BUU-[ACTF新生赛2020]usualCrypt

原创

Holy-Pang 于 2021-04-19 10:48:12 发布 84 收藏

分类专栏： RE-WP 文章标签： base64

本文链接：https://blog.csdn.net/qq_45771413/article/details/115859334

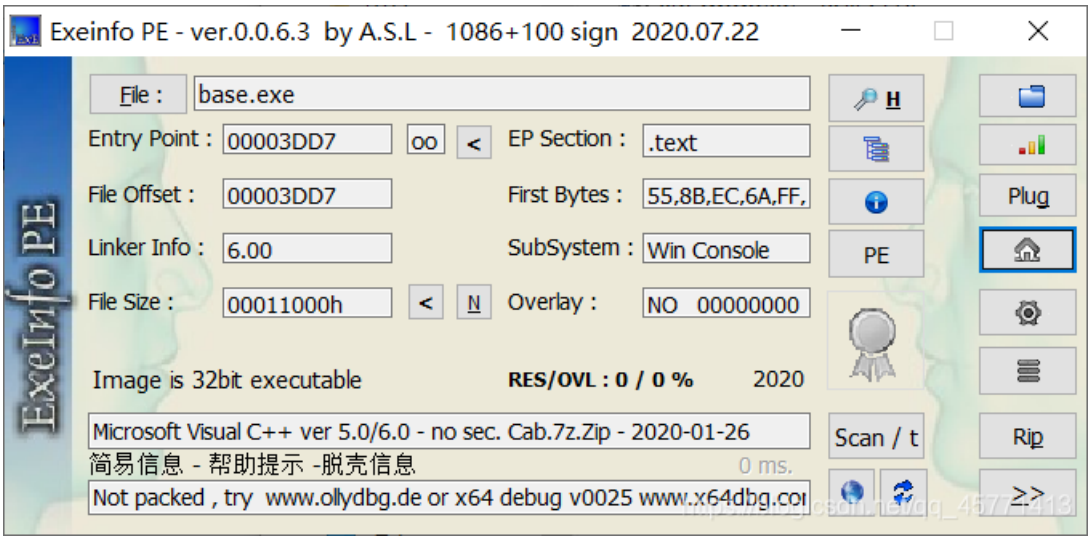
RE-WP 专栏收录该内容

17 篇文章 0 订阅

订阅专栏

查壳

无壳



IDA

进入关键函数：

题目叫base，这里的伪代码也确实像base。接下来就该找table了。

```
16  v3 = 0;
17  v4 = 0;
18  sub_401000();
19  v5 = a2 % 3;
20  v6 = a1;
21  v7 = a2 - a2 % 3;
22  v15 = a2 % 3;
23  if ( v7 > 0 )
24  {
25    do
26    {
27      LOBYTE(v5) = *(_BYTE *)(a1 + v3);
28      v3 += 3;
29      v8 = v4 + 1;
30      *(_BYTE *)(v8 + a3 - 1) = byte_40E0A0[(v5 >> 2) & 0x3F];
31      *(_BYTE *)(++v8 + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(a1 + v3 - 3) & 3)
32                                            + (((int)*(unsigned __int8 *)(a1 + v3 - 2) >> 4) & 0xF)];
33      *(_BYTE *)(++v8 + a3 - 1) = byte_40E0A0[4 * (*(_BYTE *)(a1 + v3 - 2) & 0xF)
34                                            + (((int)*(unsigned __int8 *)(a1 + v3 - 1) >> 6) & 3)];
35      v5 = *(_BYTE *)(a1 + v3 - 1) & 0x3F;
36      v4 = v8 + 1;
37      *(_BYTE *)(v4 + a3 - 1) = byte_40E0A0[v5];
38    }
39    while ( v3 < v7 );
40    v5 = v15;
41  }
42  if ( v5 == 1 )
43  {
44    LOBYTE(v7) = *(_BYTE *)(v3 + a1);
45    v9 = v4 + 1;
46    *(_BYTE *)(v9 + a3 - 1) = byte_40E0A0[(v7 >> 2) & 0x3F];
47    v10 = v9 + 1;
```


