Bandit 0 – Accessing the Server

```
└─$ ssh bandit0@bandit.labs.overthewire.org -p 2220
```

The first thing we need to do to complete the Bandit CTF challenges is to use the SSH program to login

Bandit 0 – Accessing the Server

```
└─$ ssh bandit0@bandit.labs.overthewire.org -p 2220
```

This command uses `SSH` to log in as the `bandit0` user to the `bandit.labs.overthewire.org` server on port `2220`

Bandit 0 – Accessing the Server

```
[root@localhost ~]# ssh bandit0@bandit.labs.overthewire.org -p 2220
The authenticity of host '[bandit.labs.overthewire.org]:2220 ([16.16.163.126]:22
20)' can't be established.
ECDSA key fingerprint is SHA256:IJ7FrX0mKSSHTJ63ezxjqtnOE0Hg116Aq+v5mN0+HdE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

If asked if we are sure we want to continue connecting, we need to type `yes`, then press enter

Bandit 0 – Accessing the Server

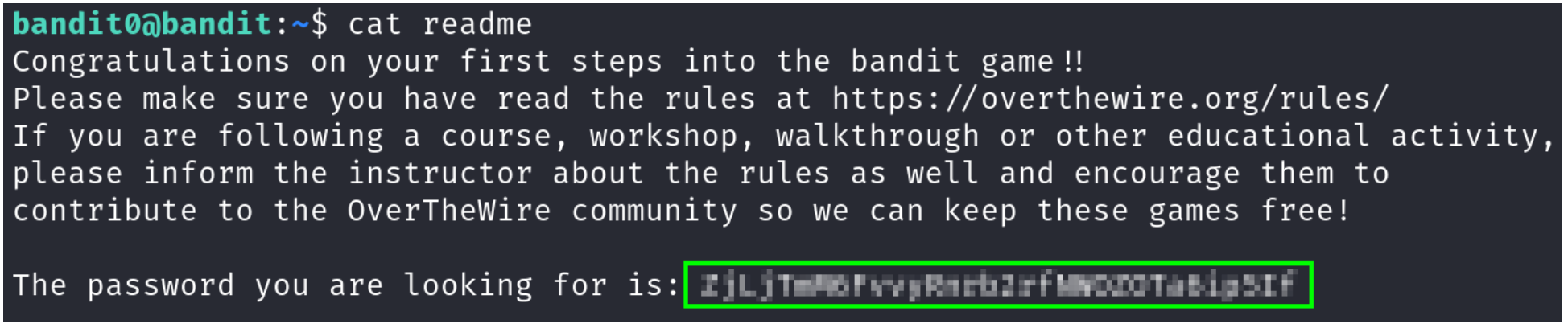
```
bandit0@bandit.labs.overthewire.org's password:
```

Then we'll be asked to enter the user's password. Type in `bandit0`, then press enter. While you're typing in the password, you will not see any feedback from the terminal. This is normal

Bandit 0 – Listing Directory Contents

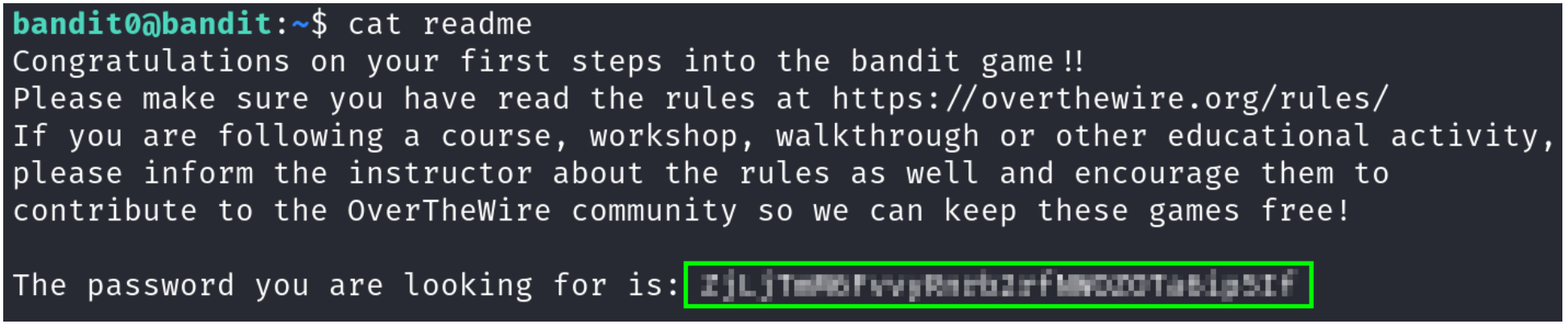
In Linux, the `ls` command is used to list out the contents of a directory. When used here, we see the `readme` file

Bandit 0 – Reading File Contents



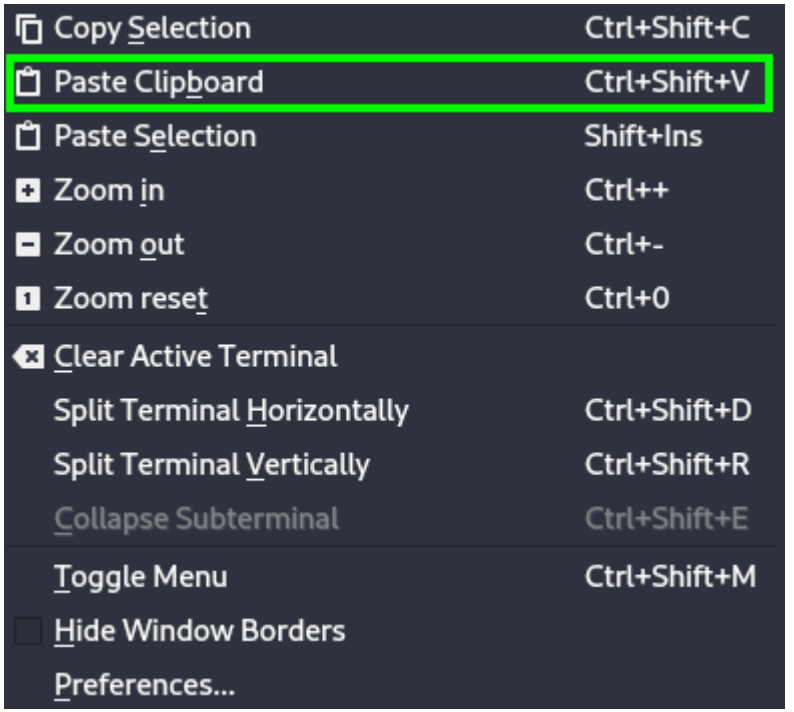
The command to read a file in Linux is the `cat` command, and the syntax is:
`cat <filename>` for example `cat readme`

Bandit 0 – Reading File Contents



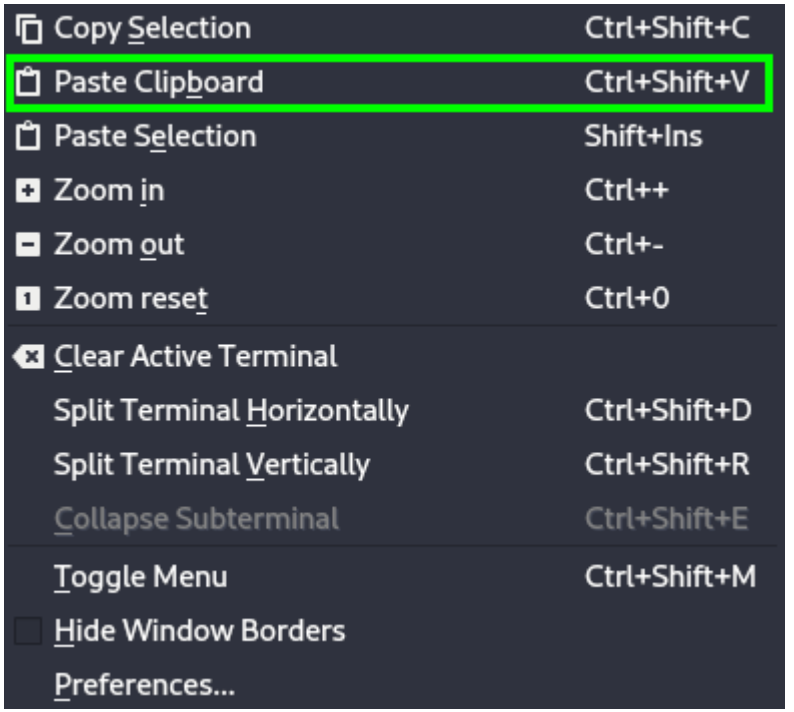
In the contents of the file, we see the password for the next level, which we will use SSH to login as the bandit1 user

Bandit 1 – Pasting in Passwords



When entering in the passwords for the Bandit CTF levels, we should paste in the password instead of keying it in

Bandit 1 – Pasting in Passwords



We can right-click then select `Paste Clipboard` or use the keyboard shortcut `Ctrl+Shift+V`

Bandit 1 – Clearing the Screen

If the screen becomes too cluttered in Linux, we can use the `clear` command to clear the screen and go back up to the top of the screen

Bandit 1 – Reading Files with Special Characters

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

This level requires us to read files with special characters, but if we try to read this file in the regular way, it doesn't work

Bandit 1 – Quitting Unresponsive Programs

If at any time a program becomes unresponsive in Linux, we can use the `Ctrl+C` keyboard shortcut to terminate the program

Bandit 1 – Reading Files with Special Characters

1) Filenames should not include spaces. We can use underscores if we want to use spaces, e.g., `my_file`

2) Filenames should not start with numbers, because certain numbers are treated as special characters in Linux

3) Filenames should never start with special characters

Bandit 1 – Reading Files with Special Characters

We can read files with special characters by referencing the exact directory where the file is. In Linux, the current directory is referenced with `./`

Bandit 1 – Reading Files with Special Characters

So to reference a file named – in the current directory, it would be `./-`

Bandit 2 – Reading Files with Spaces in the Name

