PWN的题解

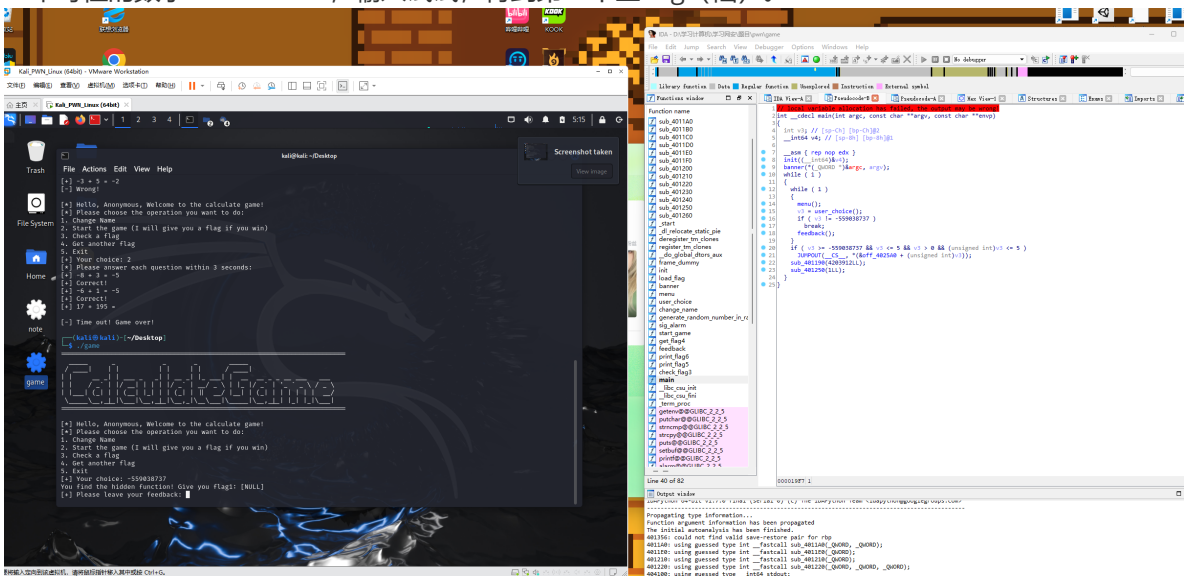
#

0.使用ida打开game，发现一份礼物

```
.data:00000000004040CC magic_number    dd 0AABBAABBh           ; DATA XREF: get_flag4+39↑r
.data:00000000004040D0                 public flag0
.data:00000000004040D0 flag0           db 'flag{7h1s_i5_4_91ft_f0r_y0u}',0
.data:00000000004040D0 _data           ends
.data:00000000004040D0
```

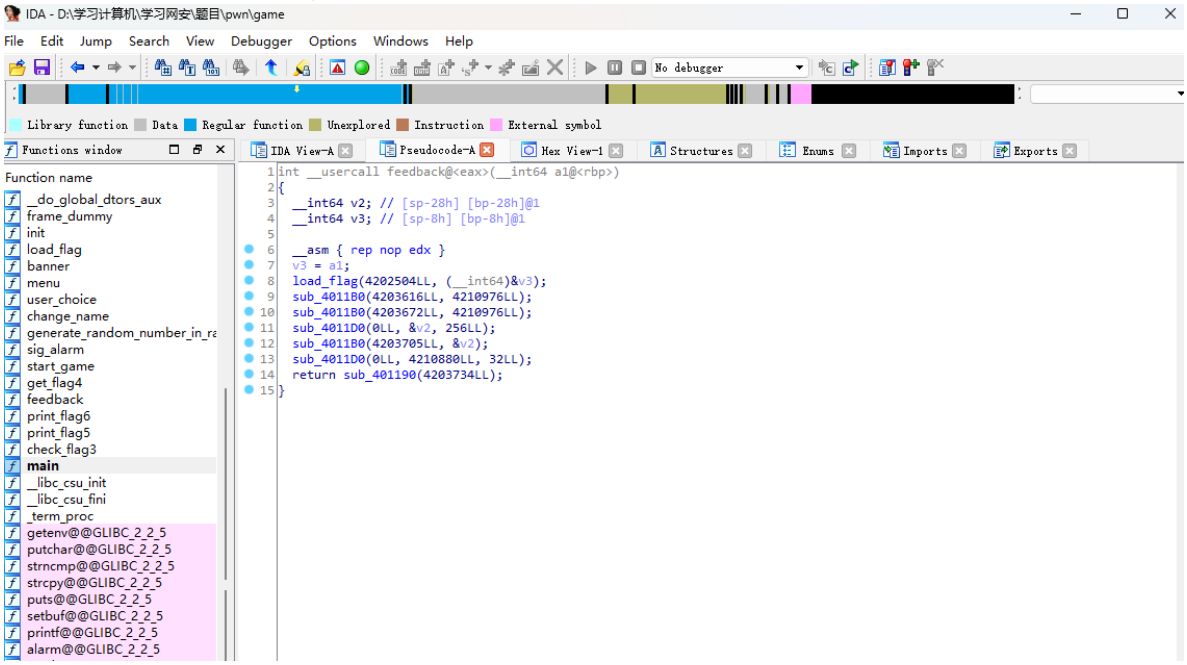
flag{7h1s_i5_4_91ft_f0r_y0u}

1.首先使用ida打开game，找到main函数，同时使用kali打开game，然后我们开始尝试。上来就看到一串奇怪的数字-559038737，输入试试，得到第一个空flag（恼）。



