Week3_Rev 公式Writeup

01 Waifu

古典纯粹的Win32逆向。WndProc追到关键函数sub_401000。
仔细看看就可以发现按下回车触发Check函数。
先判断byte_4043A6，0xA6-0x98==14，故flag长度为13。
主Check分为3个部分

Part1:

```
40  do
41  {
42    v4 = v3 & 0x1F;
43    if ( v3 & 0x1F )
44    {
45      if ( v4 == 2 && (input_s[v3] ^ 'i') != 35 )// [2]
46        v1 = 0;
47    }
48    else if ( (input_s[v3] ^ 'V') != 19 )       // [0]
49    {
50      v1 = 0;
51    }
52    if ( v4 == 4 )
53    {
54      if ( (input_s[v3] ^ 'd') != 0x53 )        // [4]
55        v1 = 0;
56    }
57    else if ( v4 == 6 && (input_s[v3] ^ 'a') != 55 )// [6]
58    {
59      v1 = 0;
60    }
61    if ( v4 == 8 && (input_s[v3] ^ 'r') != 28 ) // [8]
62      v1 = 0;
63    if ( v3 & 1 )
64    {
65      v5 = v0++ % 4;
66      v6 = 0;
67      if ( ((input_s[v3] + input_s[v3 + 1]) ^ 0x76) == dword_403260[v5] )// [others]
68        v6 = v1;
69      v1 = v6;
70    }
71    ++v3;
72  }
73  while ( v3 < 9 );
```

下标从0开始，至8结束：偶数与固定值异或；奇数与其前一位作和与固定值

异或。
能得到flag的前9位。

Part2:

```
74  v7 = input_s[11] * input_s[10] * input_s[9];
75  res1 = v7 * input_s[12];
76  if ( input_s[12] )
77    res2 = v7 / input_s[12];
78  else
79    res2 = 1;
80  if ( (res1 % 109 || res1 % 103)
81    && !(res2 % 41)
82    && input_s[9] < input_s[10]
83    && input_s[11] < input_s[9]
84    && input_s[9] < input_s[12] )
85  {
86    v1 = 0;
87  }
88  v10 = 0;
```

> - res1 = [9] * [10] * [11] * [12]
> - res2 = [9] * [10] * [11] * [12]

需要满足

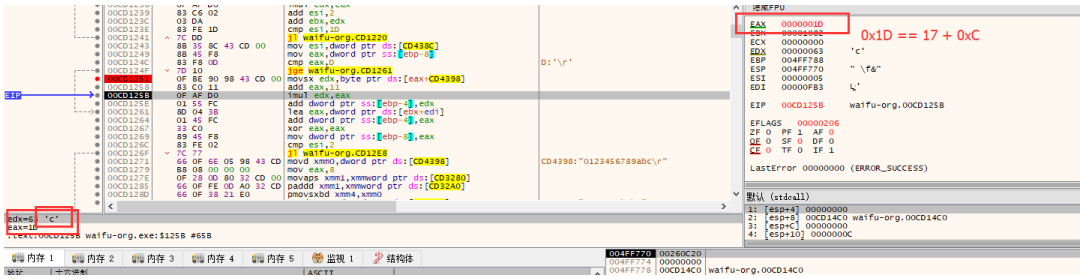
> - 1.res1 % 109 == 0 && res1 % 103 == 0
> - 2.res2 % 41 == 0
> - 3.[9] < [10] && [11] < [9] && [9] < [12]

Part3:

反编译后有点丑

```
106 do
107 {
108   v13 += v15 * byte_404387[v15];
109   v16 = (v15 + 1) * *(&dword_404388 + v15);
110   v15 += 2;
111   v14 += v16;
112 }
113 while ( v15 < 29 );
114 if ( v27 < 13 )
115   v28 += (v27 + 17) * input_s[v27];
116 res3 = v14 + v13 + v28;
117 v17 = 0;
118 if ( dword_40438C >= 2 )
119 {
120   v17 = 8;
121   v18 = _mm_add_epi32(
122           _mm_mullo_epi32(
123             _mm_cvtepi8_epi32(_mm_cvtsi32_si128(*&input_s[4])),
124             _mm_add_epi32(_mm_add_epi32(xmmword_403290, xmmword_403280), xmmword_4032A0)),
125           _mm_mullo_epi32(
126             _mm_cvtepi8_epi32(_mm_cvtsi32_si128(*input_s)),
127             _mm_add_epi32(xmmword_403280, xmmword_4032A0)));
128   v19 = _mm_add_epi32(v18, _mm_srli_si128(v18, 8));
129   v26 = _mm_cvtsi128_si32(_mm_add_epi32(v19, _mm_srli_si128(v19, 4)));
130 }
131 v20 = 0;
132 v21 = 0;
133 v22 = v17 + 16;
134 do
135 {
136   v20 += v22 * *(&dword_404388 + v22);
137   v23 = (v22 + 1) * *(&dword_404388 + v22 + 1);
138   v22 += 2;
139   v21 += v23;
140 }
141 while ( v22 < 28 );
142 v24 = v26;
143 if ( (v17 + 2 * ((11 - v17) >> 1) + 2) < 13 )
144   v24 = (v17 + 2 * ((11 - v17) >> 1) + 18) * input_s[2 * ((11 - v17) >> 1) + 2 + v17] + v26;
145 res4 = v21 + v20 + v24;
146 if ( res3 != 0x63A2 || res4 != 0x5F58 )
147   v1 = 0;
```

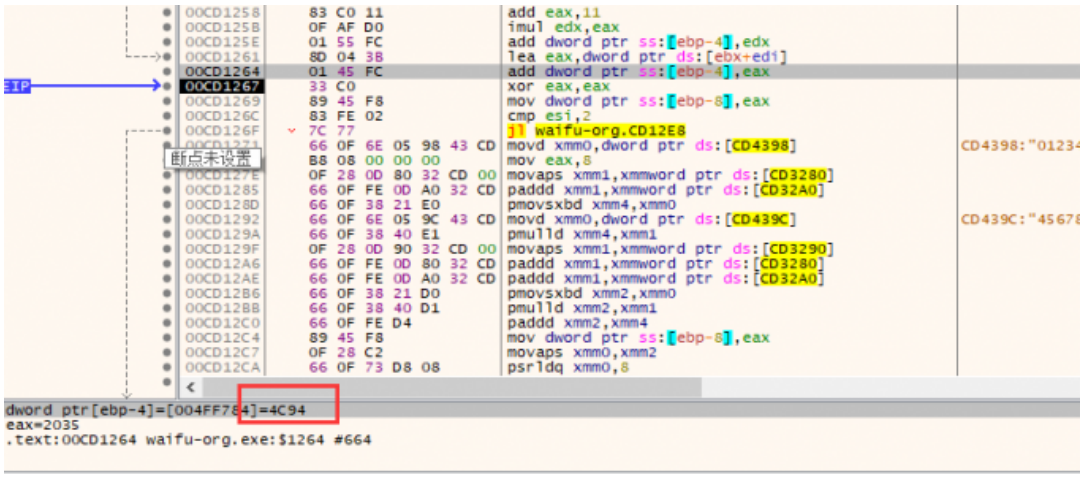
可以看出res3与res4的加密方式类似。以res3来举例:



以'0123456789abc'作为测试字符串
不难发现最后一个字符(下标0xC) 乘 (17 + 0xC)作为结果累加到某数值上
而0xC可从反编译代码中看出是个+1的递增量。
推断加密为

> - res3 += input[i] * (i + 17)

可以来验证一下




```
test = '0123456789abc'
res = 0
for i in range(13):
    res += (17+i) * ord(test[i])
print hex(res)
```

```
0x4c94
```