```
bandit2@bandit:~$ cat spaces in this filename
cat: spaces: No such file or directory
cat: in: No such file or directory
cat: this: No such file or directory
cat: filename: No such file or directory
```

In this level, we need to read a file with spaces in its name. If we try to read this file normally, we won't be able to, since the Linux interprets the spaces as the end of one file name and the beginning of another

Bandit 2 – Reading Files with Spaces in the Name

```
bandit2@bandit:~$ cat spaces in this filename
cat: spaces: No such file or directory
cat: in: No such file or directory
cat: this: No such file or directory
cat: filename: No such file or directory
```

And this is why its not recommended to put spaces in filenames. However, there's a couple of methods we could use to reference filenames with spaces in them

Bandit 2 – Reading Files with Spaces in the Name

The first method is to wrap the name of the file in quotes, either single quotes or double quotes. This ensures that Linux will interpret everything in the quotes as a single object

Bandit 2 – Reading Files with Spaces in the Name

```
bandit2@bandit:~$ cat spaces\ in\ this\ filename
[redacted]
```

The second method to insert a backslash character before every space in the filename, which lets Linux know that the space is not the start of a new filename, but part of the current filename

Bandit 3 – Changing Directories

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

In Linux, we can move into a directory by using the cd command. The syntax is `cd <directory_name>`, for example, `cd inhere`

Bandit 3 – Checking the Current Directory

If we want to check our current directory, we can use the `pwd` (present working directory) command. In Linux, all directories are start with a `/`, for example, the `/home/bandit3/inhere` directory

Bandit 3 – Hidden Files

```
bandit3@bandit:~/inhere$ ls -la
total 12
drwxr-xr-x 2 root    root    4096 Sep 19  2024 .
drwxr-xr-x 3 root    root    4096 Sep 19  2024 ..
-rw-r----- 1 bandit4 bandit3   33 Sep 19  2024 ...Hiding-From-You
```

In Linux, any file or directory that start with a `.` is a hidden file, which means that it won't appear when using the `ls` command in the regular way.

Bandit 3 – Hidden Files

```
bandit3@bandit:~/inhere$ ls -la
total 12
drwxr-xr-x 2 root    root    4096 Sep 19  2024 .
drwxr-xr-x 3 root    root    4096 Sep 19  2024 ..
-rw-r----- 1 bandit4 bandit3   33 Sep 19  2024 ...Hiding-From-You
```

The current directory in Linux is denoted as `.` and because its name starts with a dot, it is hidden by default. The same goes for the directory above the current one, which is `..`

Bandit 3 – Hidden Files

```
bandit3@bandit:~/inhere$ ls -la
total 12
drwxr-xr-x 2 root    root    4096 Sep 19  2024 .
drwxr-xr-x 3 root    root    4096 Sep 19  2024 ..
-rw-r----- 1 bandit4 bandit3   33 Sep 19  2024 ...Hiding-From-You
```

To see hidden files with the `ls` command, we have to use the command with an argument `-a`, which alters the output of the command by including hidden files in the output

Bandit 4 – Text versus Data Content

```
bandit4@bandit:~/inhere$ cat ./-file00
◆p◆◆&◆y◆,◆(jo◆.at◆:uf◆^◆◆◆@bandit4@bandit:~/inhere$
```

Computer files typically contain one of two types of content, human-readable text, or machine-readable data

Bandit 4 – Text versus Data Content

```
bandit4@bandit:~/inhere$ cat ./-file00
◆p◆◆&◆y◆,◆(jo◆.at◆:uf◆^◆◆◆@bandit4@bandit:~/inhere$
```

Files with data content are meant to be processed by computer software, and will not be readable if read by using the `cat` command

Bandit 4 – Text versus Data Content

```
bandit4@bandit:~/inhere$ ls -l
total 40
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file00
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file01
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file02
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file03
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file04
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file05
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file06
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file07
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file08
-rw-r----- 1 bandit5 bandit4 33 Sep 19  2024 -file09
```

In this level, we're meant to find out which file contains text contents, not binary. It would be tedious to look through the files one by one, but there's a way to scan all of the files at once

Bandit 4 – File Command