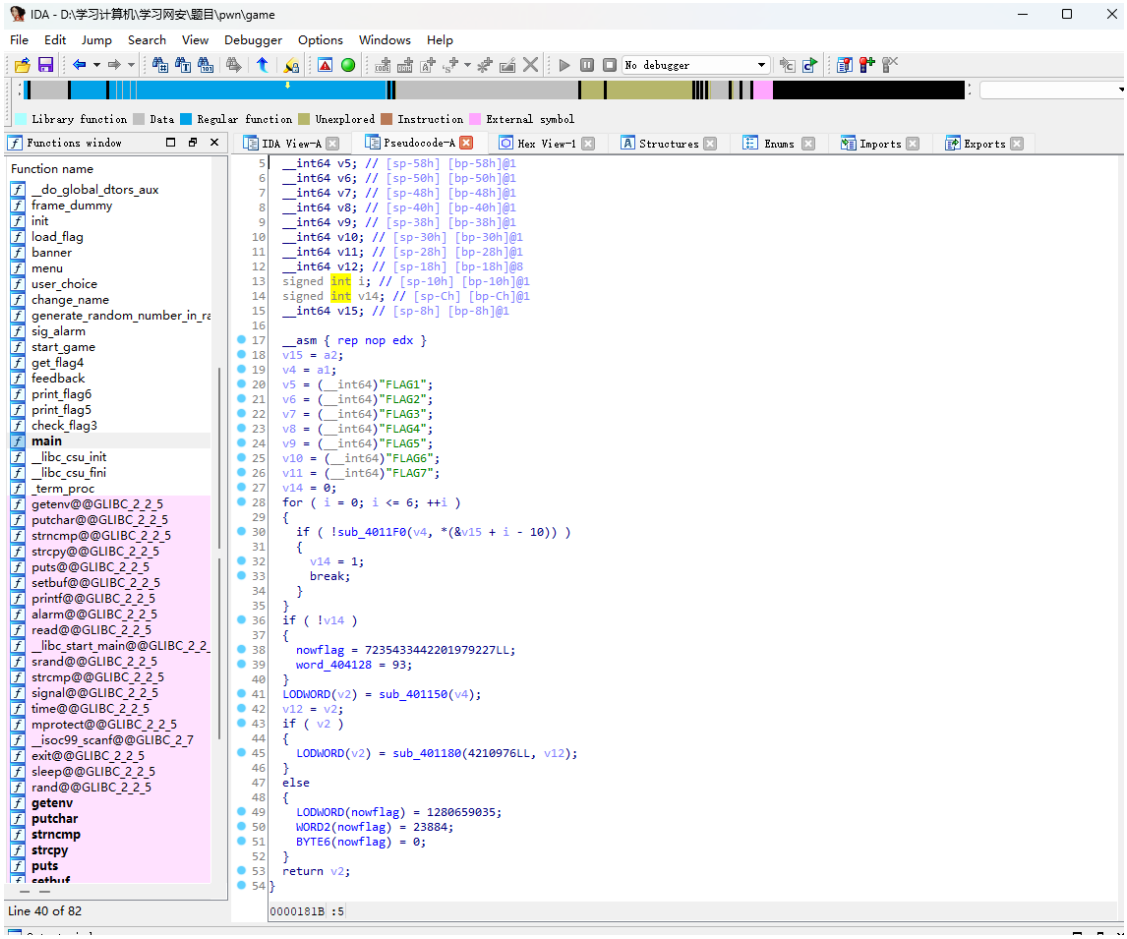
下面有个feedback函数的调用，打开来看看

然后发现，哇，有load_flag欸



2. 再看看旁边一栏有好多的flag，分析发现除了flag3以外的flag都调用了load_flag，然后load_flag把参数nowflag变为了load_flag的返回值

   点开load_flag发现全变成了NULL（恼）



flag1：[NULL]

flag2：[NULL]

flag4：[NULL]

flag5：[NULL]

flag6：[NULL]

