv localhost 3100-3200
```

This tells us that there are five open ports. Using `openssl s_client` as above, we can test to see which ports are accepting ssl. When we find the correct port, we get an sshkey in response - this can be used to log into bandit17.

Level 18

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

All we need to do here is find the difference between two files. Fortunately, there's a command for that – `diff`. Running the command `diff passwords.old passwords.new` will identify which line has been changed, and give us the final password (as below). ueksS7Ubh8G3DCwVzrTd8rAVOwq3M5x The password to Bandit 18 is `kfBf3eYk5BPBRzwjqutbbfE887SVc5Yd`

Level 19

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.

As the guidance said, simply logging in with SSH here was no good, because whilst we could get onto the system we are immediately kicked off. However, as we also know that the password is stored in a file called readme in the home directory, I realised that I didn't need to log in and move around - I could simply take the file directly. This I did using the `scp` command, to copy a remote file to my desktop. This gave me the following command :

```
scp -P 2220 bandit18@bandit.labs.overthewire.org:readme ./tmp
```

Executing this (and entering the bandit18 password) logged into the server and copied the password file into the tmp folder in my working directory.

The password to bandit 19 is `IueksS7Ubh8G3DCwVzrTd8rAVOwq3M5x`

Level 20

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.

On logging in, we are met with an executable file called bandit20-do. The first thing I did was execute it (with `./bandit20-do`). This then told me that it would let me run a command as another user. As I knew that the password was in a file called bandit20, at path `/etc/bandit_pass/bandit20`, I ran the command

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

This revealed that the password for user bandit20 is `GbKksEFF4yrVs6il55v6gwY5aVje5f0j`

Level 21

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).

As before, the first thing I did upon finding an executable was to run it and see what happened (using `./suconnect`). This told me that I needed to give it a port number to connect to. As my initial thoughts were that there was a process running on the server that I would need to communicate with, I did a quick port scan with netcat (`nc -zv localhost 1-20000`). However, upon running suconnect, I realised that this was not going on - it was reading from the port it was given. For that reason, I realised I needed to set up a simple web server, something I again decided to do with netcat. Setting up a server can be done with `nc -l localhost -p 20133`. However, we also need this server to serve up some data (bandit20's password). This can be done with the following command

```
echo GbKksEFF4yrVs6il55v6gwY5aVje5f0j | nc -l localhost -p 20133
&
```

`Echo` simply repeats the text, the pipe operator means this is passed into our server and the single `&` operator means that this process will execute in the background. Once this has been set up, we can simply run `./suconnect 20133` to connect to this. At this point, it succesfully reads the password, and serves up the next one.

The password to bandit 21 is `gE269g2h3mw3pwgrj0Ha9Uoqen1c9DGr`

Level 22

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.

Navigating into the relevant folder, and then examining cronjob_bandit22 using `cat` we can see that it is executing the bash script, stored at `/usr/bin/cronjob_bandit22.sh`. Inspecting this, again using `cat`, reveals to us that the script is copying the contents of the password file to a hidden file, `/tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv`. Reading this gives us the password.

The password for bandit 22 is `Yk7owGAcWjwMVRwrTesJEwB7WVOiILLI`

Level 23

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.

Here, as before, we first need to navigate to our cron.d file, and read the relevant files. Having a look in cronjob_bandit23, we are once again directed to a bash script, as below:

```
#!/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 whoami command returns the username - in this case that is going to be bandit23. The variable 'mytarget' is assigned to the output of `(echo I am user $myname | md5sum | cut -d ' ' -f 1)`. We can therefore find the value of mytarget by running this command ourselves, but putting in bandit23 as $myname. This gives us the address 8ca319486bfbbc3663ea0fbe81326349. As we know that the user's password is being saved here, we can just check out the contents of this to get the password.

The password for bandit23 is `jc1udXuA1tiHqjIsL8yaapX5XIAI6i0n`

Level 24

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.

This level starts in the same way as those before. By finding the bash script being executed by cron, we get this:

```
#!/bin/bash

myname=$(whoami)

cd /var/spool/$myname

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

From this, we know that we need to write a bash script that can be executed and will reveal the password.In order to write this, I first created a directory I could work in – /tmp/lloyd. I then wrote the following bash script:

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

This I saved as script.sh, set as executable by all users (using chmod 777), moved it to /var/spool/bandit 24 and waited for cron to work it's magic. After a couple of minutes, the password for bandit24 was in my conveniently named pass file, and could be read with a simple cat command.

The password for bandit 24 is `UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ`

Level 25

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.

Initially, I was tempted to use python to write the brute forcing script, as this is a scripting language I am particularly fond of, and most linux distributions have it installed by default. However, I realised that in some real world situations, this would not be an option, and I am working through this wargame to improve my own skills. For that reason, I decided to write a bash script instead.

As we know that we need to try all ~ 10 000 combinations, we need some kind of script. I created a script called `bruteforce.sh` with the command `touch bruteforce.sh` and then opened it with vim, a text editor.

I spent a period of time trying to write a bash script that would open a connection and then send the pincode + password combo. This is broadly what I would have done in python. However, I ran into some difficulty with this, as I kept getting problems with timeouts and the volume of processes I was trying to start. For that reason, I changed strategy, opting instead to save all the passwords into a list, and then dump this into the connection. One iteration of my first iteration was like this:

```
 1 #!/bin/bash
 2 password="UoMYTrfrBFHyQXmg6gzctqAwOmw1IohZ"
 3
 4 for a in `seq 0 9`; do
 5     for b in `seq 0 9`; do
 6         for c in `seq 0 9`; do
 7             for d in `seq 0 9`; do
 8                 code="$a$b$c$d"
 9                 echo $password\ $code | nc localhost 30002 &* ^C
10             done
11         done
12     done
13 done
```

However, this was no good, so I replaced line 9 with the following: `echo $password\ $code >> passwords.tx`. This created a list of possible combinations. This can then be passed into the listener using `cat passwords.txt | nc localhost 30002`. This revealed the password.

The password of user bandit25 is `uNG9O58gUE7snukf3bvZ0rxhtnjzSGzG`

Level 26

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.

We need to find out what shell user bandit26 is using; this information is likely to be stored in the /etc/passwd file. So, we take a look in here with `cat /etc/passwd | grep bandit26`. This command gives us the output:

```
bandit26:x:11026:11026:bandit level 26:/home/bandit26:/usr/bin/showtext
```

A quick google search tells us that the final item in this reading is the shell being used. So, we have a look at that with `cat /usr/bin/showtext`. This gives us the following script:

```
#!/bin/sh

export TERM=linux
```

```
more ~/text.txt
exit 0
```

Checking permissions with `chmod` tells us that we cannot edit this file - this was my first thought. Similarly, there was no mileage in editing the /etc/passwd file - this also had the correct permissions enabled. In the end, I was seriously stuck, so ended up googling for a hint - a quick search pointed me to the fact that the way to prevent the script from fully executing was to trigger the `more` command in said script. This can be done by making the terminal window as small as possible. How does this help? Well, from here we can enter vim (by pressing v), and then once in vim we have all the handy commands that come with this - including, crucially, the ability to open a different file. In this instance, the file we want to open is `/etc/bandit_pass/bandit26`. This can be opened with:

```
:e /etc/bandit_pass/bandit26
```

The password for bandit 26 is: `5czgV9L3Xx8JPOyRbXh6lQbmIOWvPT6Z`

Level 27