在伪代码的位移和与操作中，byte_40e0a0重复出现。大概率是table。

但是byte_40E0A0缺斤短两，一看就不是正常table。

```
.data:0040E0A0 ; char byte_40E0A0[10]
.data:0040E0A0 byte_40E0A0     db 'A'                  ;
.data:0040E0A0                                         ;
.data:0040E0A1                 db  42h ; B
.data:0040E0A2                 db  43h ; C
.data:0040E0A3                 db  44h ; D
.data:0040E0A4                 db  45h ; E
.data:0040E0A5                 db  46h ; F
.data:0040E0A6                 db  47h ; G
.data:0040E0A7                 db  48h ; H
.data:0040E0A8                 db  49h ; I
.data:0040E0A9                 db  4Ah ; J
.data:0040E0AA ; char byte_40E0AA[]
```


这里倒回去看关键函数的时候发现定义变量的时候还有个 sub_401000函数：

将byte_40E0AA的6到14位与byte_40E0A0的6到14位进行调换。但是问题来了，40E0A0定义时只有10位，它是如何做到6~14位操作的……

```
1 int sub_401000()
2 {
3   int result; // eax
4   char v1; // cl
5
6   for ( result = 6; result < 15; ++result )
7   {
8     v1 = byte_40E0AA[result];
9     byte_40E0AA[result] = byte_40E0A0[result];
10    byte_40E0A0[result] = v1;
11  }
12  return result;
13 }
```



```
.data:0040E0A0 ; char byte_40E0A0[10]
.data:0040E0A0 byte_40E0A0     db 'A'                  ;
.data:0040E0A0                                         ;
.data:0040E0A1                 db  42h ; B
.data:0040E0A2                 db  43h ; C
.data:0040E0A3                 db  44h ; D
.data:0040E0A4                 db  45h ; E
.data:0040E0A5                 db  46h ; F
.data:0040E0A6                 db  47h ; G
.data:0040E0A7                 db  48h ; H
.data:0040E0A8                 db  49h ; I
.data:0040E0A9                 db  4Ah ; J
.data:0040E0AA ; char byte_40E0AA[]
```



```
.data:0040E0AA ; char byte_40E0AA[]
.data:0040E0AA byte_40E0AA     db 4Bh                  ; DATA XREF: sub_401000+B↑r
.data:0040E0AA                                         ; sub_401000+11↑w
.data:0040E0AB aLmnopqrstuvwxy db 'LMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/',0
.data:0040E0E1                 align 4
```

这里百思不得其解,后面的base操作伪代码虽然现在还不熟，但是后面全都用40E0A0进行操作这就更迷惑了，题目都写着base，只用10个字符进行加密显然离谱。

为解开部分迷惑，直接动调查看40100函数调用后table的情况：

```
.data:0040E0A0 byte_40E0A0 db 'A'                      ; DATA XREF: sub_401000:loc_401005↑r
.data:0040E0A0                                         ; sub_401000+17↑w ...
.data:0040E0A1 db  42h ; B
.data:0040E0A2 db  43h ; C
.data:0040E0A3 db  44h ; D
.data:0040E0A4 db  45h ; E
.data:0040E0A5 db  46h ; F
.data:0040E0A6 db  51h ; Q
.data:0040E0A7 db  52h ; R
.data:0040E0A8 db  53h ; S
.data:0040E0A9 db  54h ; T
.data:0040E0AA ; char byte_40E0AA[]
.data:0040E0AA byte_40E0AA db 'U'                      ; DATA XREF: sub_401000+B↑r
.data:0040E0AA                                         ; sub_401000+11↑w
.data:0040E0AB aLmnopqrstuvwxy db 'VWXYPGHIJKLMNOZabcdefghijklmnopqrstuvwxyz0123456789+/',0
.data:0040E0E1 align 4
.data:0040E0E4 ; char byte_40E0E4[]
```


