LEVEL 0

Step-by-Step Explanation:

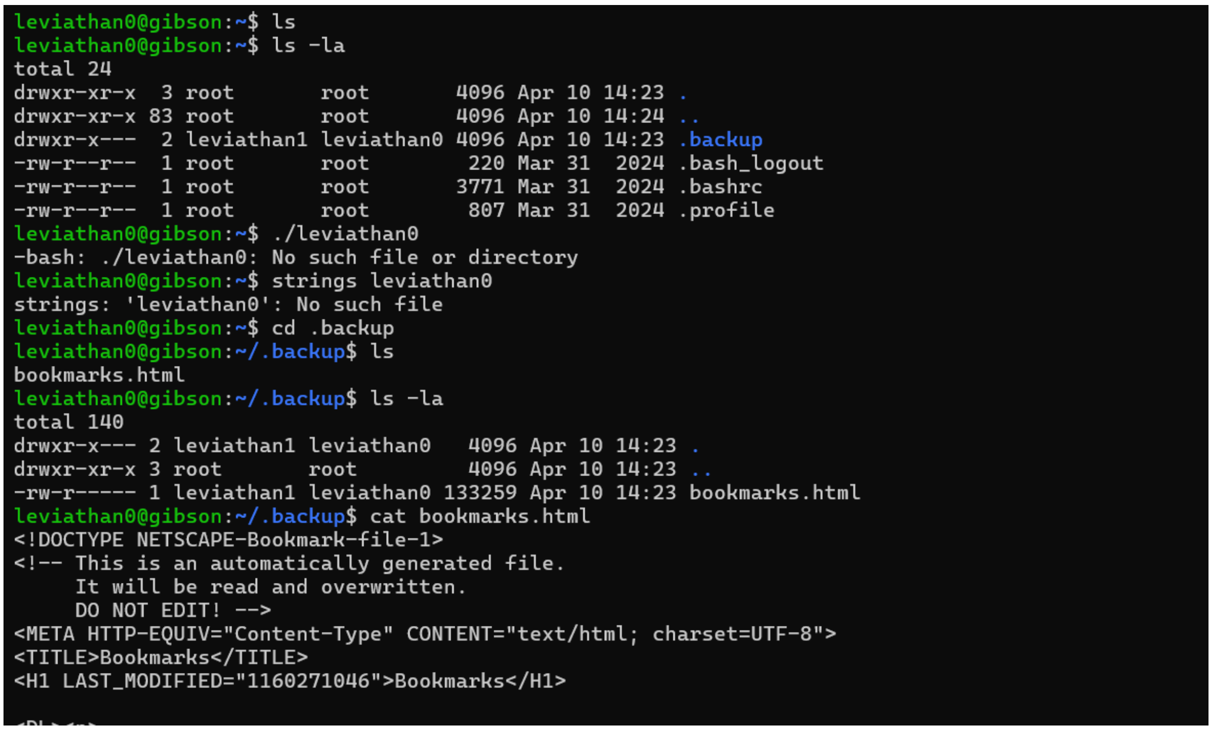
1. First, we connected to the Leviathan server and listed the files using `ls`.
2. No obvious files were found, so we used `ls -la` to show hidden files.
3. We discovered a hidden directory `.backup` owned by leviathan1 but readable by us.
4. We changed into the `.backup` directory using `cd .backup`.
5. Inside `.backup`, we listed files with `ls -la` and found a file named bookmarks.html.
6. Since bookmarks.html could contain a lot of text, and the password could be hidden inside, we used `strings bookmarks.html | grep -i pass` to quickly search for any mention of "pass".
7. This revealed the password hidden inside the file without having to scroll through a huge amount of text manually.
8. After obtaining the password, we used `ssh leviathan1@gibson` to log in to the next level.

Tools Used:

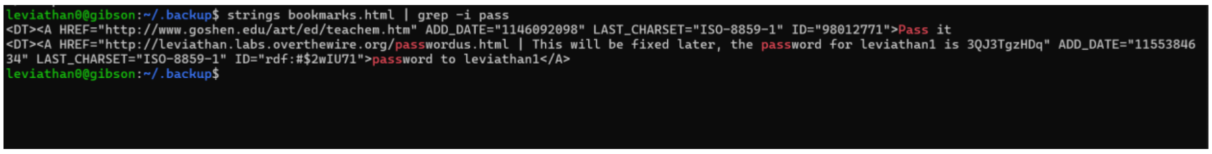
- `ls` and `ls -la` to explore the file system.
- `cd` to move between directories.
- `strings` to extract readable text from a file.
- `grep` with `-i` flag for quick and case-insensitive search.
- `ssh` to login into the next level.

Logic Behind the Solution:

- Hidden folders often contain important clues in CTFs (Capture The Flag) challenges.
- `strings` helps when facing large files, filtering only readable data.
- Using `grep` with keywords like "pass" can quickly pinpoint passwords inside huge or messy files.
- Permissions were key: the `.backup` folder was accessible because of group permissions (`leviathan0` group access).



```
leviathan0@gibson:~$ ls
leviathan0@gibson:~$ ls -la
total 24
drwxr-xr-x  3 root       root       4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root       4096 Apr 10 14:24 ..
drwxr-x---  2 leviathan1 leviathan0 4096 Apr 10 14:23 .backup
-rw-r--r--  1 root       root        220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root       3771 Mar 31  2024 .bashrc
-rw-r--r--  1 root       root        807 Mar 31  2024 .profile
leviathan0@gibson:~$ ./leviathan0
-bash: ./leviathan0: No such file or directory
leviathan0@gibson:~$ strings leviathan0
strings: 'leviathan0': No such file
leviathan0@gibson:~$ cd .backup
leviathan0@gibson:~/.backup$ ls
bookmarks.html
leviathan0@gibson:~/.backup$ ls -la
total 140
drwxr-x--- 2 leviathan1 leviathan0   4096 Apr 10 14:23 .
drwxr-xr-x 3 root       root         4096 Apr 10 14:23 ..
-rw-r----- 1 leviathan1 leviathan0 133259 Apr 10 14:23 bookmarks.html
leviathan0@gibson:~/.backup$ cat bookmarks.html
<!DOCTYPE NETSCAPE-Bookmark-file-1>
<!-- This is an automatically generated file.
     It will be read and overwritten.
     DO NOT EDIT! -->
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=UTF-8">
<TITLE>Bookmarks</TITLE>
<H1 LAST_MODIFIED="1160271046">Bookmarks</H1>
```



```
leviathan0@gibson:~/.backup$ strings bookmarks.html | grep -i pass
<DT><A HREF="http://www.goshen.edu/art/ed/teachem.htm" ADD_DATE="1146092098" LAST_CHARSET="ISO-8859-1" ID="98012771">Pass it
<DT><A HREF="http://leviathan.labs.overthewire.org/passwordus.html | This will be fixed later, the password for leviathan1 is 3QJ3TgzHDq" ADD_DATE="11553846
34" LAST_CHARSET="ISO-8859-1" ID="rdf:#$2wIU71">password to leviathan1</A>
leviathan0@gibson:~/.backup$
```

LEVEL 1

Step-by-Step Explanation:

