Bandit Labs Guide:

Overview:

This document is intended to be a comprehensive guide to / documentation of my time working on Bandit Labs, from OverTheWire.org. The goal is to document my thought process while working through each level, while also being able to possibly guide others who may need a hint or some help on these challenges. I will be working through each level and documenting my process. I will not reveal the password to each level because it literally serves no purpose to do so due to the fact that just inputting passwords on each level doesn't teach you anything. For each level it is very important to read the level goals before starting each level. For every level it's very important you read the level goal, the commands you may find helpful, and if applicable the suggested reading, or else sometimes you may have no idea how to attack the level.

Note Before Starting:

This document is meant to be used as a guide, as in if you get stuck you can look through my process and maybe get a hint on how to solve it. Granted this document does give the entire process of each level, but the way it's intended to be used is as hints. If you really can't figure out a level, as long as you really gave it your best shot, then it's fine to look at the answer.

Helpful tips for solving each level:

- Most commands in the terminal will have a page in the manual where you can find help, you can access this manual by running: `man [name of command]` and if there isn't a manual page, there will probably be a command option that you can run, such as -h or --help.
- Try everything, look through manual pages, look through web pages about terminal commands, read articles, read documentation. LEARN SOMETHING NEW! That is the important takeaway in these challenges, is to learn something new, and to challenge yourself. A lot of these challenges is to learn how the commands work and the different flags available to each command.

How to log in:

Here I will show how to log into the game using the unix SSH command:
`ssh [username]@[link] -p[port number]`

The username for every level going forward will be constructed as follows: Bandit# where the number sign is the level number you are trying to access.

So for this level it will be:

```
ssh bandit0@bandit.labs.overthewire.org -p2220
```

When prompted for a password, the password will be bandit0 which is the initial password to log into the game. Now you are into the first level.

Bandit level 0 → 1:

For this level I used the `ls` command to be able to view the contents of the current directory I was in, and found a file titled "readme". Then to read it I used the command strings which will print the strings of printable characters in a file: `strings readme.` The password is contained within this file.

Bandit level 1 → 2:

For this level I used the ls command to once again view the files in the current working directory, then noticed there is one file that is named "-". There is however a problem with that. Usually when you use a dash, it means that any command you use will pull from standard input (standard input is input from the terminal). Therefore we have to use '`./`' before the name of the file. This works because it's as if you are giving the command the path of a file (`./` just means whatever working directory you are in). Then we can do the command: `strings ./-` and it should output the contents of the file -. The password is contained in this file.

Bandit level 2 → 3:

For this level using the ls command shows a file named: "spaces in this file name". This file name poses a problem, because when using a command, you select individual options for the command by placing spaces between them. This means that it would read the file name as individual arguments for the command, not just as one argument, so you have to use an escape character for the space character. The escape character is '\' meaning you would put a '\' before the space. So when you try to read the file you would input the command as '`strings spaces\ in\ this\ filename`'. The password for the next level is contained in the file.

Bandit level 3 → 4:

Upon loading this level, using the ls command shows that there is a singular folder in this directory called 'inhere'. To enter that folder we will use the cd command to change directories

(C D as in change directory), using the command '`cd inhere`'. Upon entry to the folder we can try to use ls again to list the files in the directory, but if we do we will notice that there are no files in this directory. So they must be hidden. We can reveal hidden files by using the find command. When using find we will see that there is a file named `./.hidden` which could not be seen from using the ls command. But now we know that it exists, so we can use the strings command on the file `./.hidden`. The password for the next level is contained in this file.

Bandit level 4 → 5:

The level goal states: "The password for the next level is stored in the only human-readable file in the inhere directory." Now that we know the criteria of the file we are looking for, we can now proceed into the inhere directory and run the file command on the seven files in this directory. We could try brute forcing and running the strings command on every file and finding which file has ascii text, but that's stupid, and I don't like stupid. Instead, let's just run one command which will show us what type of every file in the inhere directory. To do this we are going to make use of the file command. We are going to use the following command: '`file ./*`'. What this command actually means is to run the file command on the file * in the current directory (. indicates current directory), but what star actually stands for is every filename. * is what is known as the wildcard character, so file will search for files that are like the pattern we describe, and since we gave the pattern *, the terminal interprets that as any pattern. Once we run this command the terminal outputs a list of all the file names and their relative types, and we will see that one of the files is an ascii text file, therefore we can run `strings` on that file and get our password for this level.

