看雪CTF2017第二题lelfeiCM的writeup

原创

anhkgg 于 2017-06-14 09:09:56 发布 1105 收藏

分类专栏： 原创 文章标签： ctf 逆向 看雪 crackme

本文链接：https://blog.csdn.net/angelxf/article/details/73198188

题目入口：http://ctf.pediy.com/game-fight-32.htm，可下载相关文件

0. 定位算法位置

由于是console程序，并且没有隐藏字符串，通过OD/IDA找到关键字符串，所在函数就是关键算法函数：

```
.data:00409058 aWellDone       db 'WELL DONE!',0Ah,0   ; DATA XREF: _main:loc_401257o
.data:00409064 aWrongKey___    db 'WRONG KEY...',0Ah,0 ; DATA XREF: _main+231o
.data:00409072                 align 4
.data:00409074 aKeyFormatError db 'key format error...',0Ah,0 ; DATA XREF: _main+9Ao
```

其实就在main函数中，然后看获取输入之后干了什么。
首先检查输入长度是不是在8到20之间，不是提示key len error

```
.text:00401066                 cmp     ecx, 8
.text:00401069                 jl      loc_40127A
.text:0040106F                 cmp     ecx, 14h
.text:00401072                 jg      loc_40127A
.text:00401078                 xor     esi, esi
.text:0040107A                 xor     edx, edx
.text:0040107C                 test    ecx, ecx
```

是不是都是数值，不是就提示key format error…

```
.text:00401082                 jle     short loc_4010AC
.text:00401084
.text:00401084 loc_401084:                             ; CODE XREF: _main+94j
.text:00401084                 mov     al, [esp+edx+4138h+key]
.text:00401088                 cmp     al, 30h
.text:0040108A                 jle     short loc_401090
.text:0040108C                 cmp     al, 39h
.text:0040108E                 jle     short loc_401091
.text:00401090
.text:00401090 loc_401090:                             ; CODE XREF: _main+8Aj
.text:00401090                 inc     esi
.text:00401091
.text:00401091 loc_401091:                             ; CODE XREF: _main+8Ej
.text:00401091                 inc     edx
.text:00401092                 cmp     edx, ecx
.text:00401094                 jl      short loc_401084
.text:00401096                 test    esi, esi
.text:00401098                 jz      short loc_4010AC
.text:0040109A                 push    offset aKeyFormatError ; "key format error...\n"
.text:0040109F                 call    f_printf_401BE0
```

下面接着就是算法的重要部分了，一看到下面的函数，就知道有点小类结构了

```
.text:004012C0 ; KEY_OBJ1 *__thiscall f_keyobj_init_4012C0(KEY_OBJ1 *this)
.text:004012C0 f_keyobj_init_4012C0 proc near          ; CODE XREF: _main+B3 p
.text:004012C0                                         ; f_keyobj_calc_mul_401730+29p ...
.text:004012C0                 push    esi
.text:004012C1                 mov     esi, ecx
.text:004012C3                 mov     dword ptr [esi], offset off_4080C8
.text:004012C9                 call    ds:GetTickCount
.text:004012CF                 mov     ecx, esi
.text:004012D1                 mov     [esi+200Ch], eax
.text:004012D7                 mov     [esi+2008h], eax
.text:004012DD                 call    f_keyobj_init_seed1_401A60
.text:004012E2                 mov     eax, esi
.text:004012E4                 pop     esi
.text:004012E5                 retn
.text:004012E5 f_keyobj_init_4012C0 endp
```

1. 算法类结构分析，各类函数的功能分析

先把类结构大致整理出来，方便后续分析

```
00000000 KEY_OBJ1        struc ; (sizeof=0x2010) ; XREF: _mainr
00000000                                         ; f_keyobj_calc_mul_401730r
00000000 vtable_4080C8   dd ?
00000004 cur_calc_pos    dd ? //结果长度
00000008 seed_array_1024_1 dd 1024 dup(?) //保存key的值
00001008 seed_array_1024 dd 1024 dup(?) //保存序号
00002008 TickCnt_key_seed dd ?
0000200C TickCnt1        dd ?
00002010 KEY_OBJ1        ends
```

然后就是几个关键函数：

1.1 初始化数据

```
.text:00401A60 ; char *__thiscall f_keyobj_init_seed1_401A60(KEY_OBJ1 *this)
...
.text:00401A8F                 call    f_kyeobj_getindex_4019E0 //更加GetTickCount获取随机index，用于打乱
...
.text:00401ABA                 mov     esi, [ecx]
.text:00401ABC                 sub     ecx, 4
.text:00401ABF                 mov     [eax], esi
.text:00401AC1                 add     eax, 4
.text:00401AC4                 dec     edx
```

这个地方首先就想到了每次GetTickCount不一样，那么算法怎么保证结果相同呢，便想到肯定跟index顺序无关，后面验证果然是，我就把401A60给patch了一下，然初始化的序号结构没有打乱顺序，保持0-0x3ff，如下

```
//nop了00401A8F调用的循环部分
.text:00401A8F                 call    f_kyeobj_getindex_4019E0
//这里其实就是seed_array_1024[1023]，不让它倒过来赋值，修改为lea     ecx, [esi+1008h]
.text:00401AAE                 lea     ecx, [esi+2004h]
```

这样之后，就可以很方便查看数据变换，观察这两个字段即可

```
00000004 cur_calc_pos    dd ? //结果长度
00000008 seed_array_1024_1 dd 1024 dup(?) //保存key的值
```