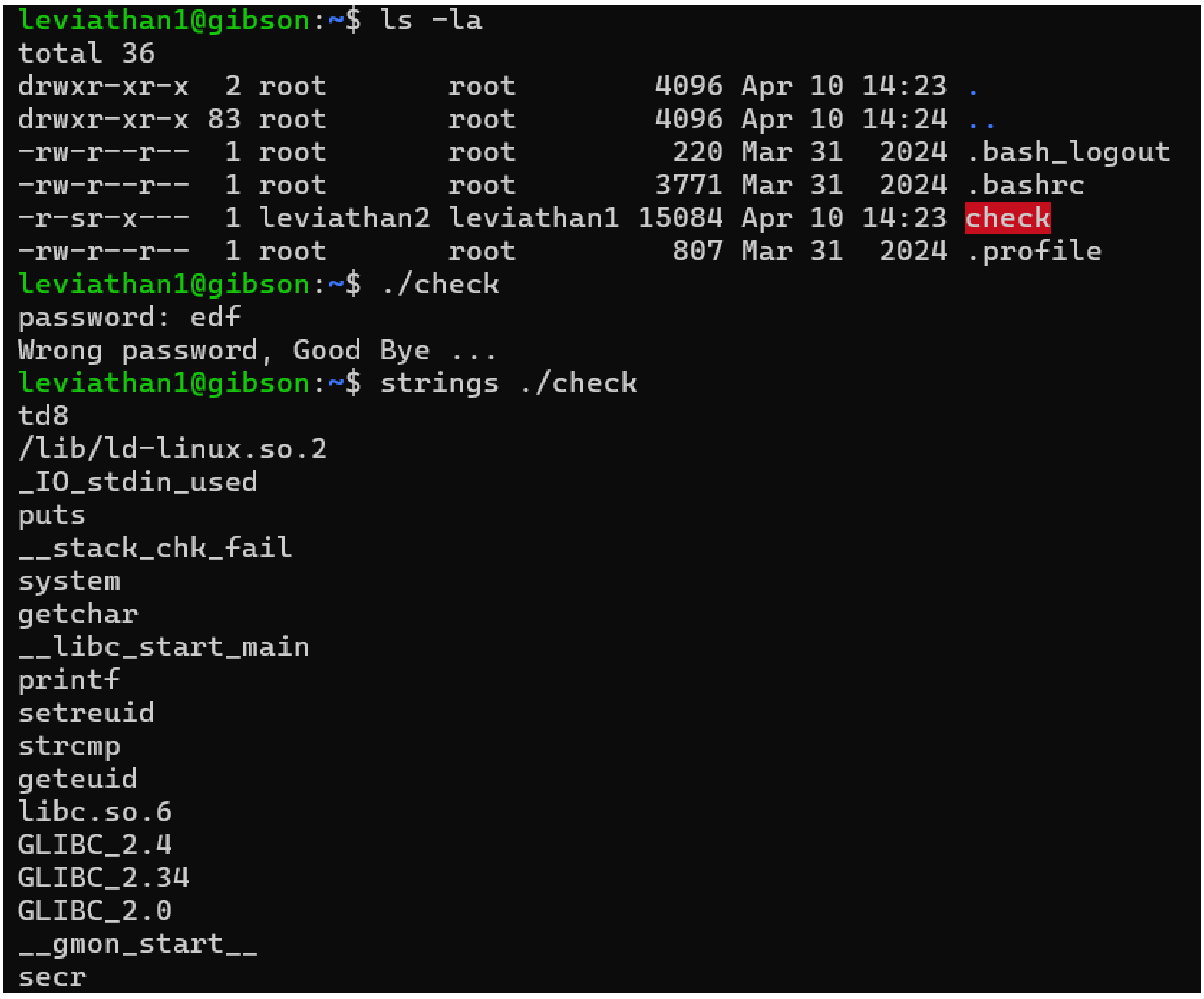
1. We connected to the server as leviathan1 using the password obtained from Level 0.
2. Listed files with `ls` and `ls -la` and found a binary executable (for example, "check").
3. We ran the program with `./check`, and it prompted for a password.
4. As guessing would not be efficient, we used `strings check` to read human-readable text from inside the binary.
5. Hidden within the binary output, we found a string that looked like the password.
6. We executed the program again (`./check`), entered the found password, and were given the password for leviathan2.
7. Finally, we used `ssh leviathan2@gibson` to log into the next level.

Tools Used:

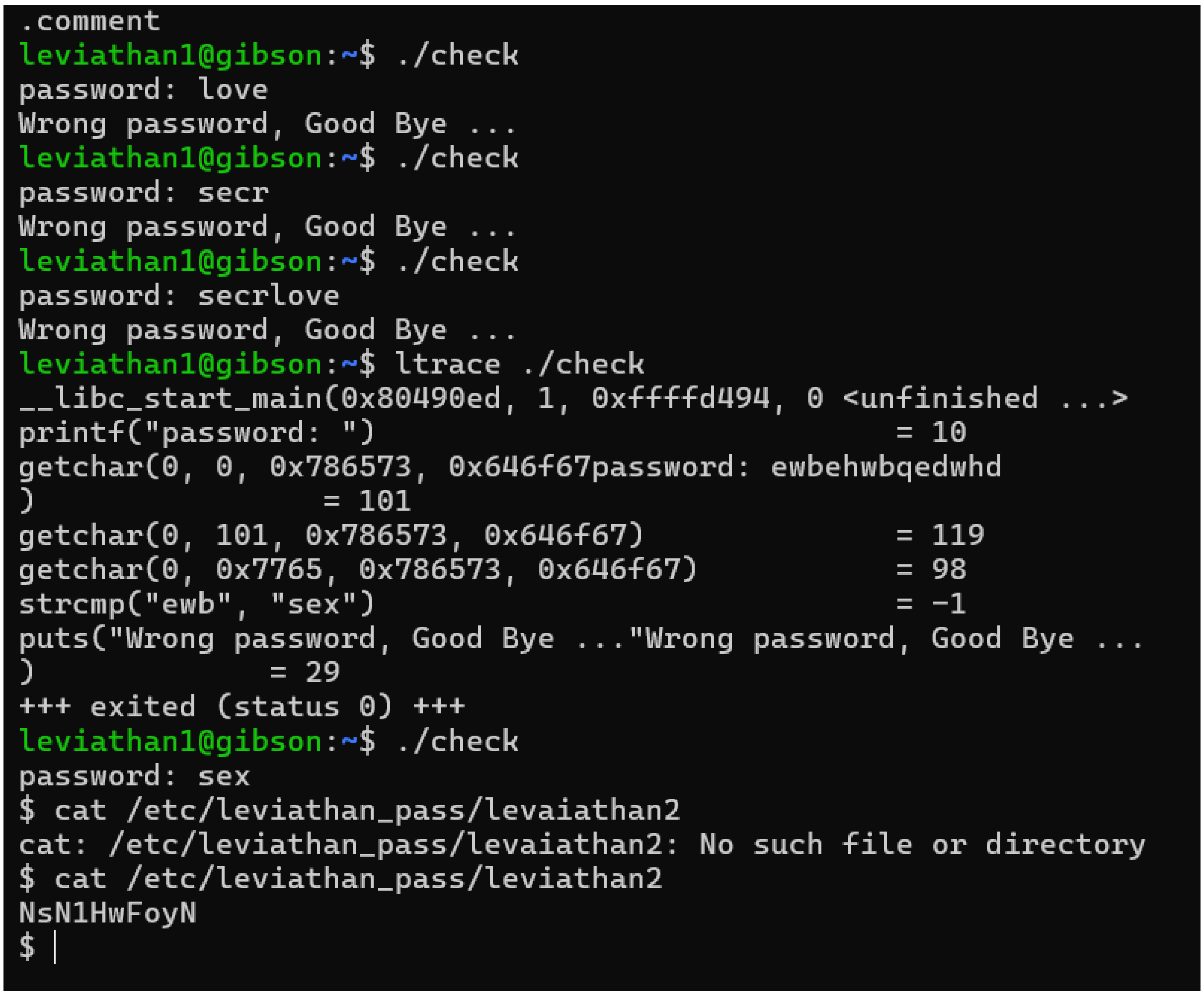
- `ls` and `ls -la` to explore available files.
- `./` to run executables.
- `strings` to reveal readable text inside a binary.
- `ssh` for moving between levels.

