Bi0s Tasks

* Linux:

**Level 0**

To get to level 0 we need to simply **SSH** into Bandit with the username: **bandit0** and password: **bandit0**

**Level 0 - 1**

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

So type cat <filename> to open it.

**Password: boJ9jbbUNNfktd78OOpsqOltutMc3MY1**

**Level 1 – 2**

We are told that the password is in a file called “-“.

We need to delimit the dash to read it. So use ./<filename> to delimit it.

**Password:** **CV1DtqXWVFXTvM2F0k09SHz0YwRINYA9**

**Level 2 - 3**

This time we simply need to read a file with spaces in it’s name.

So here we surround the file name in quotes.

**Password:** **UmHadQclWmgdLOKQ3YNgjWxGoRMb5luK**

**Level 3 – 4**

I ran "ls" to see what files and directories there are and then I ran "cd" to move into the inhere folder.

Then I ran "ls -a" to see all of the files including the hidden ones.

All hidden files and folders in linux are stored with a dot in front of their name.

The hidden file is named “.hidden” and after running "cat .hidden"

**Password: pIwrPrtPN36QITSp3EQaw936yaFoFgAB.**

**Level 4 – 5**

With the help of the command "file" which gives you information about any file passed as a parameter.

The files were names -file00 through -file09 and therefore we have to add the ./ in order for file to read the files.

**Password: koReBOKuIDDepwhWk7jZC0RTdopnAYKh**

**Level 5 – 6**

To start I "cd" into the inhere folder and ran "ls" to see the files and folders and noticed that there were a bunch of folders and inside those folders a couple of files.

To make it easier and not to read every file I tried to "find" a way to look for the specific file with the properties given.

I looked through the manual page of the find command and looked for ways of pointing out the properties of the file. I found the -size, -type, and -executable options.

I put them together with the properties given and I ran the "find" command.

**Password:** **DXjZPULLxYr17uwoI01bNLQbtFemEgo7**

**Level 6 – 7**

For this part I used the find command below to find a file owned by user bandit7, owned by group bandit6 and 33 bytes of size.

With that command we found the file in the following directory: /var/lib/dpkg/info/bandit7.password.

**Password: HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs**

**Level 7 – 8**

I saw the data.txt file and when I read it a bunch of lines came up and it looked like all of the lines had the same structure.

It was a word followed by some spaces and then a possible password.

The hint was that the password is next to the word millionth, so I used the command below to read the file and then grep the word millionth.

The command only return 1 line and it contains the password.

**Password: cvX2JJa4CFALtqS87jk27qwqGhBM9plV**

**Level 8 – 9**

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

To do this I sorted the lines alphabetically and then removed all duplicates from the output.

**Password: UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhR**

**Level 9 – 10**

According to the hint, the file contains both strings and binary data which can make it difficult to read.

In order to sort out the plain text I ran "cat data.txt | string".

The next part is to grep the lines that start with the = sign.

**Password: truKLdjsbJ5g7yyJ2X2R0o3a5HQJFuLk**

**Level 10 – 11**

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

The data.txt contains 1 line that was encoded in base64. In order to decode the file I ran the command below:

**cat data.txt | base64 –decode**

**Password: IFukwKGsFW8MOq3IRFqrxE1hxTNEbUPR**

**Level 11 – 12**

The data.txt file contains 1 line that was encrypted with the ROT13 algorithm.

In order to decrypt it, I have to replace every letter by the letter 13 positions ahead.

For example, with this encryption the letter a would be replaced with n.

**cat data.txt | tr '[A-Za-z]' '[N-ZA-Mn-za-m]'**

The command above send the line to stdout where tr shifts every letter 13 positions.

**Password: 5Te8Y4drgCRfCx8ugdwuEX8KFC6k2EUu**

**Level 12 – 13**

As mention in the description, data.txt is a **hexdump** of a file that has been **repeatedly** compressed.

To obtain the password for the next level, multiple decompression operations on the file are needed to be performed.

Consequently, this would make **CHANGES** to the current directory.

Unfortunately, we **DO NOT** have write permissions.

As such, it is suggested to create a directory under /tmp to perform our activities.