验证正确。res4同理，以+16来替代+17。
最后res3 res4与对应数值比较。

由此发现，一共13位的flag，前9位是可以一位位解密，后4位则需要爆破才可以得到。
dec脚本

```
dec = [0] * 13

# part1
```

```
xor_arr = [0xCE,0x11,0xC3,0x92]
for i in range(10):
    if i == 0:
        dec[i] = ord('V') ^ 0x13
    elif i == 2:
        dec[i] = ord('i') ^ 0x23
    elif i == 4:
        dec[i] = ord('d') ^ 0x53
    elif i == 6:
        dec[i] = ord('a') ^ 0x37
    elif i == 8:
        dec[i] = ord('r') ^ 0x1C
for i in range(4):
    dec[2*i+1] = (xor_arr[i] ^ 0x76) - dec[2*(i+1)]


# part2
res1 = 0
res2 = 0

for i in range(32, 127):
    for j in range(32, 127):
        for k in range(32, 127):
            for l in range(32, 127):
                res1 = i * j * k * l
                res2 = i * j * k / l
                if (i < j) and (k < i) and (i < l) and (res1 % 1
                    print i,j,k,l


# part3
try_arr =
[[41,103,36,109],[44,109,37,103],[49,109,34,103],[50,103,46,109]
[51,109,38,103],[52,109,38,103],[54,103,45,109],[57,109,34,103],
[60,103,34,109],[61,103,37,109],[61,109,47,103],[61,109,54,103],
[62,103,42,109],[62,103,49,109],[62,103,56,109],[62,109,35,103],
[62,109,45,103],[62,109,50,103],[62,109,55,103],[62,109,60,103],
[64,103,40,109],[65,109,62,103],[66,103,48,109],[67,103,57,109],
[68,103,67,109],[68,109,49,103],[68,109,53,103],[69,103,39,109],
[69,109,41,103],[70,109,62,103],[71,103,33,109],[71,103,44,109],
[71,103,66,109],[72,103,44,109],[73,109,43,103],[73,109,69,103],
[75,109,62,103],[76,109,52,103],[77,103,71,109],[78,109,76,103],
[79,103,39,109],[79,103,78,109],[79,109,77,103],[80,103,32,109],
[80,109,62,103],[81,103,45,109],[81,103,60,109],[81,103,75,109],
[84,103,78,109],[85,103,73,109],[85,109,62,103],[86,103,55,109],
[86,109,82,103],[87,109,53,103],[88,103,36,109],[88,103,71,109],
[89,103,59,109],[89,109,64,103],[90,103,54,109],[90,103,81,109],
[91,109,66,103],[92,103,50,109],[93,103,35,109],[93,103,42,109],
[93,109,40,103],[93,109,45,103],[93,109,50,103],[93,109,55,103],
```

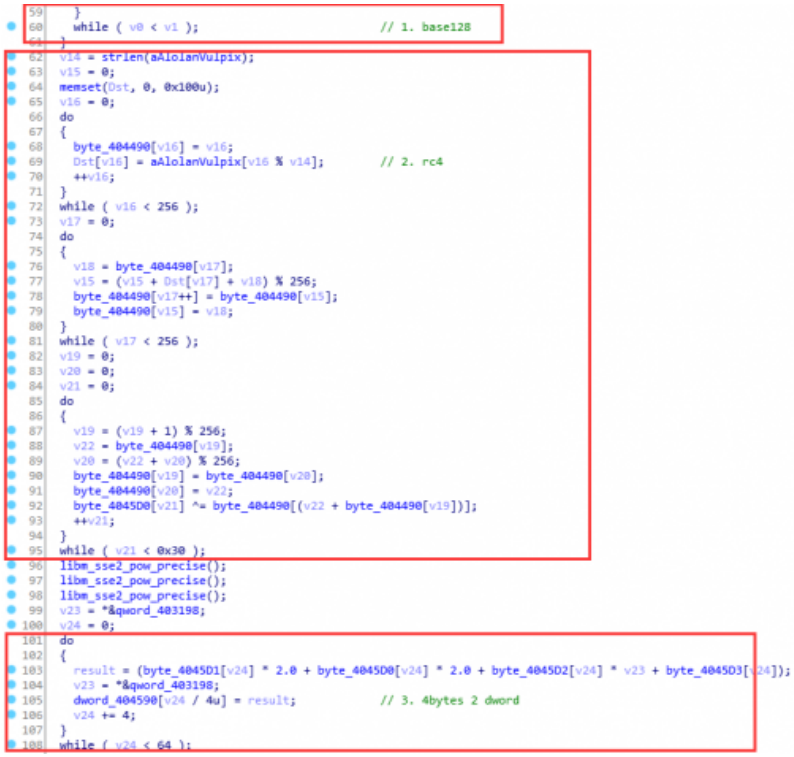
```
[94,109,61,103],[95,103,37,109],[96,103,33,109],[96,103,80,109],
[96,109,90,103],[97,103,34,109],[97,103,51,109],[97,103,68,109],
[98,103,89,109],[98,109,34,103],[98,109,51,103],[98,109,68,103],
[99,103,32,109],[99,103,71,109],[100,103,46,109],[100,109,74,103
[101,109,61,103],[101,109,89,103],[102,103,77,109],[102,109,49,1
[103,104,58,109],[103,104,63,109],[103,105,50,109],[103,105,81,1
[103,106,70,109],[103,106,79,109],[103,107,88,109],[103,108,45,1
[103,108,92,109],[103,109,32,120],[103,109,34,107],[103,109,36,1
[103,109,42,115],[103,109,42,125],[103,109,45,110],[103,109,45,1
[103,109,45,122],[103,109,49,104],[103,109,50,117],[103,109,51,1
[103,109,52,126],[103,109,60,111],[103,109,63,115],[103,109,63,1
[103,109,71,108],[103,109,71,120],[103,109,75,111],[103,109,78,1
[103,109,79,104],[103,109,79,105],[103,109,82,109],[103,109,84,1
[103,109,84,125],[103,109,86,116],[103,109,90,111],[103,109,90,1
[103,109,93,107],[103,109,93,119],[103,109,97,117],[103,109,99,1
[103,109,102,114],[103,110,43,109],[103,110,71,109],[103,111,43,
[103,112,74,109],[103,113,48,109],[103,113,96,109],[103,114,59,1
[103,116,49,109],[103,116,52,109],[103,116,98,109],[103,117,56,1
[103,118,82,109],[103,119,66,109],[103,119,101,109],[103,120,64,
[103,121,52,109],[103,121,71,109],[103,122,69,109],[103,122,85,1
[103,125,67,109],[103,125,84,109],[103,126,52,109],[104,109,103,
[104,126,103,109],[105,109,103,115],[105,109,103,125],[107,109,1
[109,113,103,119],[109,115,103,123],[109,126,103,115],[109,126,1
res3 = 0
res4 = 0
explo_res = [0] * 4
for i in range(len(try_arr)):
    for j in range(9):
        res3 += (j+17) * dec[j]
        res4 += (j+16) * dec[j]
    for k in range(4):
        res3 += (k+9+17) * try_arr[i][k]
        res4 += (k+9+16) * try_arr[i][k]
    if res3 == 0x63A2 and res4 == 0x5F58:
        explo_res = try_arr[i]
    else:
        res3 = 0
        res4 = 0
#print explo_res # [68, 109, 53, 103]

for i in range(13):
    if(i<9):
        dec[i] = chr(dec[i])
    else:
        dec[i] = chr(explo_res[i-9])
print ''.join(dec)
# flag: EnJ07_VvnDm5g
```