Logic Behind the Solution:

- Binary analysis in beginner CTFs usually hides passwords inside strings.
- `strings` is a very quick tool to pull readable information without needing advanced reverse engineering.
- Running the executable directly confirmed our findings.
- Correct password input unlocks the next level.



```
leviathan1@gibson:~$ ls -la
total 36
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan2 leviathan1 15084 Apr 10 14:23 check
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan1@gibson:~$ ./check
password: edf
Wrong password, Good Bye ...
leviathan1@gibson:~$ strings ./check
td8
/lib/ld-linux.so.2
_IO_stdin_used
puts
__stack_chk_fail
system
getchar
__libc_start_main
printf
setreuid
strcmp
geteuid
libc.so.6
GLIBC_2.4
GLIBC_2.34
GLIBC_2.0
__gmon_start__
secr
```



```
.comment
leviathan1@gibson:~$ ./check
password: love
Wrong password, Good Bye ...
leviathan1@gibson:~$ ./check
password: secr
Wrong password, Good Bye ...
leviathan1@gibson:~$ ./check
password: secrlove
Wrong password, Good Bye ...
leviathan1@gibson:~$ ltrace ./check
__libc_start_main(0x80490ed, 1, 0xffffd494, 0 <unfinished ...>
printf("password: ")                             = 10
getchar(0, 0, 0x786573, 0x646f67password: ewbehwbqedwhd
)                   = 101
getchar(0, 101, 0x786573, 0x646f67)              = 119
getchar(0, 0x7765, 0x786573, 0x646f67)           = 98
strcmp("ewb", "sex")                             = -1
puts("Wrong password, Good Bye ..."Wrong password, Good Bye ...
)                = 29
+++ exited (status 0) +++
leviathan1@gibson:~$ ./check
password: sex
$ cat /etc/leviathan_pass/levaiathan2
cat: /etc/leviathan_pass/levaiathan2: No such file or directory
$ cat /etc/leviathan_pass/leviathan2
NsN1HwFoyN
$ |
```

LEVEL 2

Objective:

To solve the Leviathan2 challenge, the task was to retrieve the password for the user 'leviathan3' by exploiting the SUID binary `printfile`.

Concepts and Tools Used:

1. SUID (Set User ID) Binary:

- The binary `printfile` is an SUID binary, which means it runs with the permissions of the file owner (in this case, `leviathan3`). This allows it to access resources that may be restricted for regular users.

- Checking the `printfile` binary using `ls -la` showed that it had the SUID bit set, and its owner is `leviathan3`, which is important for gaining access to the password.

2. Symlinks:

- A symbolic link (`ln -s`) was created to point to the `/etc/leviathan_pass/leviathan3` file, where the password for the next level is stored.

- This symlink effectively allows the `printfile` program to access the `leviathan3` password file even if it normally would not have permissions to do so.

3. Using `printfile` to Retrieve the Password:

- Running `printfile` on the symlinked file `test file.txt` (which points to `leviathan3`) printed the password for `leviathan3`.

- The password `f0n8h2iWLP` was revealed as the result of the `printfile` command.

4. Troubleshooting:

- While attempting to use `printfile`, you encountered an error (`/bin/cat: file.txt: No such file or directory`). This was due to a misconfiguration in the program's internal logic. However, running the command with the symlink correctly retrieved the password.

- You ensured that the symlink was pointing to the correct file by using the `ln -sf` command to force it to be linked properly.

Final Password:

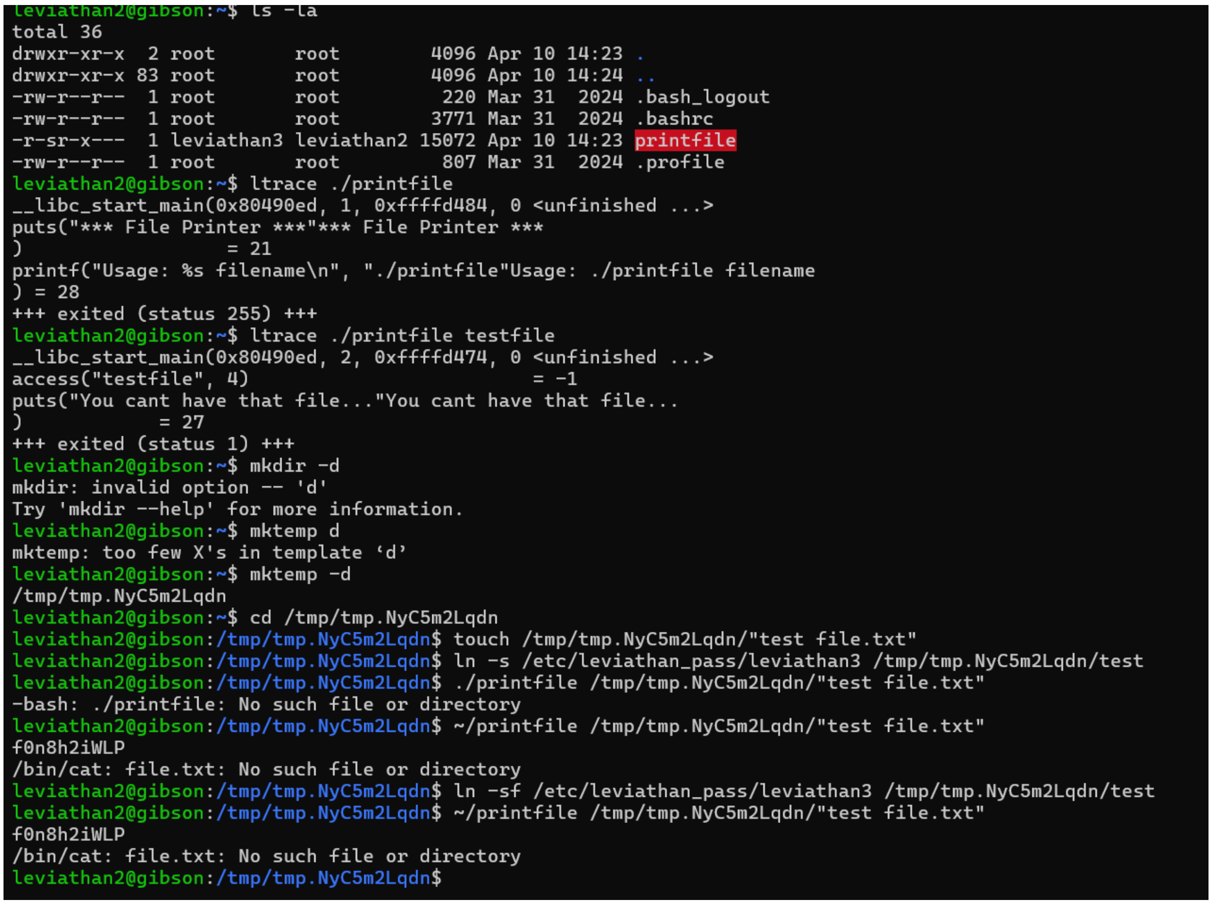
The password retrieved from `leviathan3` was `f0n8h2iWLP`.

Tools Used:

- `mktemp`: Used to create a temporary directory.
- `touch`: Used to create an empty test file.
- `ln -s`: Used to create a symbolic link to the target file (`leviathan3`).
- `~/printfile`: The SUID binary used to access the password file.
- `ls -la`: Used to check the file permissions and attributes.

