171106 逆向-SWPU（Re300-初窥）

原创

奈沙夜影 于 2017-11-07 11:38:57 发布 277 收藏

分类专栏： CTF

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/whklhhhh/article/details/78466323

版权

CTF 专栏收录该内容

163 篇文章 4 订阅

订阅专栏

1625-5 王子昂 总结《2017年11月6日》【连续第402天总结】

A. 08067CTF-re300

B.

首先查壳，发现似乎有点段的问题，但是没识别到壳

拖入OD和IDA尝试，都能正常反编译

于是运行查看可能的突破点：

这个确认按钮会不停的动，点不到

文本框输入无反应

在OD中对GetWindowTextA下断，尝试输入发现无反应，说明程序不是随时都在接收输入的，必须要点到这个确定按钮

IDA查找GetWindowTextA的交叉引用，发现有数十个，无从下手

于是只好先搞定确定按钮乱飞的问题啦：

百度一下查找改变控件的实现，发现是通过SetWindowPos来做的，IDA中的交叉引用仍然有数十个，没法定位准确的调用位置

于是在OD中对该API下断，鼠标放到按钮上果然断到，查看调用堆栈后锁定sub_41451E
再向上交叉引用找到改变位置的地方：

```
int __thiscall sub_403730(int this, int a2, int a3, int a4)
{
  int v4; // edi@1
  HWND v5; // ST10_4@1
  HWND v6; // eax@1
  int v7; // eax@1
  int v8; // eax@1
  int v9; // esi@1
  int v10; // edx@3
  struct tagRECT v12; // [sp+Ch] [bp-24h]@1
  struct tagRECT Rect; // [sp+1Ch] [bp-14h]@1

  v4 = this;
  _mm_storeu_si128((__m128i *)&Rect, 0i64);
  v5 = *(HWND *)(this + 32);
  _mm_storeu_si128((__m128i *)&v12, 0i64);
  GetWindowRect(v5, &Rect);
  v6 = GetParent(*(HWND *)(v4 + 32));
  v7 = sub_40F4CE(v6);
  GetWindowRect(*(HWND *)(v7 + 32), &v12);
  sub_409D75(v4, (LPPOINT)&Rect);
  sub_409D75(v4, (LPPOINT)&v12);
  v8 = sub_528DFF();
  v9 = v8 % v12.right;
  if ( v8 % v12.right > v12.right - Rect.right )
    v9 = v8 % v12.right - Rect.right;
  v10 = sub_528DFF() % v12.bottom;
  if ( v10 > v12.right - Rect.bottom )
    v10 -= Rect.bottom;
  sub_41451E(v4, 0, v9, v10, 0, 0, 5u);         // SetWindowPos
  return sub_40F284((void *)v4);
}
```

http://blog.csdn.net/whk1hhhh

Rect结构体就是MFC中描述矩形规格的，OD中把这个函数NOP掉
（直接搞会报错，猜测是修改不完全造成堆栈不平衡，__security_check_cookie这个函数检查时抛出异常导致终止，因此还要在上级函数将它NOP掉）

搞定以后就能断到GetWindowTexaA了：

```
int __thiscall sub_40F99B(int this, int a2)
{
  int v2; // esi@1
  int v3; // ecx@1
  int v4; // eax@2
  int v5; // ST08_4@2
  CHAR *v6; // eax@2
  int result; // eax@2

  v2 = this;
  v3 = *(_DWORD *)(this + 108);
  if ( v3 )
  {
    result = (*(int (__stdcall **)(int))(*(_DWORD *)v3 + 140))(a2);
  }
  else
  {
    v4 = GetWindowTextLengthA(*(HWND *)(v2 + 32));
    v5 = v4 + 1;
    v6 = (CHAR *)sub_40BF84(v4);
    GetWindowTextA(*(HWND *)(v2 + 32), v6, v5);
    result = sub_40681E(-1);
  }
  return result;
}
```

http://blog.csdn.net/whk1hhhh

向上溯源找到核心算法：

```
21    sub_4029F0(0x80004005);
22  input = (*(int (__thiscall **)(int))(*(_DWORD *)v0 + 12))(v0) + 16;
23  v14 = 0;
24  v1 = sub_413CC7(1002);
25  sub_40F99B(v1, (int)&input);                  // GetText
26  v2 = strlen((const char *)input);
27  if ( v2 <= 8 )
28  {
29    memset(&input + v2, 0, 8 - v2);             // 用0补齐8个字节
30    i58 = *(_DWORD *)(input + 4);               // 后4个字符
31    i04 = *(_DWORD *)input;                     // 前4个字符
32    i58_ = i58;
33    v10 = BYTE3(i04) | ((BYTE2(i04) | ((BYTE1(i04) | ((unsigned __int8)i04 << 8)) << 8)) << 8);
34    v11 = BYTE3(i58) | ((BYTE2(i58) | ((BYTE1(i58_) | ((unsigned __int8)i58 << 8)) << 8)) << 8);// 逆序
35    qmemcpy(&v9, &unk_5A1308, 0x80u);           // 密钥复制
36    sub_401B20((int)&v10, (int)&v9);            // 加密变换
37    LOBYTE(i04) = BYTE3(v10);
38    BYTE1(i04) = BYTE2(v10);
39    BYTE2(i04) = BYTE1(v10);
40    BYTE3(i04) = v10;
41    LOBYTE(i58_) = BYTE3(v11);
42    BYTE1(i58_) = BYTE2(v11);
43    BYTE2(i58_) = BYTE1(v11);
44    BYTE3(i58_) = v11;
45    v10 = i04;
46    v11 = i58_;
47    v4 = 0;
48    while ( *((_BYTE *)&v10 + v4) == byte_581500[v4] )// 校验结果
49    {
50      if ( ++v4 >= 8 )
51      {
52        v7 = (const CHAR *)dword_5ABA1C;
53        goto LABEL_9;
54      }
55    }
56    v7 = dword_5ABA20;
```

