看雪CTF2017秋季赛第五题分析

原创

inquisiter 于 2018-01-20 09:23:25 发布 1099 收藏


```
void __stdcall keygen(SomeArr *computed, const char *input)
{
  signed int v2; // ebx
  int v3; // edi
  char v4; // al
  char *base18ed; // [esp+0h] [ebp-4h]
  signed int base18_len; // [esp+10h] [ebp+Ch]

  base18ed = (char *)malloc(3 * strlen(input));
  compute_base18(input, base18ed);              // 这里先看输入输出的值，然后在函数里参看关键点的值，进行函数分
                                                // 把字符串按62进制转换成整形数据，把整形数据转换成18进制表示的
  base18_len = strlen(base18ed);                // base18ed
                                                // 012E77A8  45 44 41 48 45 34 35 30  EDAHE450
                                                // 012E77B0  43 37 34 31 47 48 34 34  C741GH44
                                                // 012E77B8  31 45 31 31 42 48 38 34  1E11BH84
  v2 = 0;
  if ( base18_len > 0 )
  {
    v3 = base18_len;
    do
    {
      v4 = base18ed[v2];
      if ( v4 < '0' || v4 > '9' )               // 18进制字符串转换成整形
      {
        if ( v4 >= 'A' && v4 <= 'H' )
          v3 = v4 - '7';
      }
      else
      {
        v3 = v4 - '0';
      }
      mult_arrays_pow(v3 / 3, v3 % 3 + 1, (int)computed);
      ++v2;
    }
    while ( v2 < base18_len );
  }
  free(base18ed);
}
```

这里computed_base18对内部字符串和输入字符串进行变换，经过mult_arrays_pows对computed进行操作。最后让computed和初始化的一个块进行对比。

```
char *__stdcall compute_base18(const char *input, char *out)
{
  int v2; // eax
  char v3; // al
```

```
int v4; // ebx
_DWORD *v5; // eax
int v6; // ebx
char v7; // al
char *result; // eax
signed int v9; // [esp+10h] [ebp-208h]
int i; // [esp+14h] [ebp-204h]
BigInt v11; // [esp+18h] [ebp-200h]
BigInt res2; // [esp+60h] [ebp-1B8h]
char v13; // [esp+A0h] [ebp-178h]
int v14; // [esp+A4h] [ebp-174h]
unsigned int res1; // [esp+A8h] [ebp-170h]
int v16; // [esp+ECh] [ebp-12Ch]
BigInt num_18; // [esp+F0h] [ebp-128h]
BigInt num_62; // [esp+138h] [ebp-E0h]
BigInt inputa; // [esp+180h] [ebp-98h]
char v20; // [esp+1C8h] [ebp-50h]

v14 = -1;
v13 = 0;
memset(&res2, 0, 0x40u);
initStruct(1, 0, (int)&res1);
initStruct(62, 0, (int)&num_62);
initStruct(18, 0, (int)&num_18);
v9 = strlen(input);
v2 = 0;
for ( i = 0; v2 < v9; i = v2 )
{
  v3 = input[v2];
  if ( v3 < '0' || v3 > '9' )                   // 字符串转换成单个62为单位的十进制值
  {
    if ( v3 < 'A' || v3 > 'Z' )
    {
      if ( v3 < 'a' || v3 > 'z' )
        v4 = -1;
      else
        v4 = v3 - 0x3D;
    }
    else
    {
      v4 = v3 - 0x37;
    }
  }
  else
  {
    v4 = v3 - 0x30;
  }
  if ( v4 != -1 )                               // 第二次变换    按权重乘以相应个数的62
  {
    initStruct(v4, 0, (int)&inputa);            // 第一次执行inputa=v4
    v5 = BigIntMultiply((int)&res1, &v20, (int)&inputa);// res
                                                // 100161AD0  00 00 3E C4 23 89 B3 3B  ..>?壇;
                                                // 00161AD8  44 8F 58 4E 25 00 00 00  D膚N%...
    HP_Plus((int)&res2, &v11, (int)v5);         // res2
                                                // 00161A88  88 41 DD 2E 43 36 6E 25  圜?C6n%
                                                // 00161A90  5E 45 C0 38 04 00 00 00  ^E?...
                                                // 这个res2应该是字符串转换成以62进制的数值
                                                // 438c0455e256e36432edd4188
    HP_Mutiply((char *)&res1, (int)&num_62, (int *)&v11);
  }
```

```
      v2 = i + 1;
    }                                                   // 经过上个循环变换，接下来的循环会利用res1来进行计算。这里猜
    v6 = 0;
    if ( v14 >= 0 )
    {
      while ( 1 )                                       // 第三次变换     62进制转换为18进制
      {
        qmemcpy(&res1, BigIntMod((int *)&res2, (int *)&v11, (int)&num_18), 0x48u);
        qmemcpy(&res2, BigIntDiv((int *)&res2, (int *)&v11, (int)&num_18), 0x48u);
        if ( res1 <= 9 && !v16 )                        // 第四次变换  18进制转换
          break;

        if ( res1 - 10 <= 7 && !v16 )
        {
          v7 = res1 + '7';
          goto LABEL_22;
        }
        if ( v16 != -1 || res1 )
        {
          printf("error %d %d \n", res1, v16);
          goto LABEL_28;
        }
        out[v6] = '0';
LABEL_26:
        ++v6;
LABEL_28:
        if ( v14 < 0 )
          goto LABEL_29;
      }
      v7 = res1 + '0';
LABEL_22:
      out[v6] = v7;
      goto LABEL_26;
    }
LABEL_29:
    result = out;
    out[v6] = 0;
    return result;
}
```

