2017看雪秋季赛 第三题

原创

Flying_Fatty 于 2017-10-31 20:57:28 发布 275 收藏
分类专栏： CTF之旅 reverse 文章标签： CTF 解密

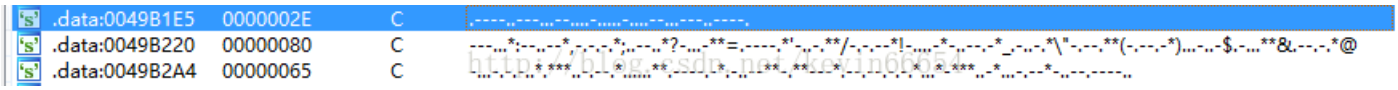
本文链接：https://blog.csdn.net/kevin66654/article/details/78407988


CTF之旅 同时被 2 个专栏收录
84 篇文章 2 订阅
订阅专栏


reverse
24 篇文章 0 订阅
订阅专栏

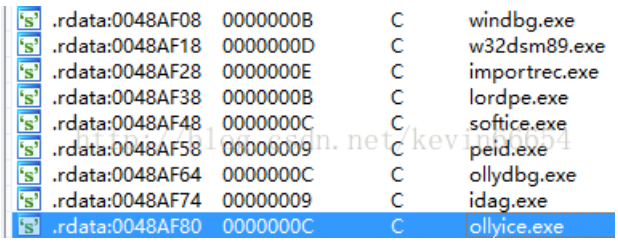
自己做的时候。

照样IDA先找找字符串，会发现这个。



经常打CTF，做misc和crypto的话，第一反应这是一个莫尔斯电码。可能和加密数据有关系。

OD调试单步可以走到这里来，DialogBoxParamA是弹框的函数。再单步执行点击验证会崩溃。程序大概率是加了反调试的（每次碰到这种我就傻逼了）



IDA里搜索字符串很明显可以找到这么多的动态工具。一定是加了各种反调试的。

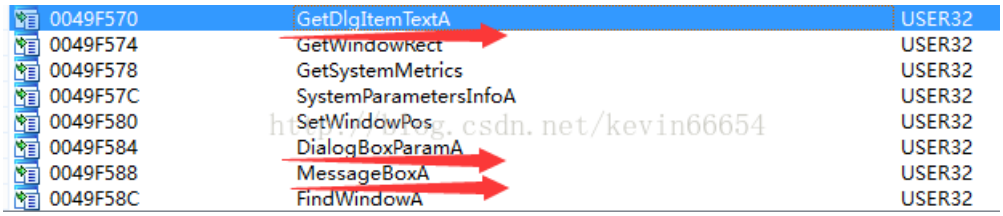
始学习之

下面开始学习之路。

官方题解

先来记录下看到的对我印象很大的一句话：新手用OD，高手看汇编！以后会大大改善自己的学习习惯和方法的。

要习惯IDA来找算法入口，直接绕过所有的调试和反调试。



这些函数是可以用来读取用户输入的。

View - Open Subview - Cross References

利用好IDA的字符串功能和导入函数imports和导出函数功能，OD的中文智能搜索也是一个好工具。

提示我们字符串长不够

（从这里就能猜到base64的使用，个人觉得还是很牵强。。。但是可以作为一种思路猜想，去验证）

```
if ( (unsigned __int8)sub_42D9AB(&unk_49B000, &v16) == 1 )
{
  MessageBoxA(0, "ok", "CrackMe", 0);
  sub_42DE51();
}
```

看到了MessageBoxA的ok！激动不激动，说明找到了正确的处理地方。

在这里

```
1 int __stdcall sub_434EF0(HWND hDlg, int a2, int a3, int a4)
2 {
```

通过交叉引用找到了对的地方。函数434EF0。

```
if ( v11 == 272 )
{
  v27 = sub_42D4F1();
  if ( v27 == 1 )
    ExitProcess(0);
  v27 = 0;
  v27 = sub_42E428();
  if ( v27 == 1 )
    ExitProcess(0);
  v27 = 0;
  v27 = sub_42D825();
  if ( v27 == 1 )
    ExitProcess(0);
  sub_42D14F(hDlg, 1);
```

