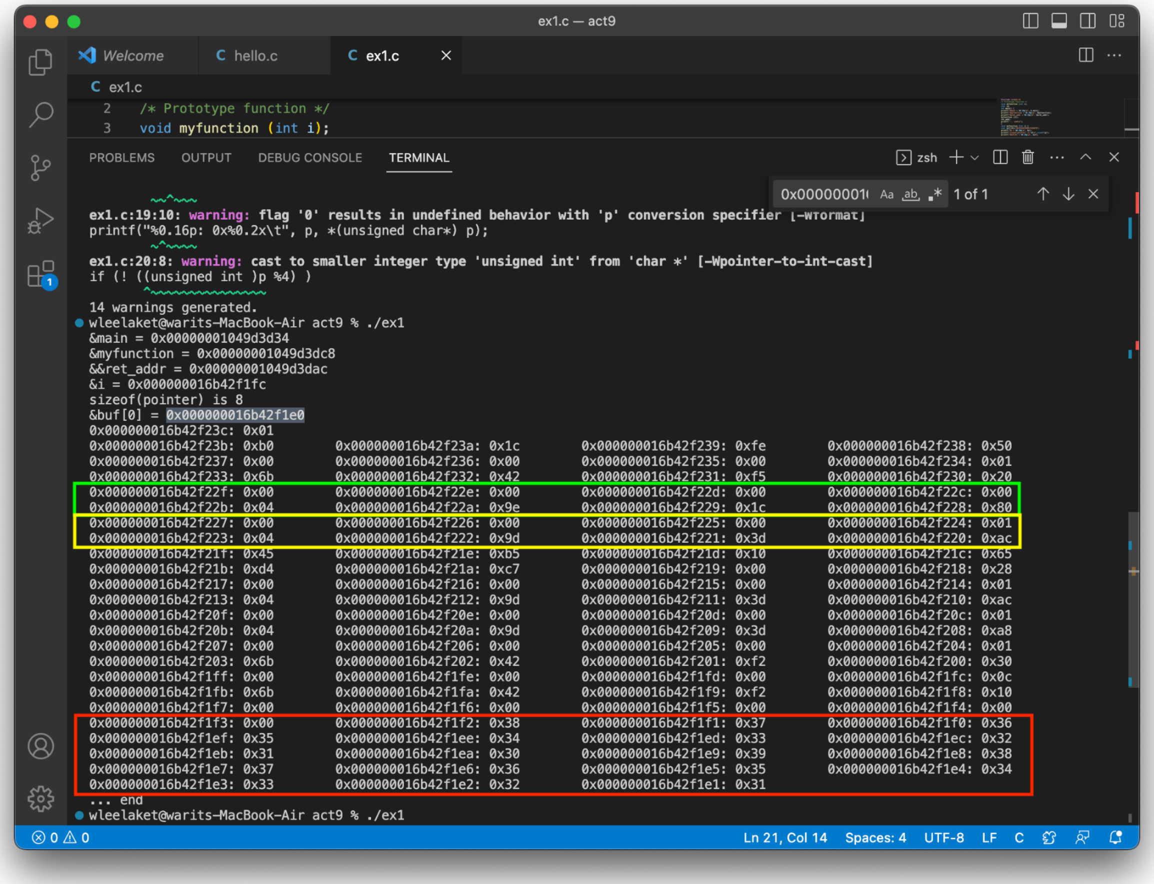
## Ex1

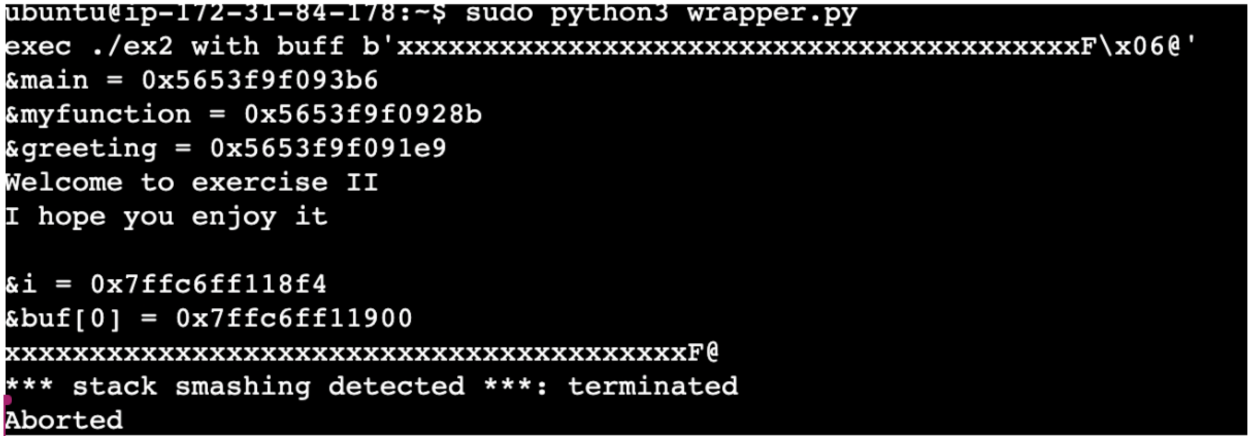


```
/* Prototype function */
void myfunction (int i);
```

```
      ~~^~~~
ex1.c:19:10: warning: flag '0' results in undefined behavior with 'p' conversion specifier [-Wformat]
printf("%0.16p: 0x%0.2x\t", p, *(unsigned char*) p);
      ~^~~~~
ex1.c:20:8: warning: cast to smaller integer type 'unsigned int' from 'char *' [-Wpointer-to-int-cast]
if (! ((unsigned int )p %4) )
     ^~~~~~~~~~~~~~~~~~

14 warnings generated.
wleelaket@warits-MacBook-Air act9 % ./ex1
&main = 0x00000001049d3d34
&myfunction = 0x00000001049d3dc8
&&ret_addr = 0x00000001049d3dac
&i = 0x000000016b42f1fc
sizeof(pointer) is 8
&buf[0] = 0x000000016b42f1e0
0x000000016b42f23c: 0x01
0x000000016b42f23b: 0xb0        0x000000016b42f23a: 0x1c        0x000000016b42f239: 0xfe        0x000000016b42f238: 0x50
0x000000016b42f237: 0x00        0x000000016b42f236: 0x00        0x000000016b42f235: 0x00        0x000000016b42f234: 0x01
0x000000016b42f233: 0x6b        0x000000016b42f232: 0x42        0x000000016b42f231: 0xf5        0x000000016b42f230: 0x20
0x000000016b42f22f: 0x00        0x000000016b42f22e: 0x00        0x000000016b42f22d: 0x00        0x000000016b42f22c: 0x00
0x000000016b42f22b: 0x04        0x000000016b42f22a: 0x9e        0x000000016b42f229: 0x1c        0x000000016b42f228: 0x80
0x000000016b42f227: 0x00        0x000000016b42f226: 0x00        0x000000016b42f225: 0x00        0x000000016b42f224: 0x01
0x000000016b42f223: 0x04        0x000000016b42f222: 0x9d        0x000000016b42f221: 0x3d        0x000000016b42f220: 0xac
0x000000016b42f21f: 0x45        0x000000016b42f21e: 0xb5        0x000000016b42f21d: 0x10        0x000000016b42f21c: 0x65