这里分析了半天，原因是一下这几句。参数很多，根本搞不清输入输出到底是那个

```
initStruct(v4, 0, (int)&inputa);          // 第一次执行inputa=v4
v5 = BigIntMultiply((int)&res1, &v20, (int)&inputa);
HP_Plus((int)&res2, &v11, (int)v5);       // res2
HP_Mutiply((char *)&res1, (int)&num_62, (int *)&v11);
```

最后发现接下来利用的几个参数中有res2和res1，

```
qmemcpy(&res1, BigIntMod((int *)&res2, (int *)&v11, (int)&num_18), 0x48u);
qmemcpy(&res2, BigIntDiv((int *)&res2, (int *)&v11, (int)&num_18), 0x48u);
```

通过分析这几个关键的参数的分析，可以大致得到整个函数的算法详情。
如果跟进BigIntMultiply、HP_Plus、HP_Mutiply三个函数，发现分析起来很麻烦。这里逆向的话，还是要根据关键参数的分析来分析算法的具体流程，如果完全跟进的话会比较麻烦。

```
sa="KanXueCrackMe2017"
```

```
sb="EDAHE450C741GH441E11BH84"
mid=33447625194439709684754318989 6
aplb="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
apls="abcdefghijklmnopqrstuvwxyz"
#[7, 1, 0, 2, 40, 22, 46, 38, 36, 53, 12, 40, 56, 33, 49, 36, 20]
sa=sa[::-1]
print sa
sb=sb[::-1]
print sb
print mid
def str_to_sixtytwo():
    l=[]
    for i in range(len(sa)):
        if(ord(sa[i])>=ord('0') and ord(sa[i])<=ord('9')):
            l.append(ord(sa[i])-ord('0'))
        if(ord(sa[i])>=ord('a') and ord(sa[i])<=ord('z')):
            l.append(ord(sa[i])-ord('='))
        if(ord(sa[i])>=ord('A') and ord(sa[i])<=ord('Z')):
            l.append(ord(sa[i])-ord('7'))
    print l

    n=0

    for i in range(len(l)):
        n=n*62
        n=n+l[i]

    return n
def len_long(midtemp):
    len=0
    while (midtemp>0):
        midtemp=midtemp/10
        len=len+1
    return len
def eighteen_to_list(midtemp):
    temp=1
    n=[]
    len=len_long(midtemp)
    while(midtemp>0):
        temp=midtemp%18
        n.append(temp)
        midtemp=midtemp/18
    return n

def list_to_str(arg):
    strn=""
    for i in range(len(arg)):
        if(arg[i]<10 and arg[i]>=0):
            strn=strn+str(arg[i])
        if(arg[i]>=9 and arg[i]<36):
            #ABCD
            n=arg[i]-9
            strn=strn+aplb[n-1]
        if(arg[i]>=36 and arg[i]<62):
            #abcd
            n=arg[i]-36
            strn=strn+apls[n-1]
    return strn
etl= eighteen_to_list(mid)
```

```
strshow=list_to_str(etl)
```

```
7102eMkcarCeuXnaK
48HB11E144HG147C054EHADE
3344762519443970968475431898 96

EDAHE450C741GH441E11BH84
```

