# Work & documentation notes of various wargames

Galen Rowell

June 30, 2020

# 1 Bandit

## 1.1 Levels

### 1.1.1 bandit0

**Password to enter:** *bandit0*
**Challenge:** Solved using the **ssh** command, which included use of flags to set user & port.

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

### 1.1.2 bandit1

**Password to enter:** *boJ9jbbUNNfktd78OOpsqOltutMc3MY1*
**Challenge:** Reading a file named '-', this was problematic due to many common shell commands using '-' to prefix an option or flag.

```
cat ./-
```

### 1.1.3 bandit2

**Password to enter:** *CV1DtqXWVFXTvM2F0k09SHz0YwRINYA9*
**Challenge:** With spaces in a filename, shell programs will interpret the input as several arguments (instead of one space-delimited string). This issue can be solved two ways.

```
cat 'spaced filename'
```

```
cat spaced\ filename
```

### 1.1.4 bandit3

**Password to enter:** *UmHadQclWmgdLOKQ3YNgjWxGoRMb5luK*
**Challenge:** The file is prepended by a '.', which causes it to be hidden from most views. The *-A* flag for **ls** will show all hidden files except '.' & '..', which are part of the directory itself.

```
ls -A1
```

### 1.1.5 bandit4

**Password to enter:** *pIwrPrtPN36QITSp3EQaw936yaFoFgAB*
**Challenge:** The file is hidden in one of ' /inhere/-file0,9'. They contain special characters that interfere with the terminal environment. The use of **less** aids, as it prompts before reading a binary file and provides somewhat of a sandbox to prevent the tty from being broken.

```
less ./-file0[0-9]
```

*Note: use :n when inside **less** to go the next file*

### 1.1.6 bandit5

**Password to enter:** *koReBOKuIDDepwhWk7jZC0RTdopnAYKh*
**Challenge:** The file is within one of many sub-folders, with human readable encoding and a file size of '1033' bytes. The use of **ls** with the recursive flag *-R*, combined with **grep** to select the file with the given size solves this problem.

```
ls -Al -R | grep --color -C 5 -e '1033'
```

### 1.1.7 bandit6

**Password to enter:** *DXjZPULLxYr17uwoI01bNLQbtFemEgo7*
**Challenge:** The file is somewhere on the server, so we should search recursively from the root of the drive. We are given the owner name, group name and size of the file, which we can plug into **find** to find the file.

```
find / -group bandit6 -size 33c 2>&1 | grep -v "Permission denied"
```

*Note: The use of a terminal redirect and **grep** remove the output of excessive file permission warnings*

### 1.1.8 bandit7

**Password to enter:** *HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs*
**Challenge:** This level is a simple grep search for the word *millionth* in a large keyword text file.

### 1.1.9 bandit8

**Password to enter:** *cvX2JJa4CFALtqS87jk27qwqGhBM9plV*
**Challenge:** The password is the only line that occurs once within an unordered text file.

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

*Note: The -u flag of **uniq** ensures only lines of 1 occurrence are printed*

### 1.1.10 bandit9

**Password to enter:** *UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhR*
**Challenge:** The given file is a binary encoded file, IE. it is not in plaintext or easy to read. **strings** will only print human-readable strings from a given input, and the use of **grep** will limit the output to a manageable size.

```
strings data.txt | grep -Ee [=]+
```

*Note: The [=]+ pattern of **grep** searches for one or more occurances of = in each line, EG. =, ==== or ========*

### 1.1.11 bandit10

**Password to enter:** *truKLdjsbJ5g7yyJ2X2R0o3a5HQJFuLk*
**Challenge:** The file is encoded in base 64, which can be encoded and decoded using the **base64** program.

```
base64 -d data.txt
```

### 1.1.12 bandit11

**Password to enter:** *IFukwKGsFW8MOq3IRFqrxE1hxTNEbUPR*
**Challenge:** The file is encoded in a ROT-13 cipher, meaning that all letters in the alphabet have been shifted 13 places. **tr** is a unix program which is used the **tr**anslate various sets of text.

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

### 1.1.13 bandit12

**Password to enter:** *5Te8Y4drgCRfCx8ugdwuEX8KFC6k2EUu*
**Challenge:** The given file is a hexdump of a binary file, which is a compressed **gzip** file. The **gzip** file is itself compressed many times with **gzip, bzip2** & **tar**. One of the best ways to discover what encoding a file has is to run **file** on the given file, as well as visual inspection with **less**.

The methodology used to solve the level was to inspect the file encoding using **file**, find the appropriate decompression program, then repeat until the end result was the final plain-text.

**gzip** has a 'unix-pipe' program version named **zcat**.
**bzip2** has a 'unix-pipe' program version named **bzcat**.
**tar** acts like a 'unix-pipe' program with the arguments **tar xO**.
The use of these *unix-pipe* versions allow use to pipe the input through *std-in* and have the decompressed output sent to *std-out*.

to uncompress a hexdump

```
xxd -r data.txt a.bin
```

to test the file encoding/type from std-in

```
file -
```

the series of decompression required

```
xxd -r data.txt a.bin
zcat a.bin | bzcat | zcat | tar xO | tar xO | bzcat | tar xO | zcat
```

### 1.1.14 bandit13

**Password to enter:** *8ZjyCRiBWFYkneahHwxCv3wb2a1ORpYL*
**Challenge:** This is a small challenge regarding SSH keys, a private key to access the next level is given. **ssh**'s **-i** flag uses the given key to authenticate the connection.

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

### 1.1.15 bandit14

**Password to enter:** *4wcYUJFw0k0XLShlDzztnTBHiqxU3b3e*
**Challenge:** This passwords for the next level is retreieved by sending the password for the current level to port 30,000 of the machine *(IE. localhost)*. This was solved by using a dated, but universal, shell program called **n**et **c**at.

```
1 nc localhost 30000
2   4wcYUJFw0k0XLShlDzztnTBHiqxU3b3e
3   Correct!
4   BfMYroe26WYalil77FoDi9qh59eK5xNr
```

*Note: Line #2 was entered manually, lines #3-4 were a 'response' from port 30,000*

### 1.1.16 bandit15

**Password to enter:** *BfMYroe26WYalil77FoDi9qh59eK5xNr*
**Challenge:** This challenge covers the use of **openssl**, and it's broad uses as a cryptographic tool for certificate and key generation/management. **openssl** has many sub-commands, of particular note is the **s_c lient** sub-command.

```
cat bandit15 | openssl s_client -connect localhost:30001 -ign_eof
```

*Note: the flag '-ign_eof' is used to allow the piping from cat into the session, alternatively the text can be manually input.*

#### 1.1.17 bandit16

**Password to enter:**
**Challenge:**

#### 1.1.18 bandit17

**Password to enter:**
**Challenge:**

#### 1.1.19 bandit18

**Password to enter:**
**Challenge:**

#### 1.1.20 bandit19

**Password to enter:**
**Challenge:**

#### 1.1.21 bandit20

**Password to enter:**
**Challenge:**

## Links & resources

1. The bandit wargame is run on a remote server, accessed by ssh. In order to write scripts to log what I executed and re-run/solve the level then a tool is needed to be able to feed the password during the handshake process. SSHpass is great for this: SSHPass tutorial
2. When scripting, it is often useful to have a temporary directory where files can be created & modified without the risk of littering such files about the filesystem. So a temporary directory (often in /tmp/) is useful, **mktemp** does this:

move to the new temporary directory

```
cd $(mktemp -d)
```

store the new temporary directory path

```
tmp_dir=$(mktemp -d)
```