02 Another Waifu

先upx脱壳。手脱工具脱随意。发现对话框过程函数，可知按下控件[OK]时会触发Check函数。

> - 第1层: Base128
> - 第2层: 正规Rc4
> - 第3层: 每4个hexByte组成dword



（base128没截全

dec脚本:

```
def dec_base128(dst, src):
    i=0
    j=0
    k=0
    while(j<48):
        dst[i + 0] = 0xff & (src[j + 0] << 1 | (src[j + 1] >> 6
        dst[i + 1] = 0xff & (src[j + 1] << 2 | (src[j + 2] >> 5
        dst[i + 2] = 0xff & (src[j + 2] << 3 | (src[j + 3] >> 4
        dst[i + 3] = 0xff & (src[j + 3] << 4 | (src[j + 4] >> 3
        dst[i + 4] = 0xff & (src[j + 4] << 5 | (src[j + 5] >> 2
        dst[i + 5] = 0xff & (src[j + 5] << 6 | (src[j + 6] >> 1
        dst[i + 6] = 0xff & (src[j + 6] << 7 | (src[j + 7] >> 0
        i+=7
```

```
        j+=8
def rc4(data, key):
    S, j, out = list(range(256)), 0, []

    for i in range(256):
        j = (j + S[i] + ord(key[i % len(key)])) % 256
        S[i], S[j] = S[j], S[i]

    i = j = 0
    for ch in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        out.append(ch ^ S[(S[i] + S[j]) % 256])

    return out

dec_flag = [0] * 48
enc_d = [0x84, 0xd1, 0x3c, 0x7d, 0x49, 0xeb, 0x7b, 0xbe, 0x85, 0
rc4_key = "Alolan_Vulpix"
dec_rc4 = rc4(enc_d, rc4_key)
dec_base128(dec_flag, dec_rc4)

for i in range(len(dec_flag)):
    dec_flag[i] = chr(dec_flag[i])
print ''.join(dec_flag)

# hgame{Ez_Encr7pt_c4n_n0t_sT0p_Ur_pr0gre5s}
```