```
bandit4@bandit:~/inhere$ file ./-file00
./-file00: data
```

The `file` command in Linux is used to return the type of contents in a file

Bandit 4 – The * Wildcard Character

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

The * special character in Linux is used as a shorthand for “all files”, and we can run a command like the one above to combine the file command with the * wildcard character to run the command on all the files in the directory

Bandit 5 - Find Command

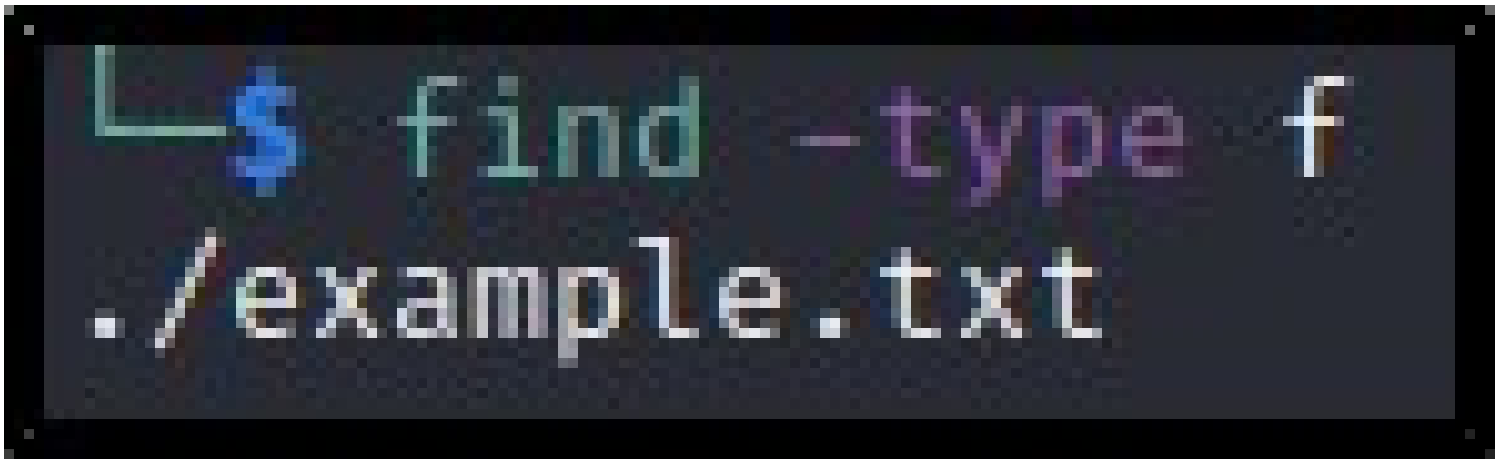
The Find command is used to search for files on the system. It can used with many different arguments and flags to refine its search parameters.

Bandit 5 - Find Command

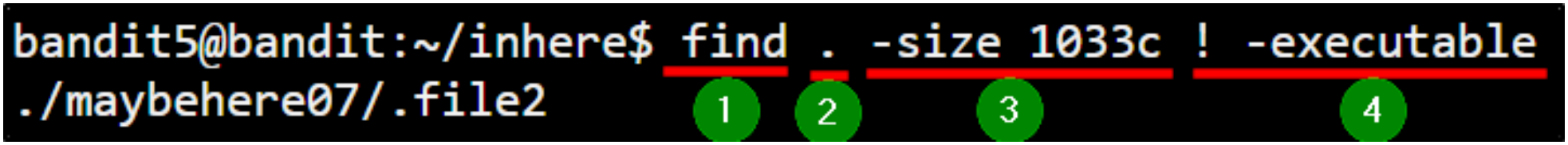
The Find command allows a search of files and / or directories in the file system, and matches files in the output according to the criteria provided by the command arguments.

The argument `-type` searches by file or directories and the argument `-size` searches for files of a particular size.

Bandit 5 - Find Command



Bandit 5 - Find Command



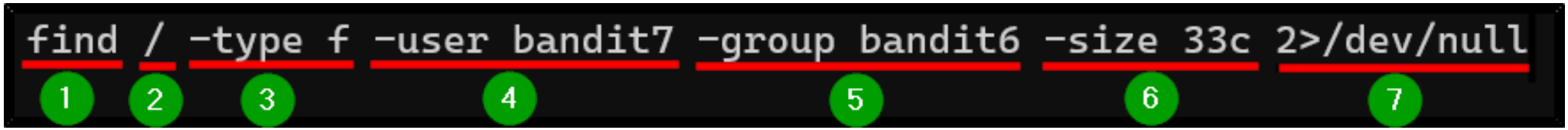
1 – The command itself

2 – The location to be searched

3 – The size of data to be returned

4 – The executable status

Bandit 6 - Find Command



1 – The command itself
2 – The location to be searched
3 – The type of data to be returned, file / directory
4 – The file / directory user ownership
5 – The file / directory group ownership
6 – The file / directory size
7 – Omit error messages from output

Bandit 7 - Grep Command

The Grep command searches within the contents of files for specified strings. It is very commonly used to pick out specific words or phrases.

Bandit 7 - Grep Command

1 – The command itself

2 – The pattern to search for in the file / directory

3 – The file to be searched

Bandit 8 - Sort Command

The Sort command takes all of the lines contained within a given file and returns them in alphabetical / numerical order.

Bandit 8 - Sort Command

1 – The command itself

2 – The input to be sorted

Bandit 8 - Uniq Command

The Uniq command takes all of the lines in a file and removes any lines with identical contents to the one above it. This command is very useful for removing consecutive blank lines in a given file

Bandit 8 - Uniq Command

1 – The command itself

2 – The count flag

3 – The file to be processed

Bandit 8 - Command Piping

```
sort data.txt | uniq -c
```

In Linux, command piping is the process of passing the output of one command into the input of a second command.

Bandit 8 - Command Piping

```
sort data.txt | uniq -c
```

This is a very useful feature, because it allows commands to be chained together to achieve a lot of flexible output.

Bandit 8 - Command Piping

1 – The first command
2 – The first command's input
3 – The pipe
4 – The second command
5 – The second command's switch

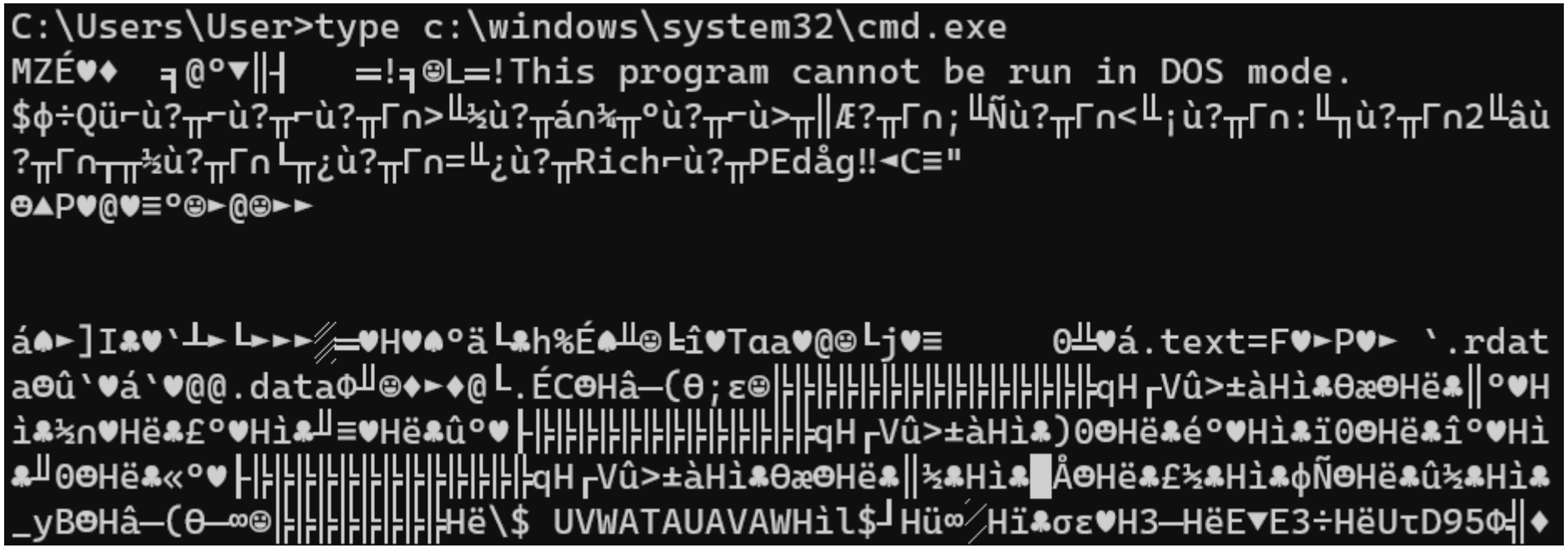
Bandit 9 - Binary Data and Text Strings

The contents of most computer files can be roughly divided into two types:

Binary Data – Which is intended to be read by software

Text Strings – Which is intended to be read by humans

Bandit 9 - Binary Data and Text Strings



Files containing binary data will output gibberish when read from the CLI console.

Bandit 9 - Strings Command

The Strings command is used to return human-readable text from files. It is often used to find text inside of files that also contain both text and binary data.

Bandit 9 - Strings Command

1 – The command itself

2 – The file to extract strings from

Bandit 10 - Base64 Command

The Base64 command encodes / decodes data according to the Base64 system. It is often used to convert data for transmission across computer networks.

0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	I	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Bandit 10 - Base64 Command

The characters used in Base 64 encoding are shown here. Note that all Base 64 encoded strings must consist of a number of characters that is divisible by 4.

0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	I	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Bandit 10 - Base64 Command