这样就还剩三个flag了

3. 来看看flag3吧

```
1 int __usercall check_flag3@<eax>(__int64 a1@<rbp>, __int64 a2@<rsi>)
2 {
3   int result; // eax@5
4   signed __int64 v3; // [sp-78h] [bp-78h]@1
5   signed __int64 v4; // [sp-70h] [bp-70h]@1
6   signed __int64 v5; // [sp-68h] [bp-68h]@1
7   signed int v6; // [sp-60h] [bp-60h]@1
8   signed __int16 v7; // [sp-5Ch] [bp-5Ch]@1
9   __int64 v8; // [sp-58h] [bp-58h]@1
10  signed int i; // [sp-Ch] [bp-Ch]@1
11  __int64 v10; // [sp-8h] [bp-8h]@1
12
13  __asm { rep nop edx }
14  v10 = a1;
15  sub_4011B0(4203850LL, a2);
16  sub_401230(4203879LL, &v8);
17  v3 = 8229867071280999782LL;
18  v4 = 4568429561244836439LL;
19  v5 = 7306288076125720167LL;
20  v6 = 1769959261;
21  v7 = 97;
22  for ( i = 0; i <= 28; ++i )
23    *((_BYTE *)&v10 + i - 112) ^= i;
24  if ( sub_401170(&v8, &v3, 29LL) )
25    result = sub_401190(4203410LL);
26  else
27    result = sub_401190(4203884LL);
28  return result;
29 }
```

emmmmm，貌似并不是很好看，我们换一个版本

```
{
  char s2[8]; // [rsp+0h] [rbp-70h] BYREF
  __int64 v2[2]; // [rsp+8h] [rbp-68h] BYREF
  int v3; // [rsp+18h] [rbp-58h]
  __int16 v4; // [rsp+1Ch] [rbp-54h]
  char s1[76]; // [rsp+20h] [rbp-50h] BYREF
  int i; // [rsp+6Ch] [rbp-4h]

  printf("[+] Please input the flag3: ");
  __isoc99_scanf("%64s", s1);
  *(_QWORD *)s2 = 0x72365C7F64636D66LL;
  qmemcpy(v2, "Wbd;{Rf?gN%|K&ee", sizeof(v2));
  v3 = 0x697F6F5D;
  v4 = 0x61;
  for ( i = 0; i <= 28; ++i )
    s2[i] ^= i;
  if ( !strncmp(s1, s2, 0x1DuLL) )
    return puts("[+] Correct flag3!");
  else
    return puts("[-] Wrong!");
}
```

这样就好看多了

发现貌似是异或题，使用这样的一个脚本来进行亦或

```
#include<stdio.h>
int main(){
    char s[30]=
{0x66,0x6D,0x63,0x64,0x7f,0x5c,0x36,0x72,'W','b','d',';','{','R','f','?','g','N'
,'%','|','K','&','e','e',0x5d,0x6f,0x7f,0x69,0x61};
    for(int i=0;i<=28;i++){
        printf("%c",s[i]^i);
    }
    return 0;
}
```

得到flag{Y0u_kn0w_h0w_7o_3srEver}

4. 这道题是唯一的pwn题
   攻击漏洞在哪呢?

```
pwndbg> vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /root/game
0x00401000         0x00402000         r-xp      /root/game
0x00402000         0x00403000         r--p      /root/game
0x00403000         0x00404000         r--p      /root/game
0x00404000         0x00405000         rw-p      /root/game
0x00007ffff7db7000 0x00007ffff7dba000 rw-p      mapped
0x00007ffff7dba000 0x00007ffff7de2000 r--p      /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7de2000 0x00007ffff7f3c000 r-xp      /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7f3c000 0x00007ffff7f92000 r--p      /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7f92000 0x00007ffff7f96000 r--p      /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7f96000 0x00007ffff7f98000 rw-p      /usr/lib/x86_64-linux-gnu/libc.so.6
0x00007ffff7f98000 0x00007ffff7fa5000 rw-p      mapped
0x00007ffff7fc0000 0x00007ffff7fc2000 rw-p      mapped
0x00007ffff7fc2000 0x00007ffff7fc6000 r--p      [vvar]
0x00007ffff7fc6000 0x00007ffff7fc8000 r-xp      [vdso]
0x00007ffff7fc8000 0x00007ffff7fc9000 r--p      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7fc9000 0x00007ffff7ff0000 r-xp      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ff0000 0x00007ffff7ffb000 r--p      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ffb000 0x00007ffff7ffd000 r--p      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffff7ffd000 0x00007ffff7fff000 rw-p      /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
0x00007ffffffde000 0x00007ffffffff000 rwxp      [stack]
```

   在feedback这个输入下是最好的攻击地址，给了0x100这么长，都够手写调用loadflag了
   先gdb看执行区域，可以执行//应该是
   再看没有system("bin/sh")看来ret2shellcode，那还是有难度
   查找rsp得到：0x7fffffffdd90
   现在攻击内容，地址都已经得到，可以开始调试//主要是害怕要栈对齐
   然后就不会了
   感谢队内只会 reverse 的pwn手的技术支持