Bandit level 5 → 6:

The level goal states: "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

Given this information we can now move on to figuring out how to gain access to our password. Moving into the `inhere` directory, we realize that there are 20 different directories, and just checking in the `maybehere00` directory shows 7 different files, so assuming that there are 7 files in each of the 20 directories, that means that in total there are about 140 files… far too many to look through individually. So let's use some terminal magic. We know that the file we are looking for has the three requirements above, so let's use that information combined with the find command to help us. After reading the manual, I found two flags which will help us in this

situation, I found one flag which allows us to define a size, and another which allows us to say we only want to find executable files. Size helps us in the obvious way that we can define the specific size we are looking for, but the way the executable flag helps us may not be obvious. In the terminal, there is a way to take the negation of something you are looking for by using the negation operator: `!`. By using the negation operator on the executable flag, we can actually get all files which are not executable. Running the command from the `inhere` directory, we will run the following: `find -size 1033c ! -executable` which reveals the path of the file the password is contained in, then after that we can run strings on that file path to reveal the password.

Bandit level 6 → 7:

The level goal states: "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

The key words here for this level are in bold, **somewhere on the server**. This gives us a big hint, that we need to search for this specific file from the root directory because it could be anywhere. Another perusal of the manual page for the find command shows us that there are two useful very useful flags for this situation, `-user`, and `-group` which allow us to search only for files which belong to that user or group respectively. Also since running a version of the command below shows a bunch of files that we don't have permission for, I decided to route all of the error messages to the null device (known as /dev/null) by using the > operator. Typically > redirects any standard output or stdout to a file, but there are variations such as 2> which redirects the output of standard error, or stderr to a file. So by adding `2>/dev/null` to the end of our command, we can redirect all those error messages to the null device leaving us with a nice single file path to the password file. The following command will reveal the path of the password file: `find / -size 33c -user bandit7 -group bandit6 2>/dev/null`

Bandit level 7 → 8:

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

The key command to use for this challenge is going to be `grep`. The grep command searches the named input file(s) for a given pattern. In this case, the level goal told us what the pattern was, its next to the word millionth. Running the following command will reveal the password:

```
grep millionth data.txt
```

Bandit level 8 → 9:

The level goal states: “The password for the next level is stored in the file data.txt and is the only line of text that occurs only once”

The section where they suggest commands to solve this level show the two important commands to be able to solve this: sort and uniq. Sort does what you would imagine it does, it creates a sorted concatenation of all file(s) to standard output. Uniq, filters adjacent matching lines from an input, and writes it to the standard output, it also has a flag -u which allows us to only print unique lines. Using sort, we can get all the repeated lines next to each other, then pipe that output into uniq as input, this way the uniq command can filter out all the garbage and we will be left with our password. The following command when run will reveal the password:
`strings data.txt | sort | uniq -u`

Coming back to this I realize I may not have made it clear what I was actually doing when I said “pipe that output as input.” the ‘|’ character will take data from stdout and use it as stdin for the next command, for example, strings prints the data.txt contents to stdout, but then we use the | operator to take that output and use it as input for sort.

Bandit level 9 → 10:

The level goal tells us that the password is one of the few human-readable strings, and that it is preceded by several ‘=’ characters. In my original attempt I only used grep on the file to try and get it to print the lines with that pattern, however because grep realizes that there are strings in this file that are not human readable, it just tells us that there are matches in the file data.txt. We can circumvent this problem by passing the data is as text through stdin. When we use strings on the binary in the file data.txt, it will just turn the binary information into random garbage and crap, however, the strings that we want will still remain intact. Therefore we can take the output of our strings command, pipe it into grep, search for the pattern and like magic our password will appear. The following command will reveal the password: `strings data.txt | grep ===`

Bandit level 10 → 11:

The level goal tells us that the password is located in the file data.txt, however it is encoded using base64 encoding. Good news for us, linux has a base64 command which is pretty good at

decoding base64 data, as long as we run it with the decode flag. A quick run of the decode base64 command on the text file will reveal our password. The following command will reveal the password: `base64 -d data.txt`

Bandit level 11 → 12:

The hint tells us that the password is in data.txt but all the letters have been rotated by 13 positions. They also show that there is some helpful reading material on Rot13 at https://en.wikipedia.org/wiki/ROT13, which I highly suggest you at the very least skim through (it's a wikipedia page, it's got good information, just read it). We don't even need to use the terminal to solve this problem, we can use a helpful Rot13 translator on the internet, at least that's what I did because it is much easier than using some roundabout terminal commands. I just went to https://rot13.com/ and plugged in what was in data.txt, and it returned to me the password.