Conclusion:

This challenge focused on understanding the implications of SUID binaries and how they can be exploited to read files normally restricted to the user that owns them. By creating a symlink to the target file and utilizing the `printfile` binary, the password for the next level was successfully retrieved.



```
leviathan2@gibson:~$ ls -la
total 36
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan3 leviathan2 15072 Apr 10 14:23 printfile
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan2@gibson:~$ ltrace ./printfile
__libc_start_main(0x80490ed, 1, 0xffffd484, 0 <unfinished ...>
puts("*** File Printer ***"*** File Printer ***
)                  = 21
printf("Usage: %s filename\n", "./printfile"Usage: ./printfile filename
) = 28
+++ exited (status 255) +++
leviathan2@gibson:~$ ltrace ./printfile testfile
__libc_start_main(0x80490ed, 2, 0xffffd474, 0 <unfinished ...>
access("testfile", 4)                            = -1
puts("You cant have that file..."You cant have that file...
)            = 27
+++ exited (status 1) +++
leviathan2@gibson:~$ mkdir -d
mkdir: invalid option -- 'd'
Try 'mkdir --help' for more information.
leviathan2@gibson:~$ mktemp d
mktemp: too few X's in template 'd'
leviathan2@gibson:~$ mktemp -d
/tmp/tmp.NyC5m2Lqdn
leviathan2@gibson:~$ cd /tmp/tmp.NyC5m2Lqdn
leviathan2@gibson:/tmp/tmp.NyC5m2Lqdn$ touch /tmp/tmp.NyC5m2Lqdn/"test file.txt"
leviathan2@gibson:/tmp/tmp.NyC5m2Lqdn$ ln -s /etc/leviathan_pass/leviathan3 /tmp/tmp.NyC5m2Lqdn/test
leviathan2@gibson:/tmp/tmp.NyC5m2Lqdn$ ./printfile /tmp/tmp.NyC5m2Lqdn/"test file.txt"
-bash: ./printfile: No such file or directory
leviathan2@gibson:/tmp/tmp.NyC5m2Lqdn$ ~/printfile /tmp/tmp.NyC5m2Lqdn/"test file.txt"
f0n8h2iWLP
/bin/cat: file.txt: No such file or directory
leviathan2@gibson:/tmp/tmp.NyC5m2Lqdn$ ln -sf /etc/leviathan_pass/leviathan3 /tmp/tmp.NyC5m2Lqdn/test
leviathan2@gibson:/tmp/tmp.NyC5m2Lqdn$ ~/printfile /tmp/tmp.NyC5m2Lqdn/"test file.txt"
f0n8h2iWLP
/bin/cat: file.txt: No such file or directory
leviathan2@gibson:/tmp/tmp.NyC5m2Lqdn$
```

LEVEL 3

Initial Exploration:

1. I first listed the files in the current directory using `ls -la` to confirm the presence of the `level3` binary, which was setuid and owned by `leviathan4`. This indicated that the program would likely have some special privileges upon execution.

Password Attempt:

2. I ran the `level3` binary and was prompted to enter a password.

- First, I tried an incorrect password: `ewrfwewfwfr`. The program returned `bzzzzzzzzap. WRONG`.

- Next, I tried another incorrect password: `kakaka`, but the program returned the same message: `bzzzzzzzzap. WRONG`.

Vulnerability Discovery:

3. Upon inspecting the behavior of the program using `ltrace`, I observed that the program compared the user input against the string "snlprintf" in the following line:

strcmp("wedw\n", "snlprintf\n") = 1

This indicated that the program expected a specific password.

Exploitation:

4. I tried the correct password `snlprintf`, and the program responded with `[You've got shell]!`, indicating successful exploitation.

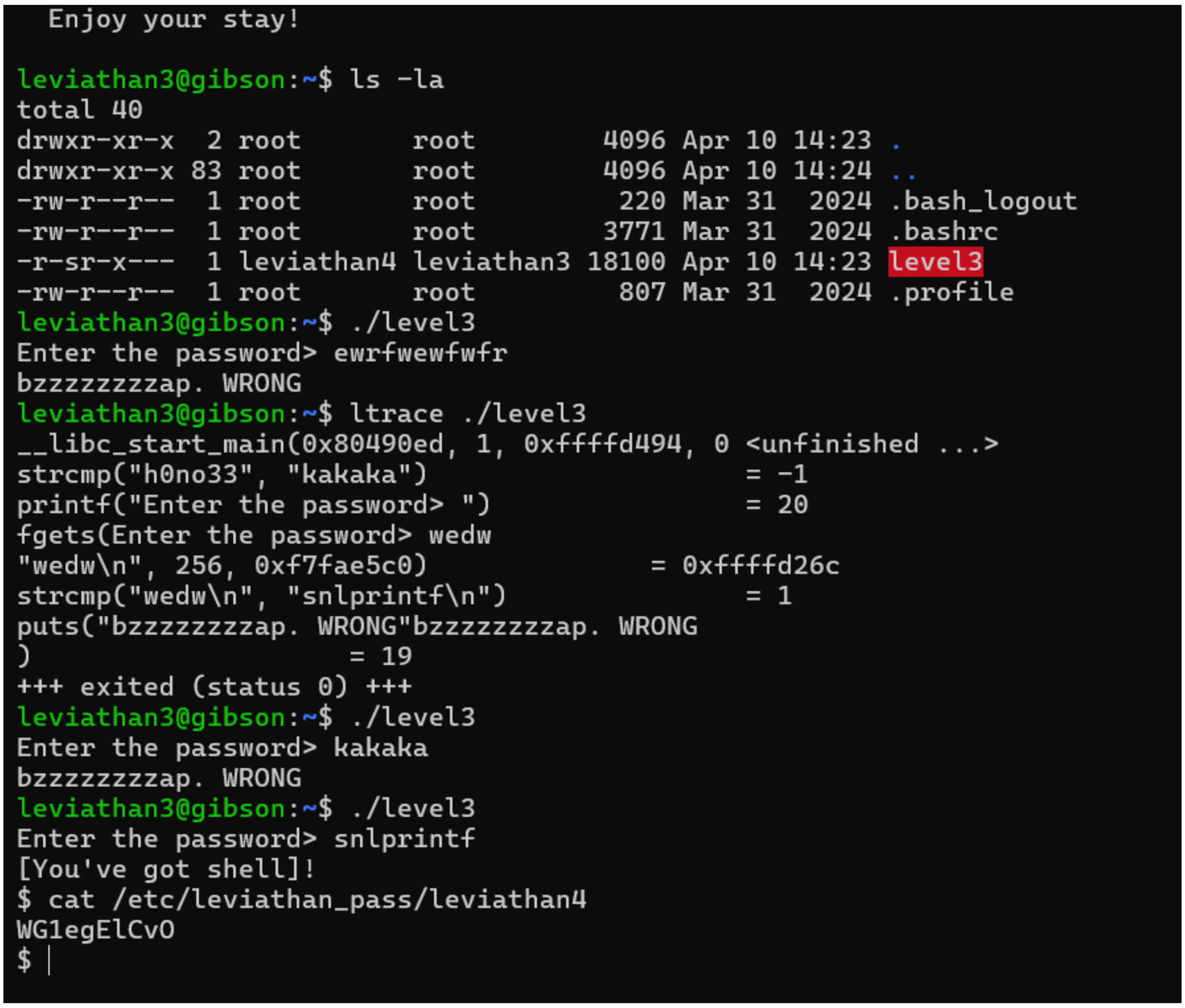
Retrieving the Next Password:

