CTF小白学习笔记(Reverse)-i春秋 SimpleGame

原创

I still … 于 2020-11-19 15:49:33 发布 224 收藏 1

分类专栏： CTF的writeup 文章标签： 信息安全 安全

本文链接：https://blog.csdn.net/qq_44370676/article/details/109812211

CTF的writeup 专栏收录该内容

8 篇文章 0 订阅

订阅专栏

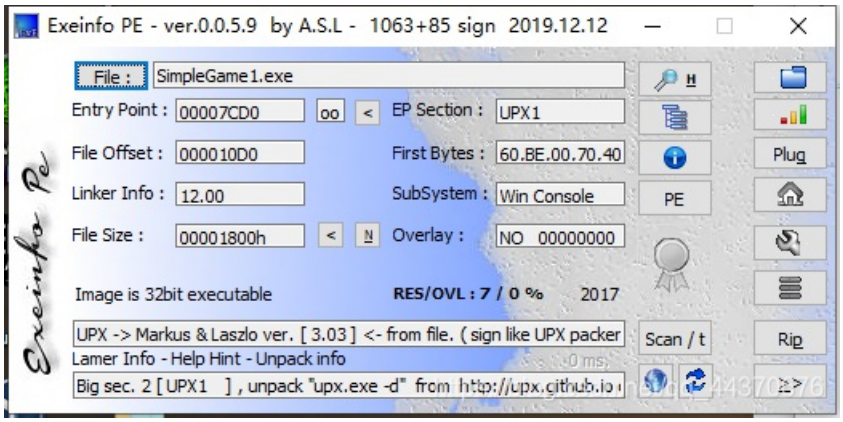
大纲

一.解题过程

静态分析

拿到题目运行，运行下发现是输入flag进行比对的题目。

先查壳：



二话不说，upx脱壳，然后开始ida分析：

main函数:

```
v9 = 0;
result = sub_401200(*(_DWORD *)&Buf);
if ( result != -1 )
{
  sub_4013C0();
  printf("Game Is Start\n");
  printf("Please Input a String\n");
  v4 = _iob_func();
  fgets(&Buf, 37, v4);
  if ( strlen(&Buf) == 36 )
  {
    if ( sub_401360(&Buf, &v6) == -1 )
    {
      printf("Wrong Format!\n");
    }
    else
    {
      if ( sub_401000(&Buf) != -1 && sub_4012C0() != -1 && sub_401300() !
      {
        printf("You Win!\n");
        return 0;
      }
      printf("You Lose!\n");
    }
  }
  else
```


首先检查输入字符串长度是否为36，sub_401360功能是检查输入是否全是小写字母，不是返回-1。这里就不截图了。同样字符串下一步比对过程全在sub_401000，sub_4012C0和sub_401300就不截图了。

来看sub_401000，这段代码有点长，部分变量我也重命名过，input0,input1都是输入字符串,row和col为二维矩阵行列索引

```
17
18  input0 = this;
19  i = 0;
20  v = 0;
21  if ( !strlen(this) )
22    return 1;
23  input1 = this;
24  while ( 1 )
25  {
26    chr = input1[i];
27    if ( chr != dword_403170[i / 2] )
28      return -1;
29    switch ( chr )
30    {
31      case 97:
32        col = 100 - input1[i + 1];
33        goto LABEL_14;
34      case 100:
35        col = input1[i + 1] - 94;
36LABEL_14:
37        chr2 = input1[i + 2];
38        if ( chr2 != dword_403170[(i + 2) / 2] )
39          return -1;
40        if ( chr2 == 115 )
41        {
42          row = input0[i + 3] - 95;
43        }
44        else
45        {
46          if ( chr2 != 119 )
47            return -1;
48          row = 99 - input0[i + 3];
49        }
50LABEL_19:
```


这段代码就不全部截图了，大致流程就是将36个字符分成9组，每组4个字符分别用a0 a1 a2 a3表示，每次对第i组进行操作。每轮循环过后，i + 4。 然后根据a0 a2的情况，用a1 a3计算出row和col值

