CTF小白学习笔记(Reverse)-i春秋 CrackMe-1

原创

I still … 于 2020-12-02 10:33:43 发布 404 收藏

分类专栏： CTF的writeup

本文链接：https://blog.csdn.net/qq_44370676/article/details/110470001

CTF的writeup 专栏收录该内容

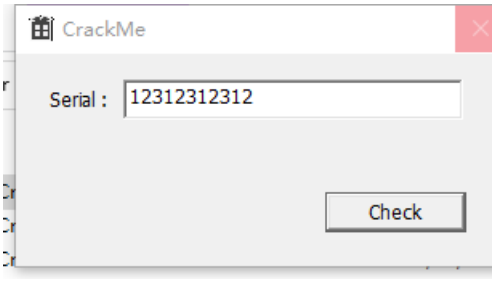
8 篇文章 0 订阅

订阅专栏

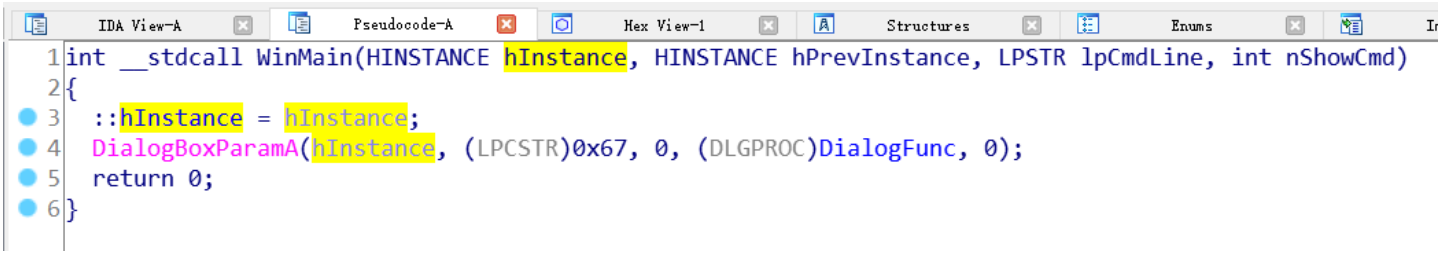
主要考察动态调试，不过动态调试的过程就略去了

解题过程

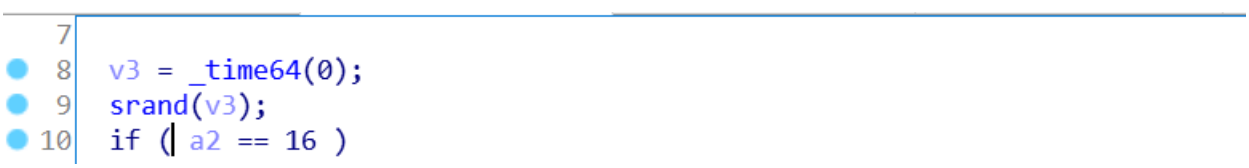
运行一下,大概就是如果输入的是flag就成功：



查壳发现无壳，直接上IDA：



点进DialogFunc



```
11  {
12    EndDialog(hWnd, 0);
13    return 0;
14  }
15  if ( a2 == 272 )
16  {
17    v7 = LoadIconA(hInstance, (LPCSTR)0x6B);
18    SendMessageA(hWnd, 0x80u, 1u, (LPARAM)v7);
19    InitCommonControls();
20    return 0;
21  }
22  if ( a2 != 273 || (_WORD)a3 != 1000 )
23    return 0;
24  v4 = sub_401210(hWnd);
25  if ( v4 <= 2345 )
26  {
27    switch ( v4 )
28    {
29      case 2345:
30        MessageBoxA(0, "Serial Check Error!", "CrackMe", 0);
31        return 0;
32      case 360:
33        return rand();
34      case 751:
35        MessageBoxA(0, "This is Flag!", "CrackMe", 0);
36        return 0;
37    }
38    return 0;
39  }
40  if ( v4 != 5173 )
```


这里关键步骤应

该在sub_401210里，如果返回值是751，那就成功了，点进去看看

```
0040
71  CHAR String[2]; // [esp+78h] [ebp-7Ch]
72  int Dst[10]; // [esp+A0h] [ebp-54h]
73  int v72[10]; // [esp+C8h] [ebp-2Ch]
74
75  _mm_storeu_si128((__m128i *)&v68, _mm_load_si128((const __m128i *)&xmmword_40FE60));
76  v1 = hDlg;
77  String[0] = 0;
78  _mm_storeu_si128((__m128i *)&v69, _mm_load_si128((const __m128i *)&xmmword_40FE50));
79  memset(&String[1], 0, 0x27u);
80  GetStartupInfoA(&StartupInfo);
81  if ( StartupInfo.dwX
82    || StartupInfo.dwY
83    || StartupInfo.dwXCountChars
84    || StartupInfo.dwYCountChars
85    || StartupInfo.dwFillAttribute
86    || StartupInfo.dwXSize
87    || StartupInfo.dwYSize
88    || SLOBYTE(StartupInfo.dwFlags) < 0 )
89  {
90    return 360;
91  }
92  if ( !GetDlgItemTextA(v1, 1001, String, 201) )
93    return 5173;
94  v3 = strlen((const char *)&v68);
95  v4 = strlen(String);
96  if ( v3 > 20 || 2 * v3 != v4 )
97    return 5173;
98  v5 = 0;
99  if ( v3 > 0 )
100 {
```

```
101   do
102   {
```