40E0A0到40E0AA连起来刚好是个变种base64 table，好奇葩_(´ཀ`」 ∠)_

ABCDEFQRSTUVWXYPGHIJKLMNOZabcdefghijklmnopqrstuvwxyz0123456789+/***
根据做题经验就当这个是table接着往下进行。来到401030函数：
这里很好理解了，大小写转换。

```
int __cdecl sub_401030(const char *a1)
{
  __int64 v1; // rax
  char v2; // al

  v1 = 0i64;
  if ( strlen(a1) )
  {
    do
    {
      v2 = a1[HIDWORD(v1)];
      if ( v2 < 97 || v2 > 122 )
      {
        if ( v2 < 65 || v2 > 90 )
          goto LABEL_9;
        LOBYTE(v1) = v2 + 32;
      }
      else
      {
        LOBYTE(v1) = v2 - 32;
      }
      a1[HIDWORD(v1)] = v1;
LABEL_9:
      LODWORD(v1) = 0;
      ++HIDWORD(v1);
    }
    while ( HIDWORD(v1) < strlen(a1) );
  }
  return v1;
}
```


最后得到的字符串和 'zMXHz3TlgnxLxJhFAdtZn2fFk3lYCrtPC2l9'比较，相同则成功。

解题

将字符串 'zMXHz3TlgnxLxJhFAdtZn2fFk3lYCrtPC2l9'进行大小写互换。然后用猜测的变表进行base解码得到flag。

EXP

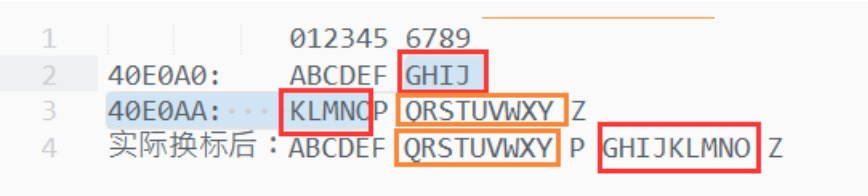
```
arr2='zMXHz3TIgnxLxJhFAdtZn2fFk3lYCrtPC2l9'
arr=''
for s in arr2:
    s=ord(s)
    if(s>64 and s<91):
        s+=32
    elif(s>90 and s<123):
        s-=32
    arr+=chr(s)
print(arr)

#table='ABCDEFQRSTUVWXYGHIJKLMNOPZabcdefghijklmnopqrstuvwxyz0123456789+/'
table='ABCDEFQRSTUVWXYPGHIJKLMNOZabcdefghijklmnopqrstuvwxyz0123456789+/'

byte_code=''
origin_code=''
origin_str=''
for s in arr:
    if(s=='='):
        byte_code=list(byte_code)
        byte_code.pop(-1)
        byte_code.pop(-1)
        byte_code=(''.join(byte_code))
    else:
        byte_code+="{0:0>6}".format(format(table.find(s),'b'))
count=0
for s in byte_code:
    origin_code+=s
    count+=1
    if(count%8==0):
        origin_str+=chr(int(origin_code,2))
        origin_code=''
print(origin_str)
```

flag

flag{bAse64_h2s_a_Surprise}

坑&填坑

为什么 40E0A0和40E0AA合起来才是table。根据初始状态和换表后状态对比可以发现，40E0A0包含了40E0AA，但是40E0AA不包含40E0A0。而40E0A0是 char [10],为什么会这样依旧不知道



动调发现之前研究的别人wp的换表table是错的，P的位置不对_(´ཀ`」 ∠)_
实际换表如上图所示，红色拼接移到P后Z前，黄色移到P前ABCDEF后