尝试IAT修复

这里就容易踩坑了，dword_403170中的值在运行时发生了改变。可能是在main函数的sub_4013C0中改动的。所以得在动态调试的过程中dump出来，可是upx脱壳后的程序不能运行。所以用ImportReConstructor进行IAT修复。

IAT修复用到了脱壳前的exe和upx脱壳后的exe，这里用IDA动态调试(7.0版是真的香)

这里用IDA打开脱壳前的exe

alt + t 搜索popa

```
UPX1:00407E33                push    ebx
UPX1:00407E34                push    edi
UPX1:00407E35                call    ebp
UPX1:00407E37                lea     eax, [edi+207h]
UPX1:00407E3D                and     byte ptr [eax], 7Fh
UPX1:00407E40                and     byte ptr [eax+28h], 7Fh
UPX1:00407E44                pop     eax
UPX1:00407E45                push    eax
UPX1:00407E46                push    esp
UPX1:00407E47                push    eax
UPX1:00407E48                push    ebx
```

```
UPX1:00407E49                   push    edi
UPX1:00407E4A                   call    ebp
UPX1:00407E4C                   pop     eax
UPX1:00407E4D                   popa
UPX1:00407E4E                   lea     eax, [esp+2Ch+var_AC]
UPX1:00407E52
UPX1:00407E52 loc_407E52:                               ; CODE XREF: start+186↓j
UPX1:00407E52                   push    0
UPX1:00407E54                   cmp     esp, eax
UPX1:00407E56                   jnz     short loc_407E52
UPX1:00407E58                   sub     esp, 0FFFFFF80h
UPX1:00407E5B                   jmp     loc_401779
UPX1:00407E5B start             endp ; sp-analysis failed
UPX1:00407E5B
```


打个断点后开始调试，这里一路F8，知道运行到0x40170C处

IDA View-EIP

```
UPX0:004016EC db    0
UPX0:004016ED ; ---------------------------------------
UPX0:004016ED
UPX0:004016ED loc_4016ED:
UPX0:004016ED mov     ecx, ___security_cookie+1CCh
UPX0:004016F3 mov     eax, off_402050
UPX0:004016F8 mov     [eax], ecx
UPX0:004016FA push    ___security_cookie+1CCh
UPX0:00401700 push    ___security_cookie+1C8h
UPX0:00401706 push    ___security_cookie+1C4h
EIP UPX0:0040170C call    sub_401400
UPX0:00401711 add     esp, 0Ch
UPX0:00401714 mov     ___security_cookie+1BCh, eax
UPX0:00401719 cmp     ___security_cookie+1C0h, 0
UPX0:00401720 jnz     short loc_401758
UPX0:00401722 push    eax
UPX0:00401723 call    off_402034
UPX0:00401729 mov     ecx, [ebp-14h]
UPX0:0040172C mov     eax, [ecx]
UPX0:0040172E mov     eax, [eax]
UPX0:00401730 mov     [ebp-1Ch], eax
UPX0:00401733 push    ecx
UPX0:00401734 push    eax
```