```
└─$ echo -n password | base64
cGFzc3dvcmQ=
```

In cases where an encoded string is not divisible by 4, the encoding process will “pad out” the string with equal symbols until the string is divisible by 4.

Bandit 10 - Base64 Command

1 – The command itself

2 – The decode switch

3 – The file to be operated upon

Bandit 11 – ROT13 Cipher

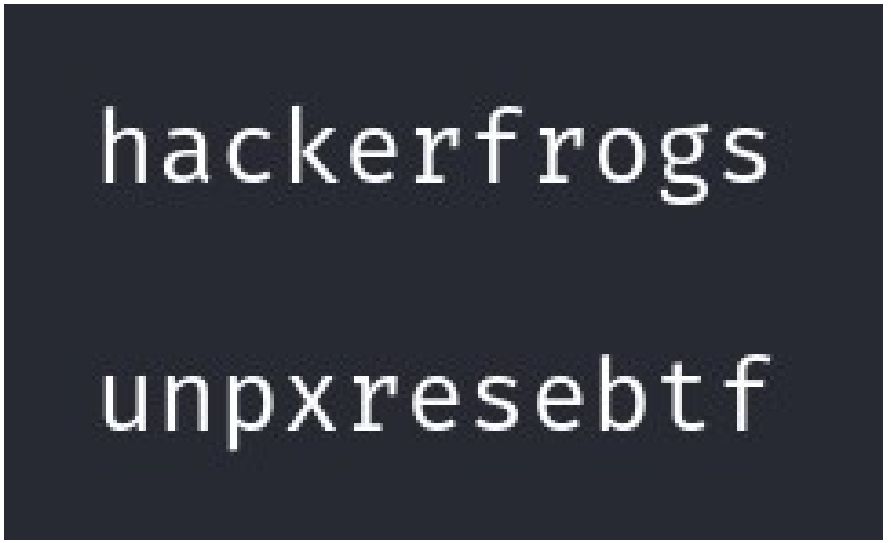
```
bandit11@bandit:~$ cat data.txt
Gur cnffjbeq vf 7k16JArUVv5LxVuJfsSVdbbtaHGlw9D4
```

The data.txt file contents are an encrypted message. The encryption method is called ROT13

Bandit 11- ROT13 Cipher

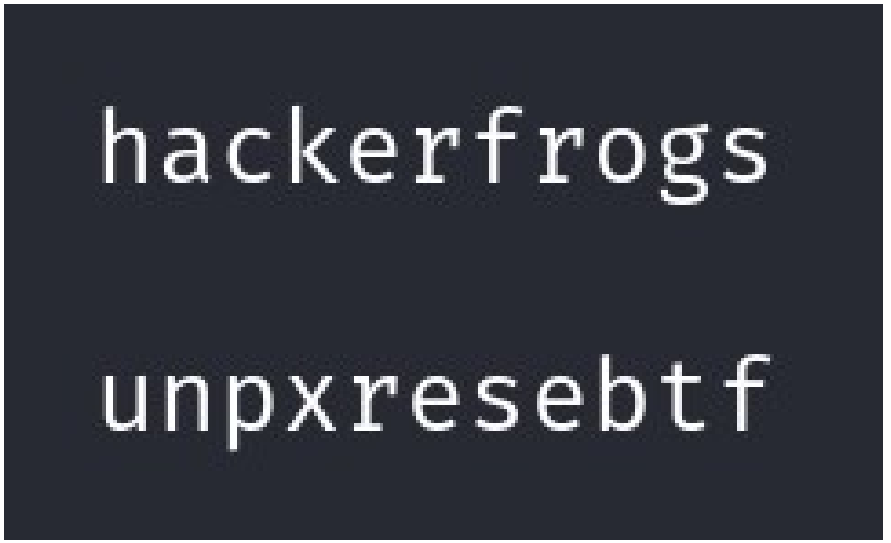
The ROT13 cipher is a simple substitution cipher where the encryption method is shift each plaintext letter 13 positions in the alphabet to form the ciphertext

Bandit 11- ROT13 Cipher



So if we use this cipher to encrypt the plaintext `hackerfrogs`, the resulting ciphertext would be `unpxresebtf`

Bandit 11- ROT13 Cipher



To decrypt the ciphertext we would do the same operation, shifting each ciphertext letter by 13 places in the alphabet

Bandit 11 - Tr Command

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

The Linux Tr command can be used to transform specified characters to other specified characters, and it can be used to simulate ROT13 decryption

Bandit 12 – XXD Command

```
xxd -r data.txt > data
```

In Linux, the XXD program is used to both create hex dump files as well as convert those hex dump files back into their original formats

Bandit 12 – Compressed Files

```
bandit12@bandit:/tmp/...bandit12theshyhat$ file data
data: gzip compressed data, was "data2.bin", last modi
2024, max compression, from Unix, original size modulo
```

In computing, compression is often used to reduce the size of files, either to reduce the amount of storage space for those files or to transmit them across networks

Bandit 12 – Compressed Files

```
bandit12@bandit:/tmp/...bandit12theshyhat$ gunzip data
gzip: data: unknown suffix -- ignored
```

Depending on the method of compression, different programs need to be used to decompress the files and get access to the original contents

Bandit 12 – Compressed Files