Bandit level 12 →13:

The hint for this level informs us that data.txt is actually a hex dump of data which was compressed multiple times. (more info on hexdumps at https://en.wikipedia.org/wiki/Hex_dump). Knowing that is actually very helpful because it can help point us in the right direction for the command we need to use. We can't read any of the information as a hex dump, so we need to reverse the hex dump. For this we will use the `xxd` command. This command can either make or reverse hex dumps, and we will take this reverse hex dump and store it in a temporary directory as they suggest in the level goal (`mkdir /tmp/yourname123`). Using the command:
`xxd -r data.txt /tmp/yourname123/myfile1` you will end up with a file named data in the directory /tmp/yourname123/ and from there we can move on to figure out how to uncompress it. Lets also cd to that directory /tmp/yourname123 so that we can just work in there. If we use the file command on our new data file, it will read out: "gzip compressed data, was 'data2.bin', last modified…" The important part of what it reads out is the part when it tells us that the data is gzip compressed. Now that we know the type of compression on this file, we can use certain steps to decompress it.

The Decompression Process:

When decompressing the file, we know that there are several layers of compression, so that being said we don't know how many times we have to decompress each type of compression. Reading the manual for the different types of compressions that we will see (gzip, bzip2, and tar) show us that there are three commands that decompress both types of compression respectively: zcat, bzcat,

and tar (with some special options selected). NOTE: zcat, bzcat and the reverse tar command all write data to stdout, so we use the > operator to direct the output into a file that we can work with.

- So as said before we know that the data file we most recently created was gzip compressed data, so to decompress this layer, we will use the `zcat` command: `zcat myfile1 > myfile2`
- So now we can run the file command on data2 which will reveal the next type of compression that was used to create this file, and we will learn it was another layer of gzip compression. So we once again run the command:
  `zcat myfile2 > myfile3`
- Once again running the file command on myfile3, its discovered that the type of compression used for this layer was bzip2. Therefore the command we run to revert it is:
  `bzcat myfile3 > myfile4`
- Running file on myfile4 shows that it is once again gzip compressed data, so lets run zcat again:
  `zcat myfile4 > myfile5`
- Running file on myfile5 shows that POSIX tar archive was used to archive this data. Archivers are things such as winrar or 7-zip, these are also used to compress data. For this we will use the tar command with the flags x, v, and f. The options and their purposes are: x is used to extract an archive, f is used to specify name of the tar archive and v is used for more detailed listing of the contents.
  `tar -xvf myfile4`
- When we run the tar command it shows us that a new file data5.bin has been created. Then running file on data5.bin reveals it is another tar archive:
  `tar -xvf data5.bin`
- Then running file on data6.bin which was created by the previous tar command, it shows it is bzip2 compressed data:
  `bzcat data6.bin > myfile7`
- myfile7 will turn out to be a tar archive:
  `tar -xvf myfile7`
- The previous tar command will make a file data8.bin which was compressed using zcat compression:
  `Zcat data8.bin > myfile9`
- myfile9 when the file is run on it is revealed to be ascii text. We've finally done it.

Now that we got past all of the layers of compression, we can now just run strings on myfile9 to reveal the password.

~ Networking Interlude ~
Going forward, these next couple of levels will require some networking knowledge, but as we already know, bandit labs provide a metric ton of great reading material on various topics. It is highly suggested that you at least read/watch some of the material they suggest in these next levels.
NOTE: It is highly suggested that you read the helpful reading material here, as I am very new to networking and probably won't be able to explain everything with extreme clarity.

Bandit level 13 → 14:

The level goal states something pretty unique we haven't seen thus far: "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 currently working on."