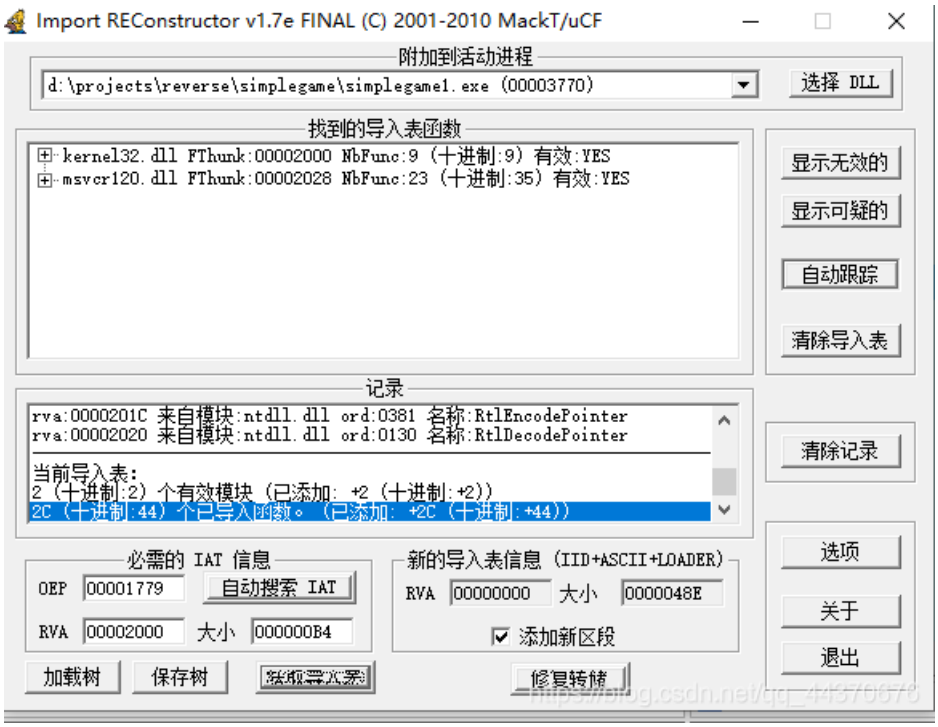

这个sub_401400就是main函数，F7进去。然后运行到下图处

```
UPX0:00401446
UPX0:00401446 loc_401446:                               ; CODE XREF: sub_401400+34↑j
IP UPX0:00401446 push    esi
UPX0:00401447 call    sub_4013C0
UPX0:0040144C mov     esi, off_40209C
UPX0:00401452 push    offset aGameIsStart               ; "Game Is Start\n"
UPX0:00401457 call    esi ; msvcr120_printf
UPX0:00401459 push    offset aPleaseInputASt            ; "Please Input a String\n"
UPX0:0040145E call    esi ; msvcr120_printf
UPX0:00401460 call    off_4020A8
UPX0:00401466 push    eax
UPX0:00401467 lea     eax, [ebp+var_2C]
```


0x40209C处保存着printf函数的地址，切换到堆栈里看看

```
00401FF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00402000  D0 97 CB 76 60 5E CB 76  60 1A CB 76 A0 3C CC 76
00402010  E0 7F CB 76 20 8B CB 76  20 A5 CB 76 B0 67 97 77
00402020  30 5E 97 77 00 00 00 00  DD BE 60 50 47 E8 5B 50
00402030  CF CB 5B 50 A3 51 5C 50  08 BF 60 50 11 50 5C 50
00402040  DB 56 63 50 39 BD 60 50  6F D6 5A 50 39 D6 5A 50
00402050  2C F6 66 50 40 F7 66 50  DD 4D 63 50 98 6C 63 50
00402060  FC 48 63 50 E7 48 63 50  34 DE 62 50 3D DE 5B 50
00402070  D4 ED 59 50 F9 ED 59 50  72 1C 5A 50 48 2B 5A 50
00402080  AA BF 5B 50 8B 47 63 50  83 E1 5B 50 4F 9D 5A 50
00402090  26 69 5A 50 F9 33 5C 50  62 FE 5B 50 D1 31 61 50
004020A0  13 19 61 50 29 21 61 50  25 24 5A 50 38 F6 66 50
004020B0  48 17 5A 50 00 00 00 00  00 00 00 00 C9 15 40 00
004020C0  00 00 00 00 00 00 00 00  10 15 40 00 FD 1A 40 00
```

这里可以看出IAT表是从0x402000 - 0x4020B4,用ImportReConstructor修复下：



但修复后的程序还是无法运行，这里main函数的地址是0x401400, start函数地址是0x401779，都试过了没用。只好在这里接着把403170处的数据dump出来，为：

```
dssdwasawawaaswddw
```