这么多的Exit，很明显就是反调试检测到了些什么东西，然后强制程序退出。可以直接看后面的算法部分。

从GetDlgItemTextA看起，第三个参数是我们的输入，标记一下。

```
v4 = GetDlgItemTextA(hDlg, 1001, &myinput, 1025);
```

然后往下看，哪些地方用了我们的input。

```
memset(&v22, 0, 1023);
base64decode((int)&myinput, 1024, (int)&inputmiddle);
v19 = 0;
memset(&v20, 0, 1023);
base64decode((int)&inputmiddle, 1024, (int)&inputdecode);
```

这里为什么是memset和base64的解密函数等会儿解答，从这里我们可以看到是把我们的输入经过了两次相同的处理，然后保存了起来，继续往下看。

```
sub_42D96A((int)&inputdecode, (int)&v19, 1024);
v18 = 3;
SM3encode(&inputdecode, 3, afterSM3decode);
for ( i = 0; i < 32; ++i )
  sub_42DF05(&SM3Hash[2 * i], "%02x", (unsigned __int8)afterSM3decode[i]);
v5 = sub_42D794((int)SM3Hash);
v6 = &myinput + sub_42D794((int)&myinput);
v7 = sub_42D794((int)SM3Hash);
```

然后，我们的inputdecode（就是经过两次相同处理后的input），分别进入了两个不同的函数处理

得到了v19和afterSM3decode（这个等会儿解释）。sub_42DF05函数是把这个值转换成了16进制（02x，不够的前置填充0），所以，SM3Hash这个字符串是64字节长度的。

v6呢，是我们的输入连接上一个啥玩意，相当于输入

```
v6 = &myinput + sub_42D794((int)&myinput);
v7 = sub_42D794((int)SM3Hash);
if ( !sub_42DB27((int)SM3Hash, (int)&v6[-v7], v5) )
{
  sub_42D0B4(v11, v12, v13);
  if ( (unsigned __int8)j_migong((int)mymap, (int)&v19) == 1 )
  {
    v8 = MessageBoxA(0, "ok", "CrackMe", 0);
    sub_42DE51(&v11 == &v11, v8);
  }
}
```

这里看到，v6和我们的SM3Hash有个比较，那一定是字符串比较对吧

看到MessageBoxA是输出我们的ok成功字符串，那这个if语句也要成功

基本逻辑分析完了，我们需要解决的几个函数：为什么是base64，SM3，迷宫，还有一开始看到的莫尔斯在哪儿？程序是怎么匹配的。

memset函数戳进去，发现基本都是赋值0的语句。

而且在都是在函数调用和运算之前，那么很大可能性是初始化赋值0的语句

一般写法是memset（a，0，sizeof（a）），正好符合IDA反汇编之后的格式了

base64函数：点进去看到的是sub_434990.

看到了%4，看到了=号，看到了对不同的模数分情况讨论

```
case 1u:                                // len mod 4 == 1
  *(_BYTE *)(v9++ + a3) += (v7 >> 4) & 3;
  if ( i < a2 - 3 || *(_BYTE *)(a2 + a1 - 2) != '=' )
    *(_BYTE *)(v9 + a3) = 16 * (v7 & 0xF);
  break;
case 2u:
  *(_BYTE *)(v9++ + a3) += (v7 >> 2) & 0xF;
  v8 = sub_42E26B();
  if ( v8 == 1 )
    sub_42E086();
  v8 = 0;
  v8 = sub_42D7F8();
  if ( v8 == 1 )
    sub_42E086();
  if ( i < a2 - 2 || *(_BYTE *)(a2 + a1 - 1) != '=' )
    *(_BYTE *)(v9 + a3) = (v7 & 3) << 6;
  break;
case 3u:
```


见过C语言实现的base64encode和base64decode解码的会习惯这种<<和>>去操作每一个字符ascii码值，然后去相加计算的

因为是每次取4个，变成3个。所以是base64的解密过程。

SM3：会跳转到sub_437E70函数。

```
memset(&v5, 0xCCu, 0x1B4u);
v7 = (unsigned int)&savedregs ^ *(_DWORD *)&mymap[836];
sub_42D294(&v6);
sub_42DF2D(&v6, a1, a2);
sub_42D15E(&v6, a3);
memset((int)&v8, 0, 232);
sub_42D65E(&savedregs, &dword_437F18);
v3 = sub_42D1E5((unsigned int)&savedregs ^ v7);
return sub_42DE51(1, v3);
```