So we know that we don't need a password, and they tell us that we are provided with a private ssh key. So we have to use the key that we were given (we can see the key sshkey.private if we just run the ls command to view the contents of the directory). If we check the manual on the ssh command, there is an option -i which will allow us to pick a file to use as an "identity" i.e. it will allow us to choose a private key to log in with. Run the ssh command with the -i flag and when you choose the machine to login to, you can choose the machine you're working i.e. localhost:

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

This will log us into bandit 14, and we can now access the password for this level by grabbing the password from the directory mentioned in the level goal. Keep this password in mind as it will be important for the next level.

Bandit level 14 → 15:

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

For this level, I had to do a lot of manual reading, and when I say a lot I mean A LOT. So for this one I will describe the process of what I had done because I think it was a good learning experience. The level goal stated it wanted me to submit a password to the localhost on a certain port, so I knew that ssh wouldn't help here because I didn't have a password or ssh key to connect to anything. So I read some manuals and I found something interesting in the telnet command, it says it is for interactive communication with another host. So I figured that with telnet, I may be able to submit a request to the localhost machine on the certain port, and get back what I was looking for. So I was able to connect using the TELNET protocol and then send a "request" in the form of sending the password for level 14. When I did, I was returned the password for level 15.

```
telnet localhost 30000 → after inputting this just send the password for level 14.
```

Bandit level 15 → 16:

The level goal states that: "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."

So I thought that maybe I should use the telnet command to connect to the host on this port again, however when I tried that everytime I tried to submit a request the server booted me from the connection. So I read through the helpful materials and I learned quite a bit. First thing was that I can't just connect using the telnet protocol because telnet is not encrypted. The level goal states we need to use SSL encryption, which is a type of secure connection to a host. So I then decided to take a look at the OpenSSL Cookbook reading, and there I found what I was looking for. Openssl is a toolkit which implements SSL and TLS network protocols, meaning that when it connects it can connect with SSL encryption. Using this tool kit you can then access a command called s_client, which is how you send a request with SSL encryption. So once I figured this out, I could connect to localhost on the port specified, and send my request. After sending my request, I was returned the password for level 16.

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

Bandit level 16 → 17:

The level goal states that: "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 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.

For this I decided to use nmap to give me a map of open ports in the range that they specified:
`nmap -p 31000-32000 localhost`
This nmap revealed 5 different open ports, and I couldn't find any way to get nmap to show me ports which only had ssl enabled, I just tried the password in all 5 ports with s_client and openssl, until I got a response that wasn't what I input from port 31790 (I also used the -quiet flag so that I wouldn't receive unnecessary information from the host). However the response I got was a private key. So with this we now have to save it to a file to be able to use. So let's develop a bash command which will send the password (from /etc/bandit_pass/bandit16) into the s_client and then write the output to a file in the tmp directory where we can store the key:

```
cat /etc/bandit_pass/bandit16 | openssl s_client -connect localhost:31790 -quiet > /tmp/key123/pkey
```

This will store the private key in a file in the tmp directory. However if we try to use this key to log in with ssh it will give us an error due to the fact that every user has read and write access to this file, and claims that the key is not secure. We need to find a way to change the access permissions so that the file only allows read/write access for the owner. That was when I came across the chmod command. It allows you to edit the permissions of a file so that only certain users can access it. Using the command: `chmod 600` I was able to edit the file permissions so that only the owner (bandit16) has read write access, and no one else can access it. Now we can ssh into bandit 17 with this key:

```
ssh -i /tmp/key123/pkey bandit17@localhost
```

Bandit level 17 → 18:

The level goal states that: "There are 2 files in the home directory: 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

For this level we can use the diff command which takes as arguments two files and shows you the differences between them through output to stdout. Any lines preceded by a < is a line from the first file that would need to be added to the second file to make them the same, and any lines preceded by a > are lines from the second file that would need to be added to the first file to make them the same. So we just need to look at this output and see what line would need to be added to passwords.old from passwords.new and that is our password.

```
diff passwords.old passwords.new
```

Take notice of the note in the goal, as it will be important for logging into bandit18

Bandit level 18 → 19:

From the last level, we notice that trying to log into bandit18 with ssh doesn't work. Well it is revealed to us in the level goal that someone has modified the file .bashrc to log us out when we attempt to log in with ssh. So we have to find a way to just get the level to return to us the password without us fully logging in. We could do this by using the -t option of ssh, which allows us to execute arbitrary screen-based programs on a remote machine, this way we don't have to stay logged in to execute our commands:

```
ssh bandit18@bandit.labs.overthewire.org -p2220 -t "strings readme"
```

This will return to us the password for bandit19

Bandit level 19 → 20:

The level goal states: "To gain access to the next level, you should use the setuid binary in the home directory. Execute it without arguments to find out how to use it. The password for this level can be found in the usual place after you have used the setuid binary.