0x000000016b42f21b: 0xd4        0x000000016b42f21a: 0xc7        0x000000016b42f219: 0x00        0x000000016b42f218: 0x28
0x000000016b42f217: 0x00        0x000000016b42f216: 0x00        0x000000016b42f215: 0x00        0x000000016b42f214: 0x01
0x000000016b42f213: 0x04        0x000000016b42f212: 0x9d        0x000000016b42f211: 0x3d        0x000000016b42f210: 0xac
0x000000016b42f20f: 0x00        0x000000016b42f20e: 0x00        0x000000016b42f20d: 0x00        0x000000016b42f20c: 0x01
0x000000016b42f20b: 0x04        0x000000016b42f20a: 0x9d        0x000000016b42f209: 0x3d        0x000000016b42f208: 0xa8
0x000000016b42f207: 0x00        0x000000016b42f206: 0x00        0x000000016b42f205: 0x00        0x000000016b42f204: 0x01
0x000000016b42f203: 0x6b        0x000000016b42f202: 0x42        0x000000016b42f201: 0xf2        0x000000016b42f200: 0x30
0x000000016b42f1ff: 0x00        0x000000016b42f1fe: 0x00        0x000000016b42f1fd: 0x00        0x000000016b42f1fc: 0x0c
0x000000016b42f1fb: 0x6b        0x000000016b42f1fa: 0x42        0x000000016b42f1f9: 0xf2        0x000000016b42f1f8: 0x10
0x000000016b42f1f7: 0x00        0x000000016b42f1f6: 0x00        0x000000016b42f1f5: 0x00        0x000000016b42f1f4: 0x00
0x000000016b42f1f3: 0x00        0x000000016b42f1f2: 0x38        0x000000016b42f1f1: 0x37        0x000000016b42f1f0: 0x36
0x000000016b42f1ef: 0x35        0x000000016b42f1ee: 0x34        0x000000016b42f1ed: 0x33        0x000000016b42f1ec: 0x32
0x000000016b42f1eb: 0x31        0x000000016b42f1ea: 0x30        0x000000016b42f1e9: 0x39        0x000000016b42f1e8: 0x38
0x000000016b42f1e7: 0x37        0x000000016b42f1e6: 0x36        0x000000016b42f1e5: 0x35        0x000000016b42f1e4: 0x34
0x000000016b42f1e3: 0x33        0x000000016b42f1e2: 0x32        0x000000016b42f1e1: 0x31
... end
wleelaket@warits-MacBook-Air act9 % ./ex1
```