看到这个玩意，每个函数都点一点，因为是个加密解密模块，可能会有MAGIC（每个加解密函数独有的字段，也就是特征值）

```
memset(&v3, 0xCCu, 0xCCu);
*(_DWORD *)a1 = 0;
*(_DWORD *)(a1 + 4) = 0;
*(_DWORD *)(a1 + 8) = 0x7380166F;
*(_DWORD *)(a1 + 12) = 0x4914B2B9;
*(_DWORD *)(a1 + 16) = 0x172442D7;
*(_DWORD *)(a1 + 20) = 0xDA8A0600;
*(_DWORD *)(a1 + 24) = 0xA96F30BC;
*(_DWORD *)(a1 + 28) = 0x163138AA;
*(_DWORD *)(a1 + 32) = 0xE38DEE4D;
*(_DWORD *)(a1 + 36) = 0xB0FB0E4E;
```


百度一下，发现是SM3压缩加密，是一个摘要算法，弄出来就是64字节，也符合了我们的猜想，为什么转成另外一个数组

迷宫怎么猜到的：

那个if语句需要为真，v19是我们的input经过两次base64解密，再转换的（无论怎么样，都是输入）

那么这个函数和第一个参数就是我们需要认真查看的东西

函数在435400，看到了四个判断，分别是z，l，p，q，猜测是上下左右，那么第一个参数就会是地图

如何对应上呢？

分析一个p作为例子，判断的是v7+v11>=0，发现v7是一个常数-1，那么v11一定是当前坐标的某一个，然后发现v改变的是4个单位，如果是4个单位为一步的话，那么减一步就是向左了。截图如下：

```
if ( *a2 == 'p' && v7 + v11 >= 0 )        // v7 == -1
{
  v2 = (_BYTE *)(a1 + 40 * v10);
  if ( *(_DWORD *)&v2[4 * (v7 + v11)] )   // change 4 * v7, left
  {
```

同理分析我们可以得到，z下，l右，q上，p左

那么去数组里找需要的地图

刚才在函数里看到了，4个为一组，40个为行的跳跃。所以10个字符为一行。把数据扣出来得到地图

```
0111111110
0011111000
1000001011
11111010#1
1000101001
1010001011
1011111001
1000011100
1111000010
1111111000
```

有个值是不能走的在地图里用#标注，#的本来的意思是排除迷宫的多解，因为有（8，3）的检验。从（0，0）开始，走到终点？

这就是这个题的bug了！只判断了地图中不能走到1的位置，所以从左上角出发，不碰到障碍物即可，不需要走到终点，先按照作者意思来

那么，我的输入转换之后是：

```
zlzllllzzzppqppzzzlllzlllzllqqpqpqqqqqllq
```

现在就到了需要研究函数42D96A的时候了，调用的是435DE0.

```
if ( *(_BYTE *)(v12 + input) == ' ' || *(_BYTE *)(v12 + input) == '/' )
{
  if ( *(_BYTE *)(v12 + input) != ' ' || *(_BYTE *)(v12 + input - 1) == '/' )// current char == '/'
  {
    if ( *(_BYTE *)(v12 + input) == '/' )
      *(_BYTE *)(v10++ + output) = '/';
  }
  else                                  // current char == ' '
  {
```

首先可以看到的是判断字符是空格还是斜杠

```
if ( (unsigned __int8)sub_42D0DC(v13, (int)&v8) != 1 || v11 >= 5 )
{
  if ( (unsigned __int8)sub_42D7B2(v13, &v8) == 1 )
  {
    *(_BYTE *)(v10++ + output) = v8;
  }
  else if ( sub_42E414(v13, (int)&v8) == 1 )
  {
    *(_BYTE *)(v10++ + output) = v8;
  }
  else
  {
    sub_42E22A("error !\n", v6);
  }
```