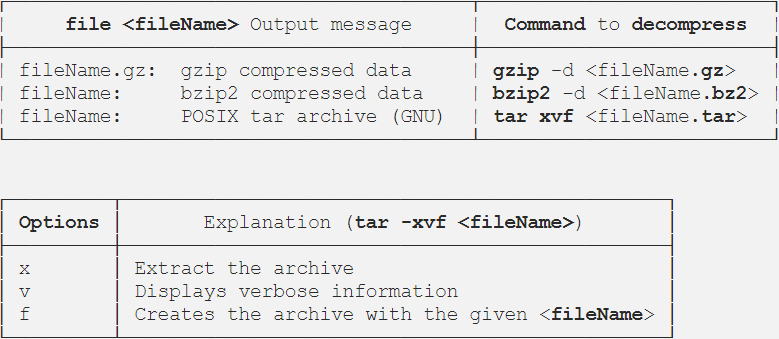
Since the file is **hexdump** of a file,  
using **xxd -r** <inputFile> <outputFile> command will:

1. First, **convert** a hex dump back to its original **BINARY** form.
2. Next, **sent** the results to the outputFile specified.

First, use **file** <fileName> command to **DETERMINE** the **file type** of the OUTPUT file.

Second, use **mv** <source> <target> command to **CHANGE** the output file to its appropriate file type stated in the previous steps.

Third, **decompress / unzip** the files using the appropriate type of command based on the file extension



Repeat the above process until running **file** <target> will display a file type of ASCII text.

If above return ASCII text, then run cat <target> to display the password in file target.

Run rm -rf /tmp/myname123 to delete the temporary created folder /tmp/myname123.

**Password: 8ZjyCRiBWFYkneahHwxCv3wb2a1ORpYL**

**Level 13 – 14**

As mention in the description above, the password is stored in **/etc/bandit\_pass/bandit14**.

Thus, running **cat /etc/bandit\_pass/bandit14**will display the password for bandit14.

To access the next level, simple login using the

sshkey.private which was provided to you on the root directory using the following command:

**ssh -i ./sshkey.private bandit14@localhost**

**Password: 4wcYUJFw0k0XLShlDzztnTBHiqxU3b3e**

**Level 14 – 15**

Using **telnet** <hostname> <port number>.

That is, **telnet** localhost 30000, it will connect to localhost on port 3000.

Once connection has been established, we will input bandit14‘s password and received a response with bandit15‘s password.

**Password: BfMYroe26WYalil77FoDi9qh59eK5xNr**

* **Algorithm:**

**Selection Sort:**

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

The algorithm maintains two subarrays in a given array. The subarray which is already sorted and the Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

**Bubble Sort:**

Bubble Sort algorithm works by repeatedly swapping the adjacent elements if they are in wrong order.

**Merge Sort:**

It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

**Binary Search:**

Search a sorted array by repeatedly dividing the search interval in half.

Begin with an interval covering the whole array.

If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.

Otherwise narrow it to the upper half.

Repeatedly check until the value is found or the interval is empty.

**Check for Balanced Parentheses matching using Stack**

Each time, when an open parenthesis is encountered push it in the stack.

When closed parenthesis is encountered, match it with the top of stack and pop it.

If stack is empty at the end, return Balanced otherwise, Unbalanced.

**Check for existence of cycle in Linked List**

Have a visited flag with each node.

Traverse the linked list and keep marking visited nodes.

Initially flag will be 0 and when it comes to the loop then it is marked as 1.

So when we see a flag as 1, then it means that already we have made it as 1 and again the same is repeating which signifies that there is a loop.

* **Programming:**

<https://codeforces.com/submissions/abhi42002/page/1>

**1061A**

In this when we use the largest number to compute the coins then the number of coins required will be minimized and at a particular stage it will be equal to the sum.

If the Sum is the odd number then we need to add one to the number of coins required since one of the cion is required to fulfill the sum that has to be calculated.

**1030A**

In this problem we need to find whether the problem is easy or Hard.

0 -> Easy

1 -> Hard

Here if 1 one is there then we should print hard.

Here if we encounter 1 then we can break and come out of the loop.