```
bandit12@bandit:/tmp/...bandit12theshyhat$ gunzip data
gzip: data: unknown suffix -- ignored
```

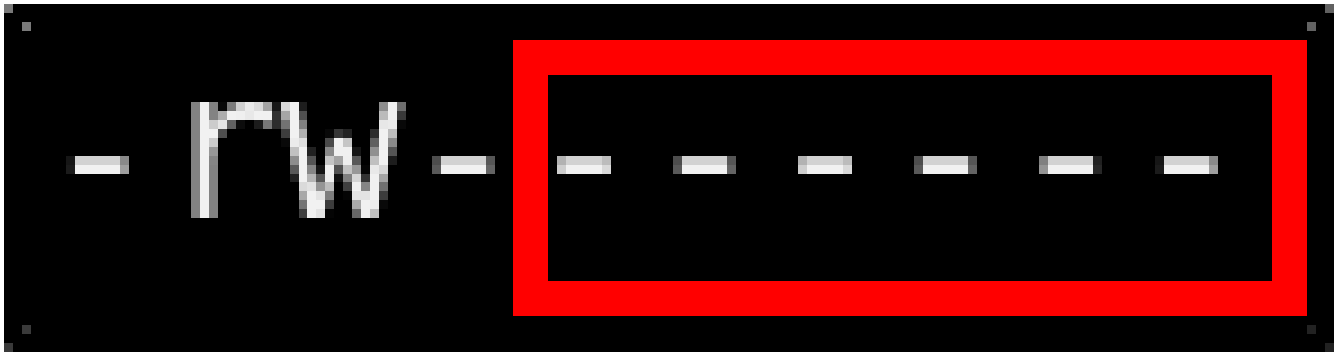
In addition, many programs will not decompress a file unless it has a specific file extension

Bandit 13 – SSH Private Keys

```
bandit13@bandit:~$ file sshkey.private
sshkey.private: PEM RSA private key
```

SSH private keys are an alternative method of login via the SSH program

Bandit 13 – SSH Private Keys



In order to use an SSH private key, it must have the correct file permissions: specifically, the file must not have global or group permissions of any kind

Bandit 14 – Remote Services

```
bandit14@bandit:~$ nmap -sV -p30000 localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-14 19:45 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000066s latency).

PORT      STATE SERVICE VERSION
30000/tcp open  ndmps?
1 service unrecognized despite returning data. If you know the ser
gerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port30000-TCP:V=7.94SVN%I=7%D=4/14%Time=67FD65C7%P=x86_64-pc-li
SF:r(GenericLines,32,"Wrong!\x20Please\x20enter\x20the\x20correct\
```

In this level, we're instructed to send the password of the current level to localhost port 30000

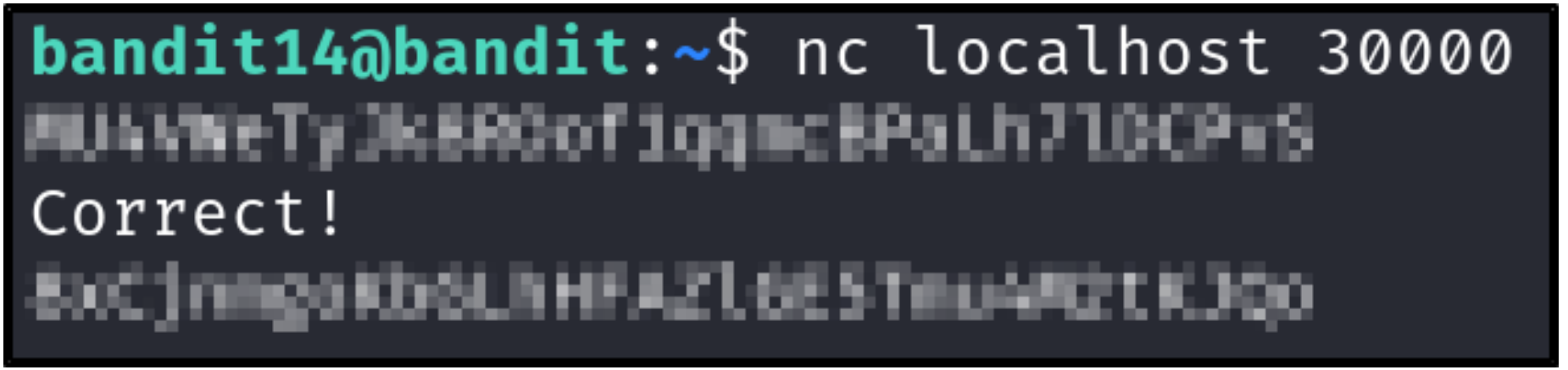
Bandit 14 – Remote Services

```
bandit14@bandit:~$ nmap -sV -p30000 localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-14 19:45 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000066s latency).

PORT      STATE SERVICE VERSION
30000/tcp open  ndmps?
1 service unrecognized despite returning data. If you know the ser
gerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port30000-TCP:V=7.94SVN%I=7%D=4/14%Time=67FD65C7%P=x86_64-pc-li
SF:r(GenericLines,32,"Wrong!\x20Please\x20enter\x20the\x20correct\
```

When scanned with **nmap**, we see that there is some sort of custom service being run on that port

Bandit 14 – Remote Services



We can connect to the service using Netcat to send the password and get the password for the next level

Bandit 15 – Sending Via OpenSSL

SSL (Secure Socket Layers) is a networking protocol which enables devices (computers, phones, etc) to communicate using an encrypted connection

Bandit 15 – Sending Via OpenSSL

In reality, SSL was replaced by the TLS (Transport Layer Security) protocol a long time ago, but modern audiences still use the terms SSL and TLS interchangeably

Bandit 15 – Sending Via OpenSSL

SSL/TLS is used to provide encrypted communication with a few other networking protocols, most notably HTTPS, which allows for encrypted webpage communication

Bandit 15 – Sending Via OpenSSL

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

```
Can't use SSL_get_servername
depth=0 CN = SnakeOil
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN = SnakeOil
verify return:1
Correct!
[redacted]
```

We can use the OpenSSL program to send the password to the port and get the password

Bandit 16 – Port Enumeration

```
bandit16@bandit:~$ nmap -p31000-32000 -vv -sV localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-1
NSE: Loaded 46 scripts for scanning.
Initiating Ping Scan at 18:28
Scanning localhost (127.0.0.1) [2 ports]
Completed Ping Scan at 18:28, 0.00s elapsed (1 total ho
Initiating Connect Scan at 18:28
Scanning localhost (127.0.0.1) [1001 ports]
Discovered open port 31960/tcp on 127.0.0.1
Discovered open port 31691/tcp on 127.0.0.1
Discovered open port 31046/tcp on 127.0.0.1
Discovered open port 31790/tcp on 127.0.0.1
Discovered open port 31518/tcp on 127.0.0.1
```

In this level, we are instructed to locate a localhost port which is serving SSL in a specific range, then send the current level's password to that port

Bandit 16 – Port Enumeration

```
bandit16@bandit:~$ nmap -p31000-32000 -vv -sV localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-1
NSE: Loaded 46 scripts for scanning.
Initiating Ping Scan at 18:28
Scanning localhost (127.0.0.1) [2 ports]
Completed Ping Scan at 18:28, 0.00s elapsed (1 total ho
Initiating Connect Scan at 18:28
Scanning localhost (127.0.0.1) [1001 ports]
Discovered open port 31960/tcp on 127.0.0.1
Discovered open port 31691/tcp on 127.0.0.1
Discovered open port 31046/tcp on 127.0.0.1
Discovered open port 31790/tcp on 127.0.0.1
Discovered open port 31518/tcp on 127.0.0.1
```

We can use the Nmap program to discover which ports are open within the specified range, and then determine what services are running

Bandit 18 – SSH Private Keys

```
Can't use SSL_get_servername
depth=0 CN = SnakeOil
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN = SnakeOil
verify return:1
Correct!
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAvmOkuifmMg6HL2YPIOjon6iWfbp7c3jx34YkYWqUH57SUdyJ
imZzeyGC0gtZPGujUSxiJSWI/oTqexh+cAMTSMlOJf7+BrJObArnxd9Y7YT2bRPQ
```

The previous level didn't give us a password, but rather an SSH private key. These keys can be used instead of passwords

Bandit 17 – SSH Private Keys