5. After gaining shell access, I ran the following command to retrieve the password for the next level:

cat /etc/leviathan_pass/leviathan4

Conclusion:

- The program had a hardcoded password (`snlprintf`) that granted access to a shell



```
  Enjoy your stay!

leviathan3@gibson:~$ ls -la
total 40
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan4 leviathan3 18100 Apr 10 14:23 level3
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan3@gibson:~$ ./level3
Enter the password> ewrfwewfwfr
bzzzzzzzzap. WRONG
leviathan3@gibson:~$ ltrace ./level3
__libc_start_main(0x80490ed, 1, 0xffffd494, 0 <unfinished ...>
strcmp("h0no33", "kakaka")                          = -1
printf("Enter the password> ")                      = 20
fgets(Enter the password> wedw
"wedw\n", 256, 0xf7fae5c0)                    = 0xffffd26c
strcmp("wedw\n", "snlprintf\n")                     = 1
puts("bzzzzzzzzap. WRONG"bzzzzzzzzap. WRONG
)                         = 19
+++ exited (status 0) +++
leviathan3@gibson:~$ ./level3
Enter the password> kakaka
bzzzzzzzzap. WRONG
leviathan3@gibson:~$ ./level3
Enter the password> snlprintf
[You've got shell]!
$ cat /etc/leviathan_pass/leviathan4
WG1egElCvO
$ |
```

LEVEL 4

Level Overview:

The challenge at Level 4 required finding the password hidden inside a binary string located in the `.trash` directory.

Steps Taken:

1. Navigating to `.trash` Directory:

   - First, I navigated to the `.trash` directory using the command:

```
cd .trash
```

2. Listing the Contents of `.trash`:

   - I listed the contents of the `.trash` directory:

```
ls -la
```

   - The directory contains a subdirectory `bin`, which might hold the password in some form.

3. Navigating to `bin` Directory:

   - I entered the `bin` directory:

```
cd bin
```

4. Viewing the Binary String:

   - I displayed the content of the `bin` file, which contained a binary string:

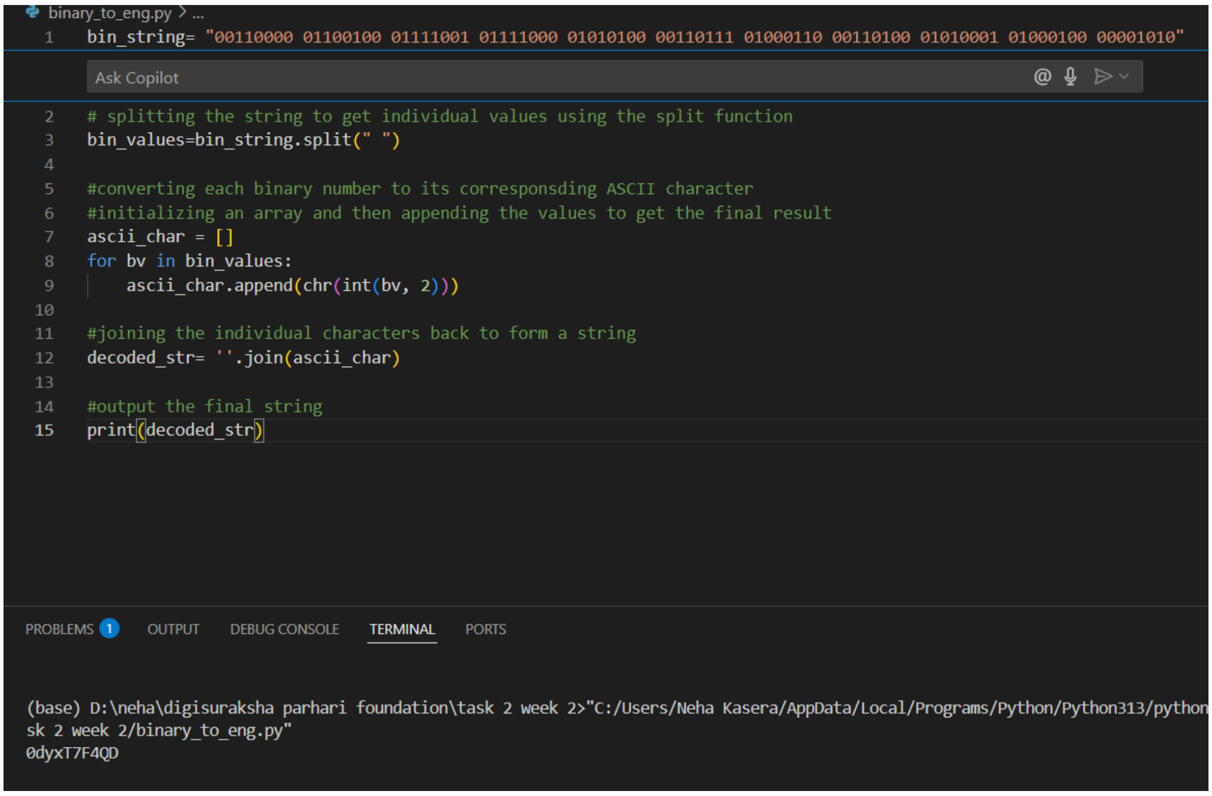
```
cat bin
```

   - The binary string found inside `bin` was:

```
00110000 01100100 01111001 01111000 01010100 00110111 01000110 00110100 01010001
 01000100 00001010
```

5. Decoding the Binary String:

   - The binary string was then converted to text. I used a Python script to convert the binary string into the ASCII text.