这里有几个可疑变量，v68, String。经过调试发现v68是个恒定的字符串，长度18，值为:

```
BinGzLFormiChunQiu
```

String就是输入字符串。这里可疑看出输入时36个字符

```
108   }
109   i = 0;
110   if ( v4 > 0 )
111   {
112     do
113     {
114       chr = String[i];
115       if ( (chr < 65 || chr > 70) && (chr < 48 || chr > 57) )
116         return 5173;
117     }
118     while ( ++i < v4 );
119   }
120   LOBYTE(Dst[0]) = 0;
```

这个判断就是确保输入的字符都是0-9 A-F, 也就是输入是16进制字符串

```
150       do
151       {
152         v18 = *(v15 - 1);
153         v19 = *v15;
154         v21 = (unsigned __int8)(2
155                                 * ((v19 ^ ((v18 ^ ((v16 ^ ((v57 ^ ((v47 ^ v48 ^ ((v49 ^ v56 ^ ((unsigned __int8)(2 * ((v19 ^ ((v18 ^ ((v16 ^ ((
156                                                                                                                                 % v13
157                                                                                                                                 + 1))
158                                                                                                                       + 2))
159                                                                                                             + 1))
160                                                                                                   + 1))
161                                                                                         + 1))
162                                                                               + 1))
163                                                     + 1))
164             % v13;
165         v17 = *v15;
166         v14 = (v56 ^ v57 ^ ((((v47 ^ v48 ^ ((v49 ^ v56 ^ (v21 + 1)) + 2)) + 1) ^ v16) + 2)) + 1;
167         --v20;
168       }
169       while ( v20 );
170       v13 += v65;
171       v15 = v53 + 16;
172       v53 += 16;
173       v22 = (*v50 ^ v50[1] ^ ((v14 ^ v50[2]) + 2)) + 1;
174       v50 += 16;
175       v14 = v22;
176     }
177     while ( (signed int)v50 < (signed int)&unk_40FDDC );
178     *(_WORD *)&Src[1] = 0;
179     *(_WORD *)Src = *(_WORD *)&String[2 * v64];
180     v23 = sub_401000(Src);
```

这中间是个漫长的加密过程，具体过程就不慢慢研究了，179行的赋值操作值得注意。经过动态调试发现,src每次取出36个字符中的2个字符，然后v23就是把这两个字符转化成byte(比如src = ‘EF’, v23就是0xEF)

```
266         v23 = v37 ^ (v45 + 2);
267         v55 = v36;
268       }
269       while ( (signed int)v36 < (signed int)&unk_40FD5B );
270       v10 = &byte_40FD40[-2];
271       v12 = (char *)&v68 + v3;
272       *((_BYTE *)v72 + v64) = v23;
273       v9 = v64 + 1;
274       v11 = v65 - 1;
275       v64 = v9;
276       --v65;
277     }
278     while ( v9 < v3 );
279   }
280   v46 = 0;
281   while ( Dst[v46] == v72[v46] )
282   {
283     ++v46;
284     if ( v46 >= 10 )
285       return 751;
286   }
```


这里就很关键了，v23各种加密后赋值到v72字符串中，最后v72字符串和Dst字符串比对，Dst字符串是写死的，值为

```
3C 81 64 30 E8 EE 0A 90 20 1B 46 52 C8 20 FE D4 8C FE
```

所以说这个程序执行了18次加密过程，每次都是上下文无关的，加密过程很复杂，不好逆向。那就只能hash了(算出从0x00-0xFF加密过后的字符)

每次输入18个，搞了好多次,dump出一个hash表

```
   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0 2F 86 FD BC 5B 92 69 08 07 3E 35 14 53 4A 01 60
1 9F 36 6D EC 0B 82 19 F8 B7 2E 25 84 43 BA B1 D0
2 CF A6 1D DC 7B B2 89 28 A7 DE 55 34 F3 6A A1 00
3 3F D6 0D 0C AB 22 B9 18 57 4E 45 A4 63 DA 51 70
4 6F C6 3D FC 9B D2 A9 48 47 7E 75 54 93 8A 41 A0
5 DF 76 AD 2C 4B C2 59 38 F7 6E 65 C4 83 FA F1 10
6 0F E6 5D 1C BB F2 C9 68 E7 1E 95 74 33 AA E1 40
7 7F 16 4D 4C EB 62 F9 58 97 8E 85 E4 A3 1A 91 B0
8 AF 06 7D 3C DB 12 E9 88 87 BE B5 94 D3 CA 81 E0
9 1F B6 ED 6C 8B 02 99 78 37 AE A5 04 C3 3A 31 50
A  4F 26 9D 5C FB 32 09 A8 27 5E D5 B4 73 EA 21 80
B BF 56 8D 8C 2B A2 39 98 D7 CE C5 24 E3 5A D1 F0
C EF 46 BD 7C 1B 52 29 C8 C7 FE F5 D4 13 0A C1 20
D 5F F6 2D AC CB 42 D9 B8 77 EE E5 44 03 7A 71 90
E 8F 66 DD 9C 3B 72 49 E8 67 9E 15 F4 B3 2A 61 C0
F FF 96 CD CC 6B E2 79 D8 17 0E 05 64 23 9A 11 30
```

本来还想写脚本的不过这个表本身就有规律，而且也就18个数字，直接手动算flag得了，最终flag为：

```
flag{838EFBFFE7D9CDDFCFC4C1C5C7CFC9CBB3C9}
```