## Ex2



```
ubuntu@ip-172-31-84-178:~$ sudo python3 wrapper.py
exec ./ex2 with buff b'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxF\x06@'
&main = 0x5653f9f093b6
&myfunction = 0x5653f9f0928b
&greeting = 0x5653f9f091e9
Welcome to exercise II
I hope you enjoy it

&i = 0x7ffc6ff118f4
&buf[0] = 0x7ffc6ff11900
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxF@
*** stack smashing detected ***: terminated
Aborted
```

**Ex3**

I couldn't get the prompt to display prettily, but as you can see. Inside the red triangle is the result from curling which indicates that I managed to get into the shell()

```
ubuntu@ip-172-31-84-178:~$ nc -l -p 60000 -c ./victim
&main = 0x00000000004013da
&vulnerable = 0x0000000000401383
&retpoint = 0x0000000000401513
&shell = 0x0000000000401236
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   477  100   477    0     0    557      0 --:--:-- --:--:-- --:--:--   557
```

4. From exercise 2 and 3 , can you explode the buffer-overflow attack even when the canary-style protection is activated? Please explain your analysis.

Ans : Canary protection is the method for preventing buffer-overflow attack, therefore if we enabled/activated canary protection during exercise 2 and 3, we wouldn't be able to attack successfully.
But there is some possible solution to perform a successful attack even with canary protection enabled. We have to find the canary value and address, then add the value at the specified address. This way we can pretend and bypass the canary protection.

5. Questions

1. Do you think that exploiting buffer-overflow attacks is trivial? Please justify your answer. (i.e. Is it trivial to write a program to exploit buffer-overflow attacks in a server ?)

   No, it's not trivial because

   1. buffer overflow is one of the most-used methods by attackers. It can be used for creating malicious programs like worms

   2. It is dangerous. From this activity, we can see how we can use buffer-overflow to execute shell commands

2. As a programmer, is it possible to avoid buffer overflow in your program (write secure code that is not vulnerable to such attack)? Explain your strategy.
   - Use up-to-date compiler
   - Be extra careful when handling input i.e. not specifying static input size
   - Try to heap over the stack to make the address to be guessed harder.
   - Avoid using vulnerable functions i.e. strcpy, strcat