```
chmod 600 bandit17.key
```

```
ssh -i bandit17.key bandit17@bandit.labs.overthewire.org -p 2220
```

Simply creating this file on our host is not enough, because SSH private keys require very specific file permissions to be considered valid

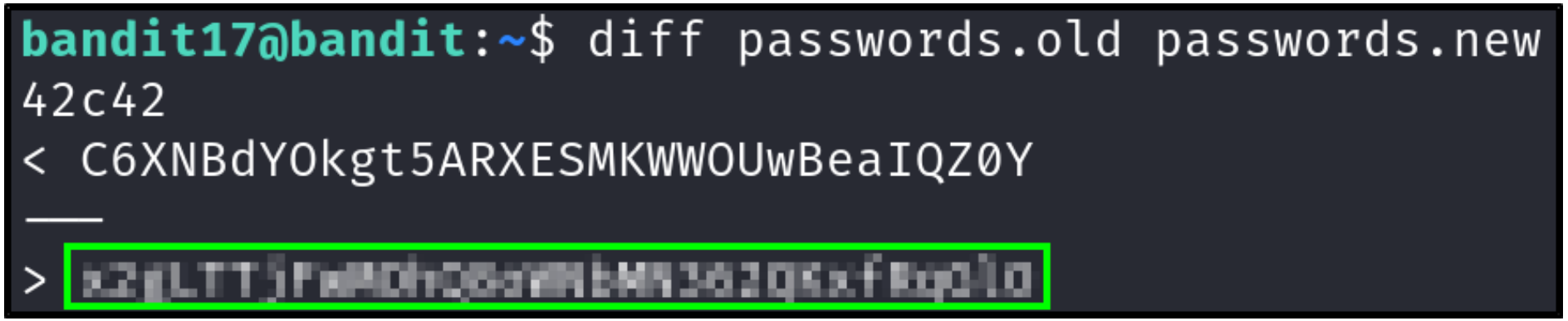
Bandit 17 – SSH Private Keys

```
chmod 600 bandit17.key
```

```
ssh -i bandit17.key bandit17@bandit.labs.overthewire.org -p 2220
```

After modifying the file permissions with the `chmod` command, we can use SSH with the `-i` parameter to login with the key

Bandit 17 – The Diff Command



We are instructed to find the line that has changed between the **passwords.old** file, and the **passwords.new** file, and use that as the password for the next level

Bandit 17 – The Diff Command

```
bandit17@bandit:~$ diff passwords.old passwords.new
42c42
< C6XNBdYOkgt5ARXESMKWWOUwBeaIQZ0Y
---
> [redacted]
```

The Diff command can be used to do this

Bandit 18 – SSH Command on Login

```
  Enjoy your stay!

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

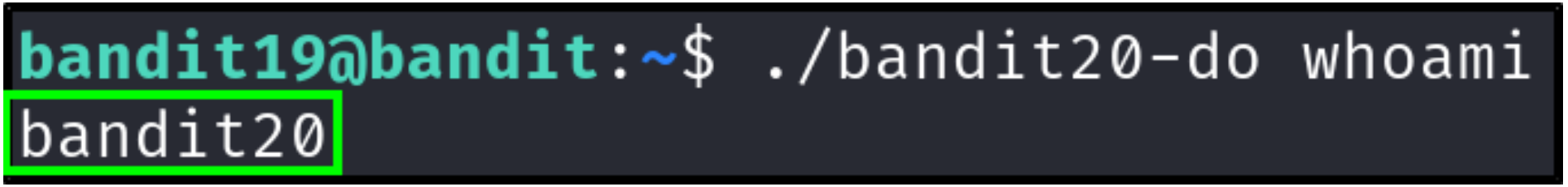
The login for this level is designed to log the user out as soon as they login, so they can't read the password for the next level

Bandit 18 – SSH Command on Login

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

SSH can be used to run a command immediately on login, and through this method we can read the password for the next level

Bandit 19 – SUID Binary



This level features a SUID binary that runs as the bandit20 user, because SUID binaries always run in the context of its file owner (bandit20)

Bandit 19 – SUID Binary

```
bandit19@bandit:~$ cat /etc/bandit_pass/bandit20
cat: /etc/bandit_pass/bandit20: Permission denied
bandit19@bandit:~$ ./bandit20-do cat /etc/bandit_pass/bandit20
0qXahGBCjOvMN5Gna7iOWsCfIyXOUbrD
```

We can use this SUID binary to read the bandit20's password

Bandit 20 – Netcat Listener Setup

```
echo [redacted] | nc -nlvp 1234 &

Listening on 0.0.0.0 1234
```

This level requires us to run a SUID binary that contacts a localhost port, and if that localhost port is outputting the password to bandit20, then it will reveal the password to bandit level 21

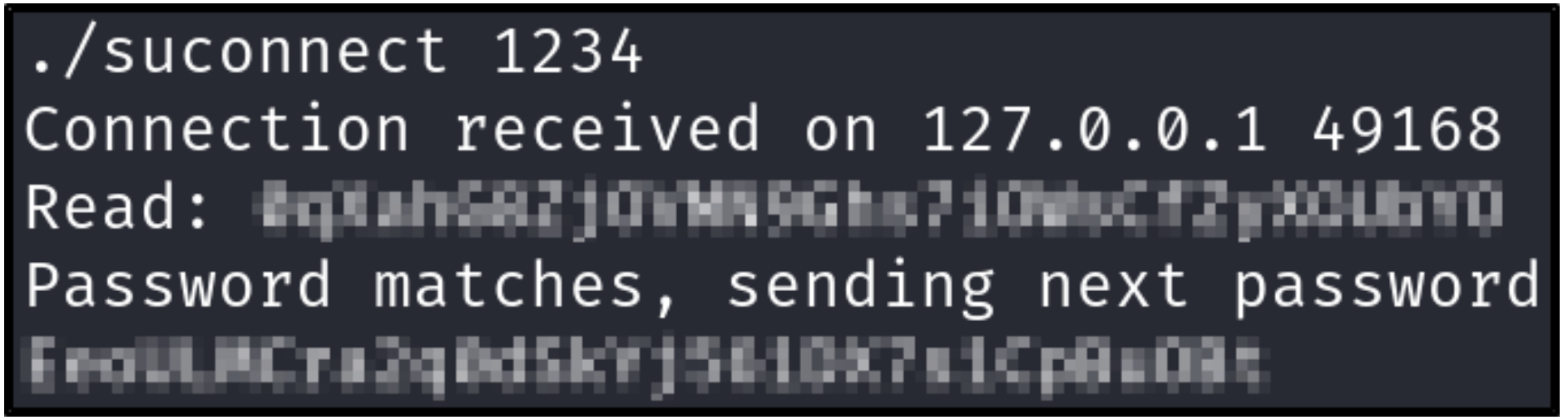
Bandit 20 – Netcat Listener Setup

```
echo [redacted] | nc -nlvp 1234 &

Listening on 0.0.0.0 1234
```

The first thing we will do is create a localhost listening port using **netcat** which outputs the password for bandit20

Bandit 20 – Netcat Listener Setup



The second step is to contact that listening port using the **suconnect** SUID binary to get the password for the next level

Bandit 21 – Cronjob Hacking

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

This level requires us to inspect cronjobs, which are commands that are run at regular intervals. There's a cronjob setup for the bandit22 user

Bandit 21 – Cronjob Hacking

```
bandit21@bandit:~$ cat /usr/bin/cronjob_bandit22.sh
#!/bin/bash
chmod 644 /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
cat /etc/bandit_pass/bandit22 > /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
```