后面所有相关函数中有关index转换的也不用关注，因为他变来变去都是0-0x3ff，就只需要关注具体数据操作了。
然后其他函数功能分析也就简单了。
下面简单列一下，不做详细说明了（很简单，就是数组操作过来过去的）

```
.text:004014E0 ; int __thiscall f_keyojb_key1_4014E0(void *this, const char *key) //将输入的key保存到seed
.text:00401970 ; void __thiscall f_keyobj_key1_s2_401970(KEY_OBJ1 *this) //数值大于10，取余存当前index位置
.text:00401730 ; signed int __userpurge f_keyobj_calc_mul_401730@<eax>(int a1@<eax>, int keyobj0@<ecx>,
text:00401840 ; signed int __userpurge f_keyobj_mul2_401840@<eax>(int a1@<eax>, int a2@<ecx>, KEY_OBJ1
```

2. 醒悟算法究竟是个什么玩意

输入的key关键处理部分

```
.text:004010E0                 push    9
.text:004010E2                 lea     ecx, [esp+413Ch+keyobj]
.text:004010E9                 call    f_keyobj_calc_mul_401730 ;
...
.text:0040110B                 lea     eax, [esp+4138h+keyobj1]
.text:00401112                 lea     ecx, [esp+4138h+keyobj]
.text:00401119                 push    eax
.text:0040111A                 mov     byte ptr [esp+413Ch+var_4], 1
.text:00401122                 call    f_keyobj_mul2_401840
...
.text:00401127                 push    9
.text:00401129                 lea     ecx, [esp+413Ch+keyobj]
.text:00401130                 mov     esi, eax
.text:00401132                 call    f_keyobj_calc_mul_401730 ;
```

先前想着输入的key用9做位移，做加法，干么呢…一直绕不清，后来重新看f_keyobj_key1_s2_401970，觉得是进位处理，一下子就灵光了，这是实现乘法运算（1024位的乘法，真实折腾，nb）。
这样算法也基本清楚了。
key*9*key*9*(…) => result

怎么校验的呢？

1. 计算结果长度必须是奇数

```
.text:00401154                call    f_keyobj_curpos_4013A0
.text:00401159                and     eax, 80000001h
.text:0040115E                jns     short loc_401165
.text:00401160                dec     eax
.text:00401161                or      eax, 0FFFFFFFEh
.text:00401164                inc     eax
.text:00401165
.text:00401165 loc_401165:                            ; CODE XREF: _main+15Ej
.text:00401165                cmp     eax, 1
```

1. result[len/2] == key[0]

```
.text:00401175                call    f_keyobj_curpos_4013A0
.text:0040117A                sar     eax, 1
.text:0040117C                push    eax
.text:0040117D                lea     ecx, [esp+413Ch+keyobj]
.text:00401184                call    f_keyobj_check1_4013B0
.text:00401189                push    0
.text:0040118B                lea     ecx, [esp+413Ch+keyobj1]
.text:00401192                mov     edi, eax
.text:00401194                call    f_keyobj_check1_4013B0
.text:00401199                cmp     edi, eax
.text:0040119B                lea     ecx, [esp+4138h+keyobj1]
.text:004011A2                jnz     short loc_40121C
```

1. 高位部分和key相同（跳过比较那个字节）

```
.text:004011D0                lea     ecx, [esp+4144h+keyobj1]
.text:004011D7                push    esi
.text:004011D8                push    ecx
.text:004011D9                lea     ecx, [esp+414Ch+keyobj]
.text:004011E0                call    f_keyobj_check2_4013E0
```

1. 低位部分和key逆序（跳过比较那个字节）

```
text:004011F6                lea     edx, [esp+413Ch+keyobj1]
.text:004011FD                push    eax
.text:004011FE                push    1
.text:00401200                push    0
.text:00401202                push    edx
.text:00401203                lea     ecx, [esp+414Ch+keyobj]
.text:0040120A                call    f_keyobj_check2_4013E0
```

感觉结果应该是这一个样子的：

```
1234567->1234567654321 //中间因为长度折腾了好久，后面查了才知道这是回文数，翻半天没有什么算法，脚本已经跑起来了
```

怎么求逆呢？算法不好，那就脚本跑吧！

3. 脚本跑

```
i = 11111111#
while True:
    break
    is1 = str(i)
    is2_len = len(is1)

    is1 = is1[:is2_len-1]
    is2 = is1[::-1]

    k = i*9*9*i

    ks1 = ''
    ks = ''
    while True:

        #print i, k
        #break
        ks1 = str(k)

        lll = len(ks1)/2

        if len(ks1) > 2*is2_len:
            #print 'long out - 1', i, len(ks1), 2*is2_len
            break

        if (is2_len + len(ks1))>1024:
            #print 'long out - 1', i, is2_len + len(ks1)
            break

        if (len(ks1)%2!=0) and (is1[0:1] == ks1[lll:lll+1]):
            print  'get -success1 > ', i, is1, k
            break

        if len(ks1)>1024:
            #print 'long out', i
            break

        k = k * i*9

    ls2_len1 = is2_len-1
    ks = ks1[:ls2_len1]

    if ((is1 == ks) and (is2 == ks1[(-1*ls2_len1):])):
        print  'get -success > ', i, is1, k
        print ''

    i += 1

    if i % 10000000 == 0:
        print '...', i

    if i > 99999999999999999999:
        break
```

4. 总结

结果最后跑出来是

```
get -success >  12345679 1234567 12345678987654321
```

因为代码中处理字符存为数值是倒着的，所以key应该是97654321