When we look around in the home directory, we find the only file is bandit20-do, which we are told is an executable file. So first we should run it and see what it says because the goal says running it without any arguments will explain how to use it:

```
./bandit20-do
```

After doing this the output tells us that the executable runs the command as another user. Assuming that it will run whatever command we give it as user bandit20 (because it's titled bandit20-do) we could try getting the password for the next level using the elevated permissions that this file gives us:

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

This will return to us the password to level 20.

Bandit level 20 → 21:

For me this was an extremely difficult level. The level goal states that: “There is a setuid binary in the home directory that does the following: it makes a connection to localhost on the port you specify as a command line 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)

This level took a long time for me to spend reading manuals, and asking questions to google. The hint here was a big help, where they say: “try connecting to your own network daemon to see if it works as you think. So I knew that we had to have a port for suconnect to actually connect to so I tried looking for open ports, but there were some open but none of them did as I expected. That was when I looked to google and people had said to use nc with the -l flag to create a port to listen on, and just have the port I make return what suconnect was looking for! So with that, I opened a second terminal window and opened a port with this terminal so that when called it would return the password to level 20:

```
echo <level20 password> | nc -l -p <random port#>
```

Then in my original terminal I ran ./suconnect with the port number that I knew I had a listening port on:

```
./suconnect <number of listening port>
```

When suconnect sent it’s request to the listening port, it received level 20’s password, and in turn sent level 21’s password back to the port. Then on my “server” terminal, I was greeted with level 21’s password.

Bandit level 21 → 22:

The level goal states that: “A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in /etc/cron.d/ for the configuration and see what command is being executed.”

So I decided to just move over to that directory mentioned in the hint and take a look at what was going on. There are several jobs that seem to be in here, but there is one called bandit22 so I took a look in that one. It appeared to be running another job file /usr/bin/cronjob_bandit22.sh to /dev/null (the null machine). So I decided to run strings on the file it was sending to the null machine. The job file was running a chmod 644 command, which after googling it I figured out that the specific chmod code gives only the owner of the file access to write, but gives anyone

access to read. The file was also running a cat command which writes the password of bandit22 into a file in the tmp directory. Running strings on the particular file in the tmp directory returns the password for bandit 22.

Bandit level 22 → 23:

The level goal states: "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.

So the first thing I did again was go straight to the /etc/cron.d directory, and ran strings on the file cronjob_bandit23 which showed it was running a shell script in another directory: /usr/bin/cronjob_bandit23.sh So I ran strings on the shell script to see what commands it was actually running. After looking at the bash script commands, I saw it was running a string with the username of the current level, into a couple of commands to get a target directory. However when I ran this script, it knew my username was bandit22, so I just ran the command myself with bandit23 as the name to get the target directory it was linking too:

```
echo I am user bandit23 | md5sum | cut -d ‘ ‘ -f 1
```

Then with this I had the name of the target in the tmp directory. Running strings on the directory /tmp/<target gained from command above> yielded the password for bandit23.

Bandit level 23 → 24:

The level goal states: "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!
NOTE2: Keep in mind that your shell script is removed once executed, so you may want to keep it around.

The level goal for this one makes it quite clear what we have to do, we must write a bash script that will return to us the password for bandit24. We also know from looking at the scripts in /etc/cron.d/ and from the hints that there are certain commands being run at regular intervals.

From reading the script /usr/bin/cronjob_bandit24.sh there is a line that says it executes every script in /var/spool/bandit24/ so we just have to inject our script into this folder. For this I used a temporary directory in tmp and used nano to edit a bash script (I suggest you use nano a little bit to become familiar with it, it's just a text editor that comes in the bash). So the bash script I wrote was:

```
cat /etc/bandit_pass/bandit24 > /tmp/<tempdirectory name>/password.txt
```

Then after that I saved the file as test.sh, I also wanted to make sure everything worked so I gave the script permissions using chmod 777 test.sh, I then made the file password.txt using a new command I found called touch (`touch password.txt → it will make the file`) and also gave the password.txt file the same permissions as test.sh using chmod. Then I copied the bash script I wrote into the /var/spool/bandit24/ directory so that it would get run. After waiting for a couple seconds I ran strings on password.txt and was delighted to see the password was in the file.