接着静态分析

从403170 dump出来的18个字符为输入字符串偶数位字符(索引0开始，a0 a2 a4 a6…)。也就是接下来我们要分析出奇数位的字符，现在大概明白了每四个4符一组，根据a0 a2不同情况(只能为a d s w 4个中的一个)，用 a1 a3生成row, col值。row, col用法如下

```
LABEL_19:
        if ( row > 5 || col > 5 )
          return -1;
        j = 0;
        index = row + col + 8 * row;
        v9 = (int *)((char *)&unk_4021A0 + 64 * (i / 4));
        do
        {
          ii = 0;
          var = v9;
          do
          {
            if ( *var )
            {
              if ( dword_403020[index + ii] )
                return -1;
              dword_403020[index + ii] = *var;
            }
            ++ii;
            ++var;
          }
          while ( ii < 4 );
          v9 += 4;
          index += 9;
          ++j;
        }
        while ( j < 4 );
        input1 = input0;
        v += 4;
        i = v;
        if ( v >= strlen(input0) )
          return 1;
        break;
      case 115:
```


这里unk_4021A0为9 * 4 * 4的矩阵， dword_403020为 9 * 9矩阵(静态分析 + 动态调试得知)，在程序的9轮循环中 判断unk_4021A0第i个矩阵能否嵌入矩阵dword_403020。嵌入部分以第i轮生成（row，col）坐标为起点4*4的小矩阵中。 能够进行嵌入的条件是嵌入部分对应元素不能同时为非0元素。(比如a[i][j] == 3 b[i + row][j + col] == 15就不能嵌入)，只要对应位置有至少1个非0元素(2个都是0也可以)就可以嵌入。嵌入完成后会修改dword_403020矩阵，并且每轮都能发现有多个潜在位置可以嵌入(但是答案唯一)。

所以现在的问题就是根据unk_4021A0， dword_403020的值推出一组坐标（用list保存，长度为9），这组坐标可以把unk_4021A0 9个矩阵嵌入到dword_403020中。然后根据这组坐标推出输入字符串奇数位。

先把unk_4021A0, dword_403020 dump出来。dump脚本为：

```python
from idaapi import *
import json

def dump_403020():
    start_address = 0x403020
    data_length = 324
    datas = []
    count = 0
    for i in range(0, data_length, 4):
        data = get_byte(start_address + i)
        # if count == 0:
        datas.append(data)
        count += 1
        if count == 4:
            count = 0
    print datas
    print len(datas)
    fp = open('D:\projects\python\IDAscripts\datas\SimpleGame403020.json', 'wb')
    # fp.write(datas)
    json.dump(datas, fp)
    fp.close()


def dump_403170():
    start_address = 0x403170
    data_length = 72
    datas = []
    for i in range(0, data_length, 4):
        data = get_byte(start_address + i)
        # if count == 0:
        datas.append(data)

    print datas
    print len(datas)
    fp = open('D:\projects\python\IDAscripts\datas\SimpleGame403170.txt', 'wb')
    fp.write(''.join(map(chr,datas)))


def dump_4021A0():
    start_address = 0x4021A0
    data_length = 572
    datas = []

    for i in range(0, data_length, 4):
        data = get_byte(start_address + i)
        # if count == 0:
        datas.append(data)
    print datas
    print len(datas)
    fp = open('D:\projects\python\IDAscripts\datas\SimpleGame4021A0.json', 'wb')
    # fp.write(datas)
    json.dump(datas, fp)
    fp.close()


if __name__ == '__main__':
    dump_4021A0()
    dump_403020()
```

dump出来后dword_403020为:

```
[[15 15 15 15 15  0  0 15 15]
 [15  0  0  0  0 15  0 15 15]
 [15  0 15 15 15 15  0  0 15]
 [ 0  0  0  0 15 15 15  0 15]
 [ 0 15  0  0 15  0 15 15 15]
 [15  0 15 15  0  0 15  0 15]
 [15  0  0 15  0  0  0  0 15]
 [15 15  0  0  0  0 15  0 15]
 [15 15 15 15 15 15 15 15 15]]
```