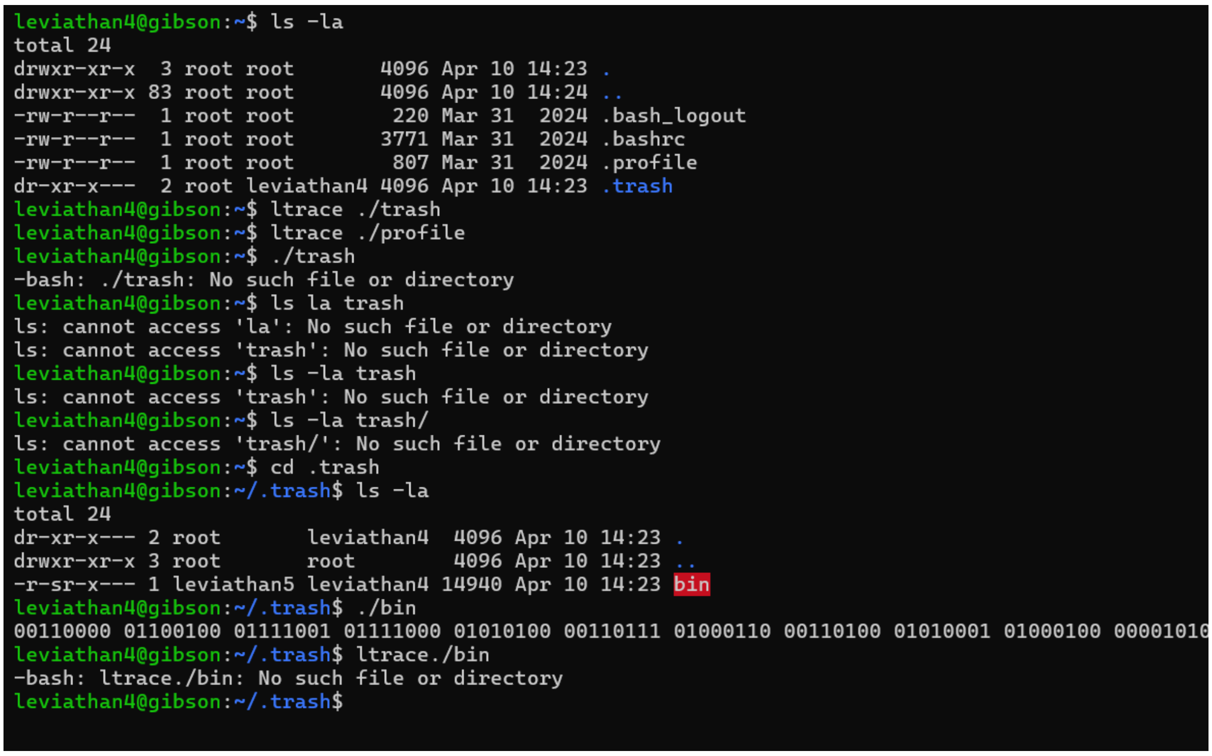
```python
bin_string= "00110000 01100100 01111001 01111000 01010100 00110111 01000110 00110100 01010001 01000100 00001010"
# splitting the string to get individual values using the split function
bin_values=bin_string.split(" ")

#converting each binary number to its corresponsding ASCII character
#initializing an array and then appending the values to get the final result
ascii_char = []
for bv in bin_values:
    ascii_char.append(chr(int(bv, 2)))

#joining the individual characters back to form a string
decoded_str= ''.join(ascii_char)

#output the final string
print(decoded_str)
```

```
(base) D:\neha\digisuraksha parhari foundation\task 2 week 2>"C:/Users/Neha Kasera/AppData/Local/Programs/Python/Python313/python
sk 2 week 2/binary_to_eng.py"
0dyxT7F4QD
```



```
leviathan4@gibson:~$ ls -la
total 24
drwxr-xr-x  3 root root       4096 Apr 10 14:23 .
drwxr-xr-x 83 root root       4096 Apr 10 14:24 ..
-rw-r--r--  1 root root        220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root root       3771 Mar 31  2024 .bashrc
-rw-r--r--  1 root root        807 Mar 31  2024 .profile
dr-xr-x---  2 root leviathan4 4096 Apr 10 14:23 .trash
leviathan4@gibson:~$ ltrace ./trash
leviathan4@gibson:~$ ltrace ./profile
leviathan4@gibson:~$ ./trash
-bash: ./trash: No such file or directory
leviathan4@gibson:~$ ls la trash
ls: cannot access 'la': No such file or directory
ls: cannot access 'trash': No such file or directory
leviathan4@gibson:~$ ls -la trash
ls: cannot access 'trash': No such file or directory
leviathan4@gibson:~$ ls -la trash/
ls: cannot access 'trash/': No such file or directory
leviathan4@gibson:~$ cd .trash
leviathan4@gibson:~/.trash$ ls -la
total 24
dr-xr-x--- 2 root       leviathan4  4096 Apr 10 14:23 .
drwxr-xr-x 3 root       root        4096 Apr 10 14:23 ..
-r-sr-x--- 1 leviathan5 leviathan4 14940 Apr 10 14:23 bin
leviathan4@gibson:~/.trash$ ./bin
00110000 01100100 01111001 01111000 01010100 00110111 01000110 00110100 01010001 01000100 00001010
leviathan4@gibson:~/.trash$ ltrace./bin
-bash: ltrace./bin: No such file or directory
leviathan4@gibson:~/.trash$
```

LEVEL 5

Steps Taken:

1. List the files in the current directory:

Command: ls -la

This command was used to list the contents of the current working directory. By running this command, we were able to verify the existence of the leviathan5 executable and other relevant files such as .bashrc, .bash_logout, and .profile. This also helped confirm that there were no immediate issues with missing files in the directory.

2. Run the leviathan5 program:

Command: ./leviathan5

When running the program for the first time, it returned the error message: Cannot find /tmp/file.log. This error indicated that the program was attempting to access a file (/tmp/file.log) that did not exist at that location. This was the key observation to understand how to bypass the error.

3. Check for a potential typo in the command:

Command: lrace leviathan5

This command was mistakenly typed as lrace, which is not a valid command. The terminal helpfully suggested the correct command ltrace instead, which is used for tracing library calls and system calls made by a program.

4. Use ltrace to analyze the program's behavior:

Command: ltrace ./leviathan5

Running ltrace on the leviathan5 program revealed that it was trying to open the file /tmp/file.log using the fopen system call. Since the file didn't exist, the program exited and printed the message: Cannot find /tmp/file.log. This confirmed that the program's flow depends on this missing file.

5. Check if /tmp/file.log exists:

Command: ls -s /etc/leviathan_pass/leviathan6 /tmp/file.log

This command was used to check if /tmp/file.log existed and also verify the permissions of the target file (/etc/leviathan_pass/leviathan6). The result showed that /tmp/file.log did not exist, so we needed to create it.

6. Create a symbolic link to /tmp/file.log:

Command: ln -s /etc/leviathan_pass/leviathan6 /tmp/file.log

At this step, we created a symbolic link from /tmp/file.log to the actual password file located at /etc/leviathan_pass/leviathan6. The ln -s command creates a symlink, which tricks the program into thinking the file it is looking for (/tmp/file.log) exists and points to the actual password file.

7. Run the leviathan5 program again:

Command: ./leviathan5

After creating the symlink, we ran the leviathan5 program again. This time, instead of failing with an error message, it successfully retrieved the password from the symlink and printed it: szo7HDB88w.

Tools Used:

ls -la:

Used to list files in the current directory to verify the existence of the necessary files.

ltrace:

Used to trace system calls and library calls made by the leviathan5 program. This helped identify that the program was trying to open /tmp/file.log and failed because it didn't exist.

ln -s:

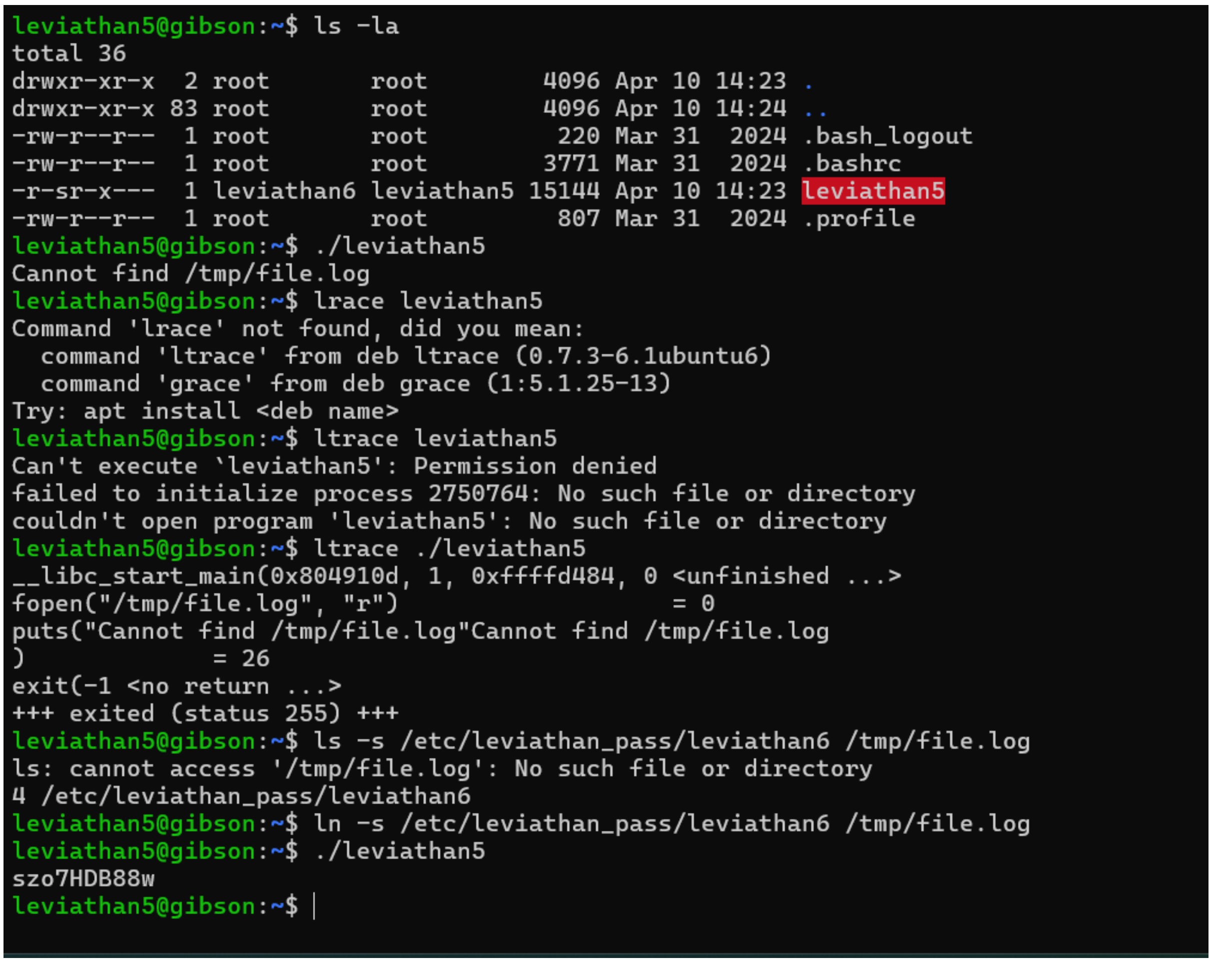
Used to create a symbolic link (symlink) to /tmp/file.log, pointing it to the actual password file /etc/leviathan_pass/leviathan6. This manipulation of the file system allowed us to bypass the error and access the password.

Logic Behind the Solution:

The leviathan5 program is searching for a file at /tmp/file.log but fails to find it. The error message indicates that the program relies on this file being present to proceed. By examining the program’s behavior using ltrace, we observed that the program performs a fopen call on /tmp/file.log.

The solution lies in creating a symbolic link at /tmp/file.log that points to the actual password file, located at /etc/leviathan_pass/leviathan6. This effectively "tricks" the program into believing that the missing file exists, allowing it to proceed and print the password when the symlink is accessed.

By following these steps, we were able to obtain the password szo7HDB88w and move to the next level.



```
leviathan5@gibson:~$ ls -la
total 36
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan6 leviathan5 15144 Apr 10 14:23 leviathan5
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan5@gibson:~$ ./leviathan5
Cannot find /tmp/file.log
leviathan5@gibson:~$ lrace leviathan5
Command 'lrace' not found, did you mean:
  command 'ltrace' from deb ltrace (0.7.3-6.1ubuntu6)
  command 'grace' from deb grace (1:5.1.25-13)
Try: apt install <deb name>
leviathan5@gibson:~$ ltrace leviathan5
Can't execute `leviathan5': Permission denied
failed to initialize process 2750764: No such file or directory
couldn't open program 'leviathan5': No such file or directory
leviathan5@gibson:~$ ltrace ./leviathan5
__libc_start_main(0x804910d, 1, 0xffffd484, 0 <unfinished ...>
fopen("/tmp/file.log", "r")                      = 0
puts("Cannot find /tmp/file.log"Cannot find /tmp/file.log
)              = 26
exit(-1 <no return ...>
+++ exited (status 255) +++
leviathan5@gibson:~$ ls -s /etc/leviathan_pass/leviathan6 /tmp/file.log
ls: cannot access '/tmp/file.log': No such file or directory
4 /etc/leviathan_pass/leviathan6
leviathan5@gibson:~$ ln -s /etc/leviathan_pass/leviathan6 /tmp/file.log
leviathan5@gibson:~$ ./leviathan5
szo7HDB88w
leviathan5@gibson:~$ |
```

LEVEL 6

Tools Used:

- Bash: Utilized for scripting the brute force approach.

- Nano: Used for creating and editing the brute force script.

- Leviathan Binary (`leviathan6`): The binary requiring a 4-digit code for authentication to proceed to the next level.

Objective:

The binary `leviathan6` requires a 4-digit code to authenticate and progress to the next level. Rather than manually trying every combination, we used a Bash script to automate the brute force attack, trying all combinations from `0000` to `9999`.

Step-by-Step Approach:

1. Create a Working Directory:

   To keep everything organized, we first created a directory `/tmp/bashbruteforce` in which to store the script.

```
mkdir /tmp/bashbruteforce

cd /tmp/bashbruteforce
```

2. Write the Brute Force Script: We wrote a simple Bash script brute.sh that iterates over all possible 4-digit combinations from 0000 to 9999. Each iteration passes the 4-digit code to the leviathan6 binary to check for a valid code.

The script brute.sh contains the following code:

```
#!/bin/bash

for i in {0000..9999}

do

 ~/leviathan6 $i

done
```

3. Make the Script Executable: After writing the script, we saved it using nano and made it executable with the chmod command:

```
chmod +x brute.sh
```

4. Run the Brute Force Script: We executed the brute force script to try every possible 4-digit combination. The script ran through all combinations, checking each one against the leviathan6 binary.

```
./brute.sh
```

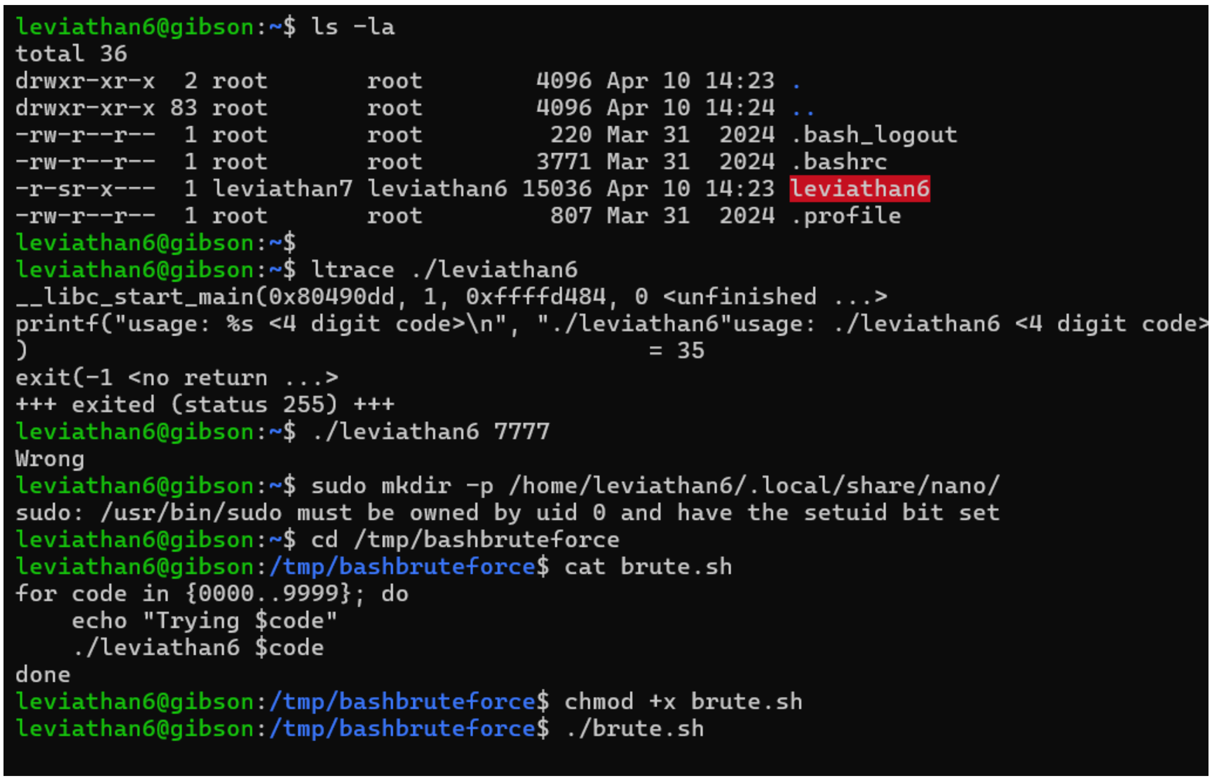
5. Wait for the Script to Complete: After running the script, it took about 20 seconds to cycle through all combinations. Once the correct code was found, the script successfully granted access to the next level.