Bandit level 24 → 25:

The level goal states: "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 pin code. There is no way to retrieve the pin code except by going through all of the 10000 combinations, called brute-forcing."

So the level asks us to just pass through all possible passwords to the daemon (which is really just a background process) so let's just do that! Though brute forcing isn't my favorite process, it may be necessary in some cases. I wrote a small bash script that when run, would generate a list of all the possible 4 digit pin code. I won't be explaining how I wrote this script because I think it is important if you are following this guide to at least attempt writing the bash script before googling it. Then once I had my list of passwords stored in a text file, I passed that into the daemon, by piping the output of the text file when read to an nc command:

```
cat passlist.txt | nc localhost 300002
```

Eventually the right passcode was passed in and we are returned the password for level 25.

Bandit level 25 → 26:

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

The more command was the key here. If we try logging into bandit26 with the ssh key that they give us, it boots us immediately on logging into level 26. By using more, we can make the

terminal show us things sequentially on incoming, and we can make more happen automatically by physically shrinking the size of our terminal window! So I shrunk the size of the window, then logged in with ssh and sure enough more took over and stopped printing too much information than the window could hold. Now I was stuck, I got the terminal to not boot me, but now what. I was just stuck here with a tiny terminal. I couldn't put in commands, and certain keys would just skip through the rest of the output that I was holding from coming through. That's when I found a thread saying that I can run the text editor known as Vim from more by pressing the v key! This was HUGE! Because from Vim, you can run commands, and I have found a little Vim commands cheat sheet which I will link here: https://www.fprintf.net/vimCheatSheet.html. Since I now had a way to potentially run commands, I was ready to acquire the password, because the vim cheat sheet shows that we can use the command :e [dit] {file} to edit other files. PERFECT! This is perfect because we know there is a directory where all bandit passwords are kept, so we can just open that file and find the password!

Bandit level 26 → 27:

The level goal states: "Good job getting a shell! Now hurry and grab the password for bandit27!"

Oh, I see. So logging in to bandit26 with the password we just acquired doesn't actually fix the original problem from the last level of getting booted from logging in after it prints the intro message. Well we should see if there is a way from Vim to acquire access to a linux shell. After browsing through a bunch of documentation I found that not only can you open a shell, you can set the path of the shell. So using the following two commands in the order shown:

```
:set shell=/bin/bash
:shell
```

I was able to open a shell for bandit26. Now we can actually get the password for bandit27! Upon inspecting what is in the home directory, we see there is a binary. If we run it with no options, it tells us it allows us to run commands as other users. So lets just have it read us the password:

```
./bandit27-do strings /etc/bandit_pass/bandit27
```

This will return to us the password.

Bandit level 27 → 28:

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

Clone the repository and find the password for the next level.”
So essentially this level requires us to have some knowledge of Git. Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency (Two good links to read up on git: https://git-scm.com/doc and https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html). After reading up on git commands, I looked at the level goal and searched for commands which would help me achieve my goal. I found the command to be able to clone the repository, and made a directory in the /tmp folder where I could work and I gave it read write permissions using the -m flag and the chmod code 777, then after moving to my temporary directory, I used the clone command to get a copy of the git repository: `git clone ssh://bandit27-git@localhost/home/bandit27-git/repo`
Now we should have our own copy of the repository, and we can see that we now have a folder called repo, which in there is a README file, which contains the password.

Bandit level 28 → 29:

The level goal states: “There is a git repository at ssh://bandit28-git@localhost/home/bandit28-git/repo. The password for the user bandit28-git is the same as for the user bandit28.
Clone the repository and find the password for the next level.”

So just as before, I made a directory in tmp and cloned the repository to there, then moved there to work in that directory. Then after moving into the repo folder, I saw there was a README.md file which is just ASCII text. Running strings on the file shows that the file has credentials but the password field is just a bunch of X’s. Maybe there was once a password here? I used the git command log on the file README.md: `git log README.md` and it shows all the different revisions that were made to the document, one of which states it was a “fix for info leak.” I found that the git log command also has a flag -p which will show the differences introduced in between each commit, and it then shows the password being changed as the differences. From this we can just pull the password.