然后对于空格的分为三类，42D7B2处理一类，42E414处理一类，另外一个函数是报错的

42D7B2戳开，调用435AA0函数，发现了这个

```
memset(&v4, 0xCCu, 0xD8u);
v6 = 0;
v5 = 0;
while ( v6 < 10 )
{
  if ( *((_BYTE *)&unk_49B1E0 + 5 * v6) == *(_BYTE *)a1
    && *((_BYTE *)&unk_49B1E1 + 5 * v6) == *(_BYTE *)(a1 + 1)
    && *((_BYTE *)&unk_49B1E2 + 5 * v6) == *(_BYTE *)(a1 + 2)
    && *((_BYTE *)&unk_49B1E3 + 5 * v6) == *(_BYTE *)(a1 + 3)
    && byte_49B1E4[5 * v6] == *(_BYTE *)(a1 + 4) )
  {
    v2 = v6 + '0';
    *a2 = v6 + '0';
    LOBYTE(v2) = 1;
    return sub_42DE51(1, v2);
  }
```

看到加‘0’，那一定是转换0-9

然后找到if里面的调用的435D00，看到了这个

```
char __cdecl sub_435D00(int a1, _BYTE *a2)
{
  char v3; // [sp+Ch] [bp-CCh]@1
  int i; // [sp+D0h] [bp-8h]@1

  memset(&v3, 0xCCu, 0xCCu);
  for ( i = 0; i < 26; ++i )
  {
    if ( *((_BYTE *)&unk_49B2A0 + 4 * i) == *(_BYTE *)a1
      && *((_BYTE *)&unk_49B2A1 + 4 * i) == *(_BYTE *)(a1 + 1)
      && *((_BYTE *)&unk_49B2A2 + 4 * i) == *(_BYTE *)(a1 + 2)
      && byte_49B2A3[4 * i] == *(_BYTE *)(a1 + 3) )
    {
      *a2 = i + 'a';
      return 1;
    }
  }
```

这个26和‘a’说明了一定是在处理小写字母，我们戳开49B2A0，可以发现：和我们想要的莫尔斯联系上了

综上，我们的逻辑思路和程序的判断思路已经对上了。

输入进来先两次base64的解码，再用莫尔斯电码转成zlpq的方向走迷宫，从左上角到右上角，成功即可

看到匹配时候的负号，说明是倒着匹配的，那么说明SM3作为后缀添加上的

（可能觉得练习得很牵强，怎么就猜到了，逆向提供的只是一个框架，一个思路，剩下的靠验证就可以）

去找到zlpq的莫尔斯

a ·-	i ··	r ·-·	1 ·----
ä ·-·-	j ·---	s ···	2 ··---
b -···	k -·-	t -	3 ···--
c -·-·	l ·-··	u ··-	4 ····-
ch ----	m --	ü ··--	5 ·····
d -··	n -·	v ···-	6 -····
e ·	o ---	w ·--	7 --···
f ··-·	ö ---·	x -··-	8 ---··
g --·	p ·--·	y -·--	9 ----·
h ····	q --·-	z --··	0 -----

于是得到了这个

```
s = 'zlzllllzzzppqppzzzlllzlllzllqqpqpqqqqqllq'
m = ""
for i in s:
 if i == 'z':
  m += '--..'
 elif i == 'l':
  m += '.-..'
 elif i == 'p':
  m += '.--.'
 else:
  m += '--.-'
print m
#m = '--...-..--...-...-...-...-..--..--..--...--...--.--.-.--..--.--..--...-...-...-..--...-...-...-..
--...-...-..--.---.-.--.--.-.--.--.---.---.---.---.-.-...-..--.-'
```

```
if ( *(_BYTE *)(v12 + input) != ' ' || *(_BYTE *)(v12 + input - 1) == '/' )// current char == '/'
{
  if ( *(_BYTE *)(v12 + input) == '/' )
    *(_BYTE *)(v10++ + output) = ' ';
}
```


以这个字符作为结尾，所以放进去

再用base64两次加密，第一次得到

```
LS0uLi4tLi4tLS4uLi0uLi4tLi4uLS4uLi0uLi0tLi4tLS4uLS0uLi4tLS4uLS0uLS0uLS4tLS4uLS0uLS0uLi0tLi4tLS4uLi0uLi4tLi
uLi4tLi4uLS4uLi0uLi0tLi4uLS4uLi0uLi0tLi0tLS4tLi0tLi0tLi0uLS0uLS0uLS0tLi0tLS4tLS0uLS0tLi0uLS4uLi0uLi0tLi0v
```

到了这里，就需要用到patch反调试了，不然没法动态OD看到SM3的加密结果