Result:

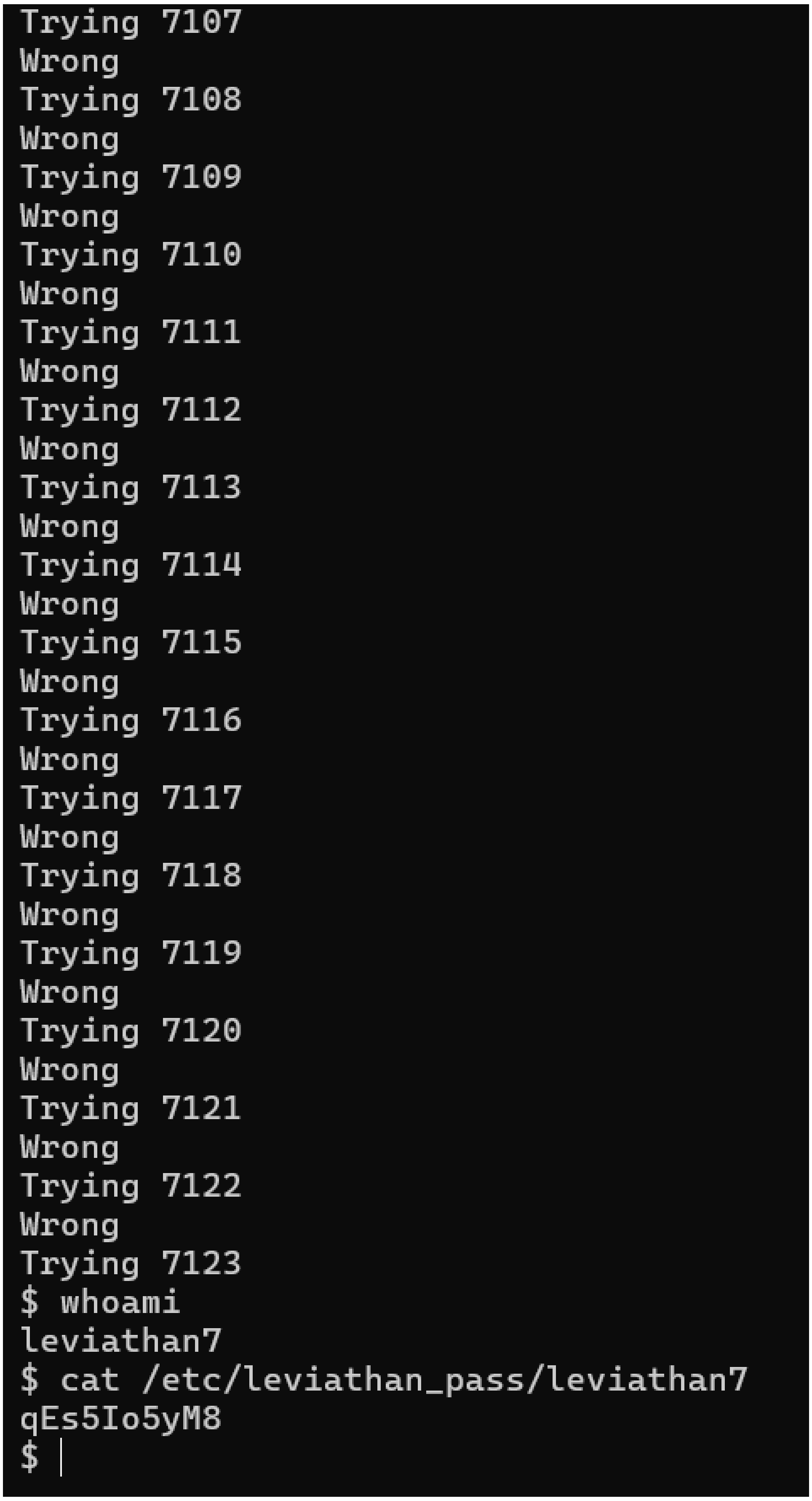
The brute force script was successful in discovering the correct 4-digit code, enabling us to access leviathan7.

Conclusion:

The brute force approach using a Bash script proved to be an efficient solution to bypassing the 4-digit code in this challenge. This method can be applied to similar challenges involving fixed-input authentication systems where the possible combinations are known and finite.



```
leviathan6@gibson:~$ ls -la
total 36
drwxr-xr-x  2 root       root        4096 Apr 10 14:23 .
drwxr-xr-x 83 root       root        4096 Apr 10 14:24 ..
-rw-r--r--  1 root       root         220 Mar 31  2024 .bash_logout
-rw-r--r--  1 root       root        3771 Mar 31  2024 .bashrc
-r-sr-x---  1 leviathan7 leviathan6 15036 Apr 10 14:23 leviathan6
-rw-r--r--  1 root       root         807 Mar 31  2024 .profile
leviathan6@gibson:~$
leviathan6@gibson:~$ ltrace ./leviathan6
__libc_start_main(0x80490dd, 1, 0xffffd484, 0 <unfinished ...>
printf("usage: %s <4 digit code>\n", "./leviathan6"usage: ./leviathan6 <4 digit code>
)                                             = 35
exit(-1 <no return ...>
+++ exited (status 255) +++
leviathan6@gibson:~$ ./leviathan6 7777
Wrong
leviathan6@gibson:~$ sudo mkdir -p /home/leviathan6/.local/share/nano/
sudo: /usr/bin/sudo must be owned by uid 0 and have the setuid bit set
leviathan6@gibson:~$ cd /tmp/bashbruteforce
leviathan6@gibson:/tmp/bashbruteforce$ cat brute.sh
for code in {0000..9999}; do
    echo "Trying $code"
    ./leviathan6 $code
done
leviathan6@gibson:/tmp/bashbruteforce$ chmod +x brute.sh
leviathan6@gibson:/tmp/bashbruteforce$ ./brute.sh
```



```
Trying 7107
Wrong
Trying 7108
Wrong
Trying 7109
Wrong
Trying 7110
Wrong
Trying 7111
Wrong
Trying 7112
Wrong
Trying 7113
Wrong
Trying 7114
Wrong
Trying 7115
Wrong
Trying 7116
Wrong
Trying 7117
Wrong
Trying 7118
Wrong
Trying 7119
Wrong
Trying 7120
Wrong
Trying 7121
Wrong
Trying 7122
Wrong
Trying 7123
$ whoami
leviathan7
$ cat /etc/leviathan_pass/leviathan7
qEs5Io5yM8
$ 
```