And we are directed to a script file, and we inspect it for possible vulnerabilities

Bandit 21 – Cronjob Hacking

```
bandit21@bandit:~$ cat /usr/bin/cronjob_bandit22.sh
#!/bin/bash
chmod 644 /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
cat /etc/bandit_pass/bandit22 > /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
```

We see that the password for bandit22 is copied to a file in the /tmp directory

Bandit 21 – Cronjob Hacking

```
bandit21@bandit:~$ cat /tmp/t7O6lds9S0RqQh9aMcz6ShpAoZKF7fgv
```

So we can read that file to get the password for the next level

Bandit 22 – Cronjob Hacking

```
#!/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
```

This is the cronjob script that is being run by user Bandit23

Bandit 22 – Cronjob Hacking

```
#!/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 script contains a variable, `mytarget`, which is an MD5 hash of a certain string

Bandit 22 – Cronjob Hacking

```
#!/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
```

That MD5 hash is then used a file name in the /tmp directory, and the password for the Bandit23 user is written into that file

Bandit 23 – Cronjob Hacking

```
myname=$(whoami)

cd /var/spool/$myname/foo
echo "Executing and deleting all scripts in /var/spool/$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 ./$i
        fi
        rm -f ./$i
    fi
done
```

This is the cronjob script for user Bandit24

Bandit 23 – Cronjob Hacking

```
myname=$(whoami)

cd /var/spool/$myname/foo
echo "Executing and deleting all scripts in /var/spool/$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 ./$i
        fi
        rm -f ./$i
    fi
done
```

The script runs all files in a certain directory

Bandit 23 – Cronjob Hacking

```
cat /etc/bandit_pass/bandit24 > /tmp/ ... shyhat_bandit24_pass.txt
```

We can write our own script in that directory, and have Bandit24 execute it with the cronjob, then retrieve the password in the /tmp directory afterwards

Bandit 24 – Brute Forcing

```
bandit24@bandit:~$ nc localhost 30002
I am the pincode checker for user bandit25. Please enter the password for
on a single line, separated by a space.
[illegible] 0000
Wrong! Please enter the correct current password and pincode. Try again.
```

In this level, we have to send the password for the previous level along with a four-digit PIN to a certain port on the localhost server

Bandit 24 – Brute Forcing

```
bandit24@bandit:~$ nc localhost 30002
I am the pincode checker for user bandit25. Please enter the password for
on a single line, separated by a space.
[illegible] 0000
Wrong! Please enter the correct current password and pincode. Try again.
```

If we send each possible combination of four-digit PINs, it will take 10000 tries to test them all, so we need to use a script to automate the process

Bandit 25 – SSH Private Key

```
bandit25@bandit:~$ cat bandit26.sshkey
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEApis2AuoooEqeYWamtwX2k5z9uU1Afl2
pTfaeRHXzr0Y0a5Oe3GB/+W2+PReif+bPZlzTY1XFwpk+Di
```

We're given an SSH private key to login as the bandit26 user, so we need to set it up

Bandit 25 – SSH Private Key

```
scp -P 2220
bandit25@bandit.labs.overthewire.org
:/home/bandit25/bandit26.sshkey .
```

We can download the private key using the Scp command

Bandit 25 – SSH Private Key

```
scp -P 2220
bandit25@bandit.labs.overthewire.org
:/home/bandit25/bandit26.sshkey .
```

The first part of the command specifies the port to communicate on

Bandit 25 – SSH Private Key

```
scp -P 2220
bandit25@bandit.labs.overthewire.org
:/home/bandit25/bandit26.sshkey .
```

The second part of the command specifies the user and server to communicate with

Bandit 25 – SSH Private Key

```
scp -P 2220
bandit25@bandit.labs.overthewire.org
:/home/bandit25/bandit26.sshkey .
```

And the third part of the command specifies filepath to the file we want to download on the remote server, and the filepath to save the file to on the localhost.

Bandit 25 – SSH Private Key

```
ssh -i bandit26.sshkey
bandit26@bandit.labs.overthewire.org
-p 2220
```

With the SSH private key on our localhost, we can use a modified SSH command to login as bandit26 with the private key as our credentials

Bandit 26 – Restricted Shell

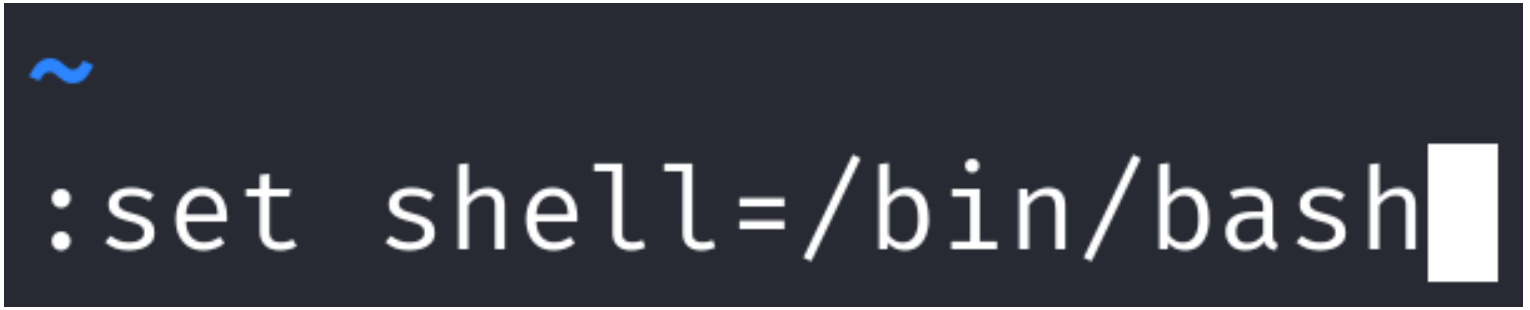
```
bandit25@bandit:~$ strings /usr/bin/showtext
#!/bin/sh
export TERM=linux
exec more ~/text.txt
exit 0
```

We're told that Bandit26 is using a different shell, and it turns out it's a shell script that looks like the above

Bandit 26 – Restricted Shell

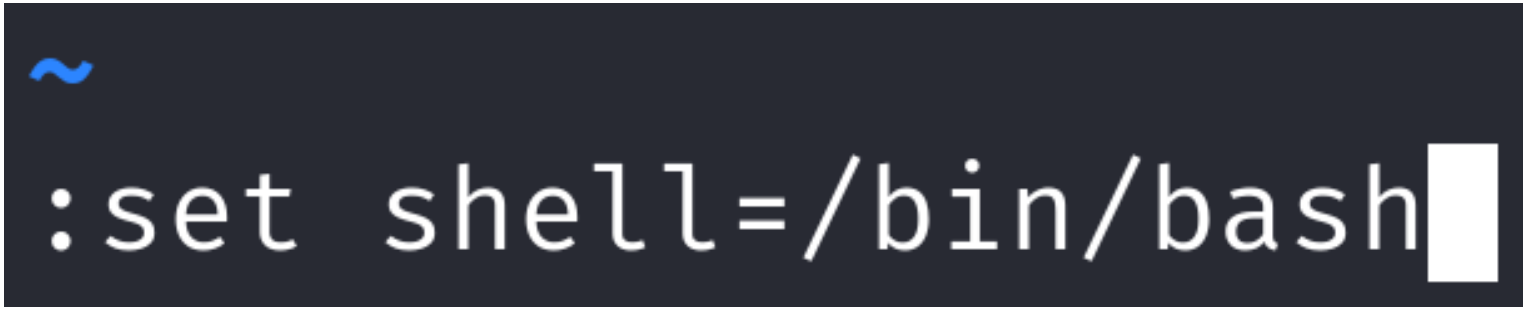
Since the file that's being read is only a few lines long, we need to make the terminal text really big, so that it doesn't fit on one page

Bandit 26 – Restricted Shell



In some Linux environments, the More command can be interrupted into the Vi or Vim text editor by pressing the v key during More execution

Bandit 26 – Restricted Shell



From the Vi text editor, we can break out into an interactive shell

Bandit 27 – Git: README files

The next several levels in Bandit cover Git software features

Bandit 27 – Git: README files

Git is a distributed version control system that tracks versions of files. It is used by programmers to help develop software collaboratively

Bandit 27 – Git: README files