Bandit level 29 → 30:

The level goal states: “There is a git repository at ssh://bandit29-git@localhost/home/bandit29-git/repo. The password for the user bandit29-git is the same as for the user bandit29.
Clone the repository and find the password for the next level.”

Upon copying the repository into a temporary directory, I was able to look and find that there was a README.md file (if you’re curious about .md files, read more about them here: https://en.wikipedia.org/wiki/Markdown) and that the password field in this file says that there are: “<no passwords in production!>” so that being said we know that the file must exist somewhere else. After researching git, we know that it's a version control system that works off the basis of “branches” and that these branches are each their own version of a product. So from what we read in the README.md file, we can assume that this is sort of a “production” branch. But there is a pretty big possibility in this case that there are other branches! I found that you can list all branches by using: `git branch -a` we can now see that there are actually four different branches in the remote repository, as in when we copied, we only copied one which was master. Since master must be production, I just assumed that dev (short for development) would be the one to have the password, so I knew I wanted to switch to that branch, which I found you could do using the git command `checkout` to checkout a different branch: `git checkout dev`. Now we have in our local files, the dev branch. Now running strings on the README.md file shows us the password!

Bandit level 30 → 31:

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

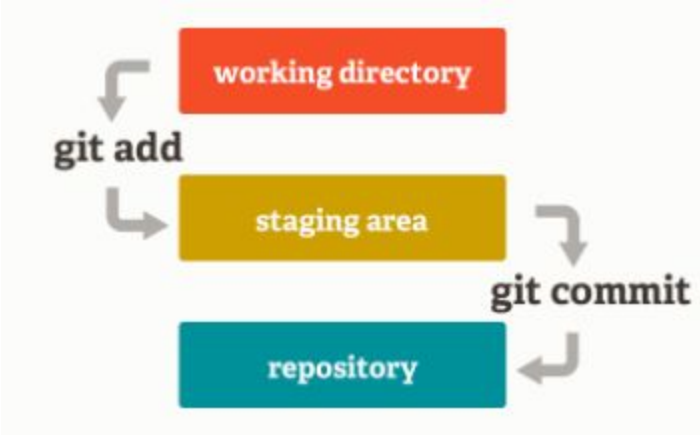
Clone the repository and find the password for the next level.”

Once we clone the repository to our local machine, we see that there is just another README.txt and it is just a text file with some worthless info. So I checked to see if there were other branches but it's just the master branch. So after looking through some stack overflow pages, I found someone said the possibility of “a reference not in the heads namespace” and he suggested trying the following command: `git ls-remote` and upon running that command, we were displayed with not only what looked like a code for the head, but a code for the master branch, and a code for something that I could only assume is a reference called secret. So I then used: `git show /refs/tags/secret` which can be used to show types of git objects (blobs, trees, tags and commits). And that command shows us the password for level 31.

Bandit level 31 → 32:

The level goal states: “There is a git repository at ssh://bandit31-git@localhost/home/bandit31-git/repo. The password for the user bandit31-git is the same as for the user bandit31.
Clone the repository and find the password for the next level.”

So for this level I cloned the git repo and put it in a temporary directory, then reading the README.md file tells us that we need to push forward a file called key.txt which just contains the string: ‘May I come in?’ So from there we needed to push this file to the remote repository, and the following diagram shows how git pushes work:



So first we need to add the file key.txt to the staging area, then once we have the file staged, we can commit it to the repository, using the commands shown in the diagram. So first we must git add: `git add -f key.txt` then we must git commit: `git commit`. Now we have committed our changes to the local repository. We need to push them to the remote repository using: `git push`. Once we do that, we will be returned the password for the next level.

Bandit level 32 → 33:

The level goal states: “After all this git stuff it's time for another escape. Good luck!”

Upon logging in we are greeted with a message saying “WELCOME TO THE UPPERCASE SHELL!” whenever we try to type some command, it is converted to uppercase and is unrecognized by the terminal. I was lost so I just started reading through the sh man page as the level goal page suggests, and I found the command $0 which will let us invoke a shell. Once we do that we can finally input commands! So I was just trying to see if maybe it will let us access level 33’s password just by reading the password file /etc/bandit_pass/bandit33 and it does!

The End:

And with that we have finished the war game Bandit!