还原算法如上，倒是不算太复杂了

```
  init_key(magicbox);
  strcpy(&v9, "KanXueCrackMe2017");
  v10 = 0;
  keygen((SomeArr *)magicbox, &v9);
  // magicbox
  // 001E18C0  00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00  ............
  // 001E18D0  04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00  ............
  // 001E18E0  08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00  ............
  v2 = (_WORD *)v1[38];
  v3 = *((_DWORD *)v2 - 3);
  for ( i = 0; i < v3; ++v2 )
  {
    if ( i < 0 || i > v3 )
      sub_401540(0x80070057);
    input[i++] = *v2;
  }
  input[i] = 0;
  ms_exc.registration.TryLevel = 1;
  keygen((SomeArr *)magicbox, input);
  v5 = 0;
  do
  {
    if ( magicbox[v5] != v8[v5] )
```

看了一些算法分析的writeup,说是这里是建模建立了个魔方的矩阵。

其实这里如果没有提示，鬼知道什么魔方变换。矩阵操作倒是能看到。
keygen((SomeArr *)magicbox, input);

```
 // [[4, 3], [4, 2], [3, 2], [5, 3], [4, 3], [1, 2],
 // [1, 3], [0, 1], [4, 1],[2, 2], [1, 2], [0, 2],
// [5, 2], [5, 3], [1, 2], [1, 2], [0, 2], [4, 3],
 // [0, 2], [0, 2], [3, 3], [5, 3], [2, 3], [1, 2]]
      mult_arrays_pow(v3 / 3, v3 % 3 + 1, (int)computed);
```

这里一个大问题是很难分析出矩阵操作的规律，不过既然要对比 if (magicbox[v5] != v8[v5])的值让他相等，其实这里关键就是要看v8变化的过程。

```
00161C90  00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00  ............
00161CA0  04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00  ............
00161CB0  08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00  ............
00161CC0  0C 00 00 00 0D 00 00 00 0E 00 00 00 0F 00 00 00  ............
00161CD0  10 00 00 00 11 00 00 00 12 00 00 00 13 00 00 00  ............
00161CE0  14 00 00 00 15 00 00 00 16 00 00 00 17 00 00 00  ............
00161CF0  18 00 00 00 19 00 00 00 1A 00 00 00 1B 00 00 00  ............
00161D00  1C 00 00 00 1D 00 00 00 1E 00 00 00 1F 00 00 00  ............
00161D10  20 00 00 00 21 00 00 00 22 00 00 00 23 00 00 00   ...!..."...#...
```

```
00161D20  24 00 00 00 25 00 00 00 26 00 00 00 27 00 00 00  $...%...&...'...
00161D30  28 00 00 00 29 00 00 00 2A 00 00 00 2B 00 00 00  (...)...*...+...
00161D40  2C 00 00 00 2D 00 00 00 2E 00 00 00 2F 00 00 00  ,...-......./...
00161D50  00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00  ............
00161D60  04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00  ............
00161D70  08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00  ............
00161D80  0C 00 00 00 0D 00 00 00 0E 00 00 00 0F 00 00 00  ............
00161D90  10 00 00 00 11 00 00 00 12 00 00 00 13 00 00 00  ............
00161DA0  14 00 00 00 15 00 00 00 16 00 00 00 17 00 00 00  ............
00161DB0  18 00 00 00 19 00 00 00 1A 00 00 00 1B 00 00 00  ............
00161DC0  1C 00 00 00 1D 00 00 00 1E 00 00 00 1F 00 00 00  ............
00161DD0  20 00 00 00 21 00 00 00 22 00 00 00 23 00 00 00   ...!..."...#...
00161DE0  24 00 00 00 25 00 00 00 26 00 00 00 27 00 00 00  $...%...&...'...
00161DF0  28 00 00 00 29 00 00 00 2A 00 00 00 2B 00 00 00  (...)...*...+...
00161E00  2C 00 00 00 2D 00 00 00 2E 00 00 00 2F 00 00 00  ,...-......./...


00161C90  00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00  ............
00161CA0  04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00  ............
00161CB0  08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00  ............
00161CC0  0C 00 00 00 0D 00 00 00 0E 00 00 00 0F 00 00 00  ............
00161CD0  10 00 00 00 11 00 00 00 12 00 00 00 13 00 00 00  ............
00161CE0  14 00 00 00 15 00 00 00 16 00 00 00 17 00 00 00  ............
00161CF0  18 00 00 00 19 00 00 00 1A 00 00 00 1B 00 00 00  ............
00161D00  1C 00 00 00 1D 00 00 00 1E 00 00 00 1F 00 00 00  ............
00161D10  20 00 00 00 21 00 00 00 22 00 00 00 23 00 00 00   ...!..."...#...
00161D20  24 00 00 00 25 00 00 00 26 00 00 00 27 00 00 00  $...%...&...'...
00161D30  28 00 00 00 29 00 00 00 2A 00 00 00 2B 00 00 00  (...)...*...+...
00161D40  2C 00 00 00 2D 00 00 00 2E 00 00 00 2F 00 00 00  ,...-......./...
00161D50  02 00 00 00 07 00 00 00 12 00 00 00 2F 00 00 00  ........./...
00161D60  16 00 00 00 1F 00 00 00 2E 00 00 00 29 00 00 00  .........)...
00161D70  1E 00 00 00 0F 00 00 00 00 00 00 00 0D 00 00 00  ............
00161D80  0A 00 00 00 23 00 00 00 10 00 00 00 11 00 00 00  ....#........
00161D90  2C 00 00 00 01 00 00 00 0E 00 00 00 05 00 00 00  ,............
00161DA0  24 00 00 00 09 00 00 00 1A 00 00 00 27 00 00 00  $.........'...
00161DB0  08 00 00 00 25 00 00 00 04 00 00 00 2D 00 00 00  ...%......-...
00161DC0  18 00 00 00 13 00 00 00 1C 00 00 00 17 00 00 00  ............
00161DD0  28 00 00 00 0B 00 00 00 0C 00 00 00 1D 00 00 00  (............
00161DE0  20 00 00 00 03 00 00 00 2A 00 00 00 1B 00 00 00   ......*......
00161DF0  14 00 00 00 2B 00 00 00 06 00 00 00 19 00 00 00  ...+.........
00161E00  22 00 00 00 21 00 00 00 26 00 00 00 15 00 00 00  "...!...&......
```