```
mkdir /tmp/...bandit27git/
cd /tmp/...bandit27git/
git clone ssh://bandit27-git@localhost
:2220/home/bandit27-git/repo
```

The first step to solving these levels is to clone the repository into a writable directory

Bandit 27 – Git: README files

```
bandit27@bandit:/tmp/ ... bandit27git/repo$ cat README
The password to the next level is: Yz9IpL0sBcCeuG7m9uQFtBZWpS6HZ8cM
```

Git projects are called repositories (repos), and each Git repo will have a README file which explains the contents of the Git repo

Bandit 28 – Git: Log files

This level in Bandit covers Git software features

Bandit 28 – Git: Log files

Git is a distributed version control system that tracks versions of files. It is used by programmers to help develop software collaboratively

Bandit 28 – Git: Log files

```
bandit28@bandit:/tmp/ ... bandit28git/repo$ git log
commit 674690a00a0056ab96048f7317b9ec20c057c06b (HEAD
Author: Morla Porla <morla@overthewire.org>
Date:   Thu Apr 10 14:23:19 2025 +0000

    fix info leak
```

Git keeps track of each instance of changes to file in the repo (commits) within its log, and we can access it with the `git log` command

Bandit 28 – Git: Log files

```
bandit28@bandit:/tmp/ ... bandit28git/repo$ git log
commit 674690a00a0056ab96048f7317b9ec20c057c06b (HEAD
Author: Morla Porla <morla@overthewire.org>
Date:   Thu Apr 10 14:23:19 2025 +0000

    fix info leak
```

By default the `git log` command shows the commit hash, the user who made the commit, the date the commit was made, and the commit description or comment

Bandit 28 – Git: Log files

```
diff --git a/README.md b/README.md
index d4e3b74..5c6457b 100644
--- a/README.md
+++ b/README.md
@@ -4,5 +4,5 @@ Some notes for level29 of bandit.
 ## credentials

 - username: bandit29
-- password: [redacted]
+- password: xxxxxxxxxx
```

But if we run the `git log` command with the `-p` flag, we can see what data changed within the commit

Bandit 29 – Git: Branches

This level in Bandit covers Git software features

Bandit 29 – Git: Branches

Git is a distributed version control system that tracks versions of files. It is used by programmers to help develop software collaboratively

Bandit 29 – Git: Branches

```
bandit29@bandit:/tmp/ ... bandit29git/repo$ git branch
* master
```

Git branches are copies of a specified commit, which allow different versions of the commit to be modified without affecting the main codebase

Bandit 29 – Git: Branches

```
bandit29@bandit:/tmp/ ... bandit29git/repo$ git branch
* master
```

If we use the `git branch` command by itself, it will show us the current branch we are using. The `master` branch is the default branch

Bandit 29 – Git: Branches

```
bandit29@bandit:/tmp/ ... bandit29git/repo$ git branch -a
* master
  remotes/origin/HEAD → origin/master
  remotes/origin/dev
  remotes/origin/master
  remotes/origin/sploits-dev
```

But if we use the `git branch` command with the `-a` flag, all of the branches in the repo will be displayed

Bandit 29 – Git: Branches

```
bandit29@bandit:/tmp/ ... bandit29git/repo$ git checkout dev
branch 'dev' set up to track 'origin/dev'.
Switched to a new branch 'dev'
```

We can use the `git checkout` command to switch branches from one to another

Bandit 29 – Git: Branches

```
diff --git a/README.md b/README.md
index d4e3b74..5c6457b 100644
--- a/README.md
+++ b/README.md
@@ -4,5 +4,5 @@ Some notes for level29 of bandit.
 ## credentials

 - username: bandit29
-- password: [redacted]
+- password: xxxxxxxxxx
```

Each branch in Git has its own logs and commits

Bandit 30 – Git: Tags

This level in Bandit covers Git software features

Bandit 30 – Git: Tags

Git is a distributed version control system that tracks versions of files. It is used by programmers to help develop software collaboratively

Bandit 30 – Git: Tags

```
bandit30@bandit:/tmp/....bandit30git/repo$ git tag
secret
```

In this level, we're learning about Git tags, which are a way of attaching labels to commits. To list all tags in a repo, use the `git tags` command

Bandit 30 – Git: Tags

```
bandit30@bandit:/tmp/....bandit30git/repo$ git show-ref --tags
84368f3a7ee06ac993ed579e34b8bd144afad351 refs/tags/secret
```

To see all of the commits with tags attached to them, you can use the git `show-ref --tags` command

Bandit 30 – Git: Tags

Finally, you can retrieve the contents of all commits that contain specific tags by using the `git show <tag_name>` command

Bandit 31 – Git: Commit / Push

This level in Bandit covers Git software features

Bandit 31 – Git: Commit / Push

Git is a distributed version control system that tracks versions of files. It is used by programmers to help develop software collaboratively

Bandit 31 – Git: Commit / Push

```
This time your task is to push a file to the remote repository.

Details:
    File name: key.txt
    Content: 'May I come in?'
    Branch: master
```

In this level, we're tasked with making a commit and push to a remote repo in Git

Bandit 31 – Git: Commit / Push

```
echo 'May I come in?' >  key.txt
git add key.txt -f
```

First, we can add a local file to a Git repo by using the `git add <file_name>` command

Bandit 31 – Git: Commit / Push

```
git commit -m "Add key.txt file"
```

```
[master 530a279] Add key.txt file
 1 file changed, 1 insertion(+)
 create mode 100644 key.txt
```

After that you can use the `git commit -m "commit comment"` command to save a commit

Bandit 31 – Git: Commit / Push

```
Writing objects: 100% (3/3), 329 bytes | 329.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote: ### Attempting to validate files... ####
remote:
remote: .oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.
remote:
remote: Well done! Here is the password for the next level:
remote: [redacted]
remote:
remote: .oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.oOo.
remote:
```

Finally, you can push that commit by using the `git push` command

Bandit 32 – Shell Escape

```
WELCOME TO THE UPPERCASE SHELL
>> whoami
sh: 1: WHOAMI: Permission denied
```

This level features a restricted shell, which is a common type of challenge in CTFs.

Bandit 32 – Shell Escape

```
WELCOME TO THE UPPERCASE SHELL
>> whoami
sh: 1: WHOAMI: Permission denied
```

This shell appears to use the `sh` shell, but it transforms all alphabetic input into uppercase, which messes up all commands with alphabetic elements

Bandit 32 – Shell Escape

```
>> $0
$ echo $SHELL
/home/bandit32/uppershell
```

In this case, we can use the `$0` command (which uses no letters) to open a new shell