03 Darling Waifu

反调试- Debug Blocker
Debug-Blocker反调试技术的原理是创建两个进程，父进程作为Debugger，子进程作为Debuggee，而一个进程在同一时间只允许被一个Debugger调试。
其中子进程中有被加密的代码，也就是被保护的代码；父进程中有个调试循环，等待着处理调试信息。
子进程代码被解密依靠的是父进程对它的操作，这需要一个信号来通知父进程是时候解密子进程了。这个信号就是异常。
简而言之：子进程遇到异常，抛出给Debugger(父进程)，父进程借此对子进程的代码解密。

程序start后根据互斥量判断父子进程。

父进程:

```
while ( 1 )
{
  memset(&DebugEvent, 0, 0x60u);
  result = WaitForDebugEvent(&DebugEvent, 0xFFFFFFFF);
  if ( !result )
    break;
  if ( DebugEvent.dwDebugEventCode == 1 )   // 1. 判断调试事件是否为异常
  {
    v2 = DebugEvent.u.Exception.ExceptionRecord.ExceptionAddress;
    if ( DebugEvent.u.Exception.ExceptionRecord.ExceptionCode == 0xC000001D// 2. 判断异常类型是否为非法指令
      && DebugEvent.u.Exception.ExceptionRecord.ExceptionAddress == &word_401366 )// 3. 判断异常地址是否为401366
    {
      ReadProcessMemory(ProcessInformation.hProcess, (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);
    }
  }
  else if ( DebugEvent.dwDebugEventCode == 5 )
  {
    return result;
  }
  ContinueDebugEvent(DebugEvent.dwProcessId, DebugEvent.dwThreadId, 0x10002u);
}
```

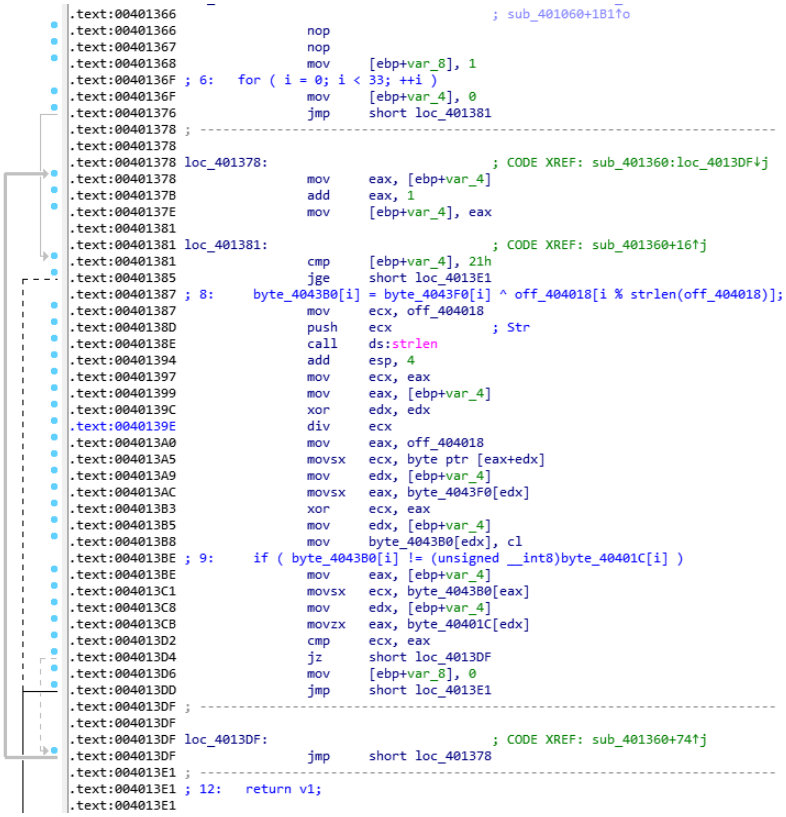
大致就可以推断出401366即为被加密的地址。Read这个地址的数据，做一个简单的异或，再Write回去。
最后Eip+2跳过异常。

子进程:

```
    if ( a3 == 1 )
    {
      if ( (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;
}
```

可以看到loc_401360本应该是个Check函数，但被加密无法识别。

所以静态方法可以手动解码然后patch回去：
将每个字节异或0x76后，可以成功反汇编了。



被加密的Check函数就很简单了，只有一层异或。
dec脚本:

```
key = "Strelitzia"
enc_flag =
[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
dec_flag = [0] * len(enc_flag)

for i in range(len(enc_flag)):
    dec_flag[i] = chr(enc_flag[i] ^ ord(key[i%len(key)]))

print ''.join(dec_flag)
# hgame{Anti_4n5i_D5bu77ing_u_D0N5}
```