这里有两个大块，初始化时是一样的，经过按字符串的一系列变化变成了下面的大块。
通过对寄存器更改观察 mult_arrays_pow(v3 / 3, v3 % 3 + 1, (int)computed);的操作堆这两字符串的更改

经过对【0，1】【0，2】操作数的分析，操作的是如下地址处的数据

```
00161D50  00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00  ............
00161D60  04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00  ............

06 00 00 00 07 00 00 00 00 00 00 00 01 00 00 00  ............
02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00

00161D50  02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00  ............
00161D60  06 00 00 00 07 00 00 00 00 00 00 00 01 00 00 00  ............
```

前一个数字代表矩阵代号，后一个数字代表循环移动的长度，1代表移动8.

也就是是说这个大块中有两个数组，第一个数组明显是用来对比变换的，不用分析，第二个数组是重点。分成了6个矩阵。【x,y】堆这几个矩阵进行操作。x代表矩阵编号，y代表矩阵变换。其实变换本身比较简单，就是简单的位移。1代表循环位移8位。

这么说倒是有点像魔方了。不过我感觉魔方变换本身不是难点，倒是这个逆向分析过程才是重点。

上面已经分析过了，原始的魔方数组经过的变换为 [[4, 3], [4, 2], [3, 2], [5, 3], [4, 3], [1, 2], [1, 3], [0, 1], [4, 1],[2, 2], [1, 2], [0, 2], [5, 2], [5, 3], [1, 2], [1, 2], [0, 2], [4, 3],[0, 2], [0, 2], [3, 3], [5, 3], [2, 3], [1, 2]]
要想使魔方还原就要经过逆运算。据说这里有不只一个解。
我们要还原还是要从最后的数据还原

```
#[[1, 2], [2, 1], [5, 1], [3, 1], [0, 2], [0, 2], [4, 1], [0, 2], [1, 2],
#[1, 2], [5, 1], [5, 2], [0, 2], [1, 2], [2, 2], [4, 3], [0, 3], [1, 1],
#[1, 2], [4, 1], [5, 1], [3, 2], [4, 2], [4, 1]]
```

化简下

```
[[1, 2], [2, 1], [5, 1], [3, 1],[4, 1], [0, 2], [5, 3], [0, 2], [1, 2], [2, 2], [4, 3], [0, 3],[1, 3],
```

3x+y-1

```
ln=[[1, 2], [2, 1], [5, 1], [3, 1],[4, 1], [0, 2], [5, 3], [0, 2], [1, 2], [2, 2], [4, 3], [0, 3],[1, 3
res=[]
for i in ln:
    res.append(3*i[0]+i[1]-1)
print res
```

[4, 6, 15, 9, 12, 1, 17, 1, 4, 7, 14, 2, 5, 12, 15, 10, 14]