unk_4021A0为：

```
[[[0 0 9 0]
  [0 9 9 0]
  [0 0 9 0]
  [0 0 0 0]]

 [[0 6 0 0]
  [0 6 0 0]
  [0 6 0 0]
  [0 6 0 0]]

 [[0 0 0 0]
  [0 3 0 0]
  [3 3 0 0]
  [3 0 0 0]]

 [[0 0 8 0]
  [0 0 8 0]
  [0 8 8 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [0 5 5 0]
  [0 5 5 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [2 2 2 2]
  [0 0 0 0]
  [0 0 0 0]]

 [[0 7 0 0]
  [0 7 7 0]
  [0 0 7 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [0 0 4 0]
  [0 0 4 0]
  [0 0 0 0]]

 [[0 1 1 0]
  [0 0 1 0]
  [0 0 1 0]
  [0 0 0 0]]]
```

这里用dfs来解除坐标数组，解题脚本如下，得到坐标后根据前面的row col生成的算法就可以得到flag:

```
flag{dcsdscdbwbadsdabwaacwcacadsdwbdcdbwc}
```

二.解题脚本

```
import numpy as np
import json

ans = []
res = []

def dfs(step, a, b):
    # 答案唯一，找齐9个坐标dfs结束
    if step == 9:
        #print(ans)
        res.extend(ans)
        return
    else:
        # 找出所有能够嵌入的(row,col)坐标，并保存到pos中
        pos = []
        for row in range(6):
            for column in range(6):
                sum = 0
                for i in range(4):
                    for j in range(4):
                        if a[step, i, j]:
                            if b[i + row, j + column]:
                                break
                        sum += 1
                if sum == 16:
                    pos.append((row, column))

        # 对已找出的所有坐标进行嵌入操作
        for position in pos:
            # 1.嵌入
            for i in range(4):
                for j in range(4):
                    if a[step, i, j]:
                        b[i + position[0], j + position[1]] = a[step,i, j]
            # 2.开始dfs
            ans.append(position)
            dfs(step + 1, a, b)
            # 3.dfs结束后还原9*9的矩阵
            for i in range(4):
                for j in range(4):
                    if a[step, i, j] != 0:
                        b[i + position[0], j + position[1]] = 0
            ans.pop()

if __name__ == '__main__':
    unk_4021A0_file = '../datas/simpleGame/SimpleGame4021A0.json'
    unk_403020_file = '../datas/simpleGame/SimpleGame403020.json'
    unk_403170_file = '../datas/simpleGame/SimpleGame403170.txt'

    matrix_4021A0 = np.array(json.load(open(unk_4021A0_file))).reshape(9, 4, 4)
    matrix_403020 = np.array(json.load(open(unk_403020_file))).reshape(9, 9)

    print(matrix_4021A0)
```

```
data_403170 = [c for c in open(unk_403170_file,'rb').read()]

dfs(0, matrix_4021A0, matrix_403020)
print(res)

flag = ''

for i in range(9):
    row, col = res[i]
    a0 = data_403170[2 * i]
    a2 = data_403170[2 * i + 1]

    if a0 == 97:
        a1 = 100 - col
        if a2 == 115:
            a3 = row + 95
        elif a2 == 119:
            a3 = 99 - row

    elif a0 == 100:
        a1 = 94 + col
        if a2 == 115:
            a3 = row + 95
        elif a2 == 119:
            a3 = 99 - row

    elif a0 == 115:
        a1 = row + 95
        if a2 == 97:
            a3 = 100 - col
        elif a2 == 100:
            a3 = col + 94

    elif a0 == 119:
        a1 = 99 - row
        if a2 == 97:
            a3 = 100 - col
        elif a2 == 100:
            a3 = col + 94

    flag += ''.join(list(map(chr,[a0, a1, a2, a3])))

flag = 'flag{' + flag + '}'
print(flag)
```

三.总结

这道题出现了久违的二维数组，甚至三维数组也来了，着实吐血。不过自己对于手动脱壳，IAT修复的了解几乎为0，以至于动态调试时要用脱壳前的程序来调试。希望路过的各位大佬能赐教赐教。