http://blog.csdn.net/whklhhhh

于是变换的算法就在sub_401b20里了，进去看看：

```
98  int result; // eax@1
99
100 v2 = a1;
101 v3 = *(_DWORD *)(a1 + 4);
102 v4 = (v3 ^ (*(_DWORD *)a1 >> 4)) & 0xF0F0F0F;
103 v5 = v4 ^ v3;
104 v6 = 16 * v4 ^ *(_DWORD *)a1;
105 v7 = (unsigned __int16)(v5 ^ HIWORD(v6));
106 v8 = v7 ^ v5;
107 v9 = (v7 << 16) ^ v6;
108 v10 = (v9 ^ (v8 >> 2)) & 0x33333333;
109 v11 = v10 ^ v9;
110 v12 = 4 * v10 ^ v8;
111 v13 = (v11 ^ (v12 >> 8)) & 0xFF00FF;
112 v14 = v13 ^ v11;
113 v15 = __ROL4__(v12 ^ (v13 << 8), 1);
114 v16 = (v14 ^ v15) & 0xAAAAAAAA;
115 v17 = v16 ^ v15;
116 v18 = v16 ^ v14;
117 v19 = a2;
118 v20 = __ROL4__(v18, 1);
119 v21 = dword_580D00[(v17 ^ *(_DWORD *)a2) & 0x3F] ^ dword_580F00[(((unsigned int)v17 ^ *(_DWORD *)a2) >> 8) & 0x3F] ^ dword_581100[(((unsigned int)v17 ^ *(_DWOI
120 v22 = __ROR4__(v17, 4);
121 v23 = v22 ^ *(_DWORD *)(a2 + 4);
122 v24 = dword_580E00[v23 & 0x3F] ^ dword_581000[(v23 >> 8) & 0x3F] ^ dword_581200[(v23 >> 16) & 0x3F] ^ dword_581400[(v23 >> 24) & 0x3F] ^ v21;
123 v25 = v24 ^ *(_DWORD *)(v19 + 8);
124 v26 = __ROR4__(v24, 4);
125 v27 = dword_580E00[(v26 ^ *(_DWORD *)(v19 + 12)) & 0x3F] ^ dword_581000[((v26 ^ *(_DWORD *)(v19 + 12)) >> 8) & 0x3F] ^ dword_581200[((v26 ^ *(_DWORD *)(v19 +
126 v28 = v27 ^ *(_DWORD *)(v19 + 16);
127 v29 = dword_580D00[v28 & 0x3F] ^ dword_580F00[(v28 >> 8) & 0x3F] ^ dword_581100[(v28 >> 16) & 0x3F] ^ dword_581300[(v28 >> 24) & 0x3F] ^ v24;
128 v30 = __ROR4__(v27, 4);
129 v31 = dword_580E00[(v30 ^ *(_DWORD *)(v19 + 20)) & 0x3F] ^ dword_581000[((v30 ^ *(_DWORD *)(v19 + 20)) >> 8) & 0x3F] ^ dword_581200[((v30 ^ *(_DWORD *)(v19 + 2
130 v32 = v31 ^ *(_DWORD *)(v19 + 24);
131 v33 = dword_580E00[(v30 ^ *(_DWORD *)(v19 + 20)) & 0x3F] ^ dword_581000[((v30 ^ *(_DWORD *)(v19 + 20)) >> 8) & 0x3F] ^ dword_581200[((v30 ^ *(_DWORD *)(v19 + 2
132 v34 = __ROR4__(v31, 4);
133 v35 = dword_580E00[(v34 ^ *(_DWORD *)(v19 + 28)) & 0x3F] ^ dword_581000[((v34 ^ *(_DWORD *)(v19 + 28)) >> 8) & 0x3F] ^ dword_581200[((v34 ^ *(_DWORD *)(v19 + 2
```

嗯.....

无能为了了，这个复杂的算法凭自己是看不出来干啥的，逐步还原的话似乎可能性也不高*(:з」∠)*

只好等WriteUp了

C. 明日计划

参照WriteUp解决re300