2021年10月广东强网杯，REVERSE的simplere

原创

沐一 · 林 于 2021-10-14 09:36:28 发布 144 收藏 2

分类专栏： CTF 逆向 文章标签： ctf

本文链接：https://blog.csdn.net/xiao__1bai/article/details/120756649

CTF 同时被 2 个专栏收录

167 篇文章 6 订阅

订阅专栏

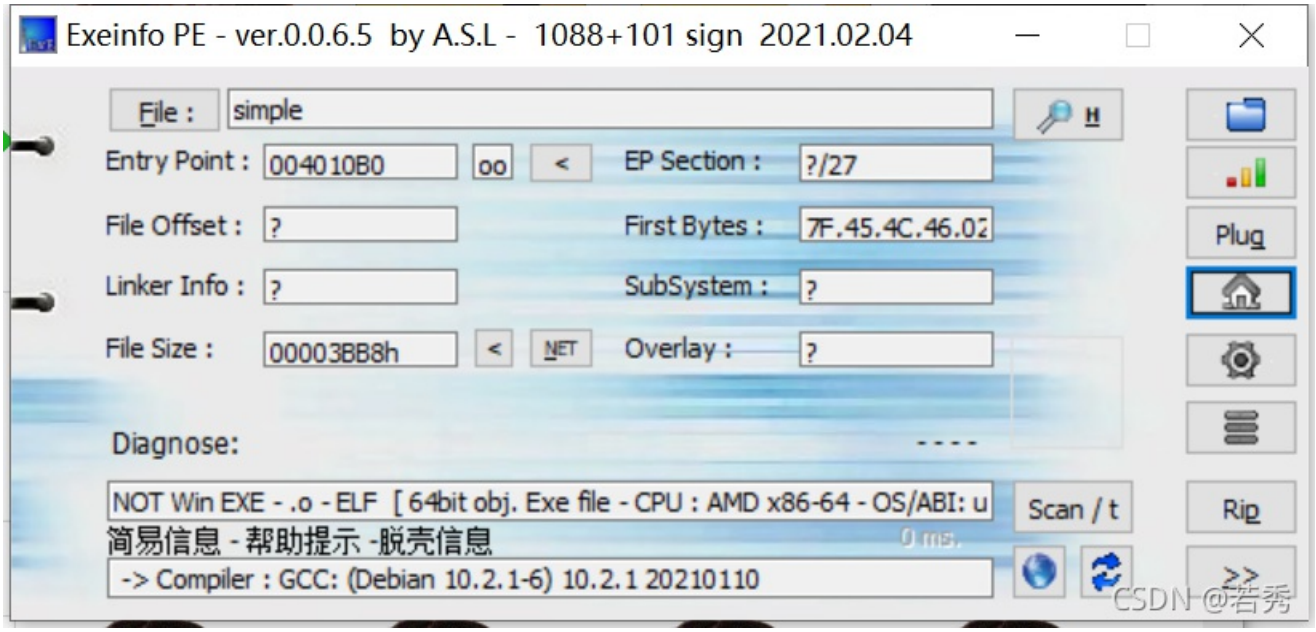
逆向

95 篇文章 6 订阅

订阅专栏

2021年10月广东强网杯，REVERSE的simplere

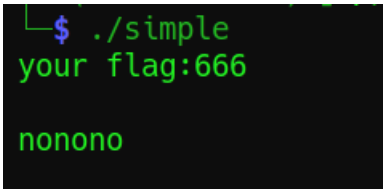
下载附件，照例扔入 `exeinfope` 中查看信息：



.

.

`64` 位 `ELF` 文件，无壳，照例先运行一下程序，查看主要回显信息：



照例扔入IDA中查看伪代码信息，有main函数看main函数：

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 result; // rax
  char v4[33]; // [rsp+0h] [rbp-250h] BYREF
  char v5[15]; // [rsp+21h] [rbp-22Fh] BYREF
  char v6[256]; // [rsp+30h] [rbp-220h] BYREF
  char v7[256]; // [rsp+130h] [rbp-120h] BYREF
  void *src; // [rsp+230h] [rbp-20h]
  void *dest; // [rsp+238h] [rbp-18h]
  int v10; // [rsp+244h] [rbp-Ch]
  int j; // [rsp+248h] [rbp-8h]
  int i; // [rsp+24Ch] [rbp-4h]

  printf("your flag:");
  __isoc99_scanf("%s", v6);
  putchar(10);
  if ( v6[15] == '-' )
  {
    for ( i = 0; i <= 14; ++i )
      v5[i] = v6[i];
    for ( j = 0; j <= 17; ++j )
      v4[j] = v6[j + 16];
    v10 = sub_401192(v5);
    if ( v10 == 1 )
    {
      dest = mmap(0LL, 0x1000uLL, 7, 33, -1, 0LL);
      src = (void *)sub_40126F();
      memcpy(dest, src, 0x28CuLL);
      ((void (__fastcall *)(char *, char *, void *))dest)(v7, v4, &unk_404320);
      v10 = sub_401342(v7);
      if ( v10 )
        puts("congratulations!");
      else
        printf("nonono");
      result = 0LL;
    }
    else
    {
      printf("nonono");
      result = 0LL;
    }
  }
  else
  {
```

接受输入

v5确定前15个字符，第16个字符是-，v4确定后18个字符。

一个关键自定义函数

（这里积累第一个经验）

上图分析了前半部分，现在跟踪那个关键自定义函数`v10 = sub_401192(v5);`这是对前`15`个字符组成的迷宫进行操作。

点进去查看逻辑，是一个走迷宫，输入15步的wasd，要从A开始走，必须走到.上，最后必须走到B上。

根据`11`可以判断是二维数组，这也是我第一次学到迷宫的`维数判断`。因为一个走`11`步，另一个一步一步走，所以`一维`字符串最后得出`二维迷宫`，长11宽5的`5*11`型，`s`和`w`字符一下走`11`步就是`上下移动`，`a`和`d`一下走1步就是`左右移动`。也就是说走得多的就是行移动，走得少的就是列移动。

A********
.*...
...*.
*******...
******B

手扒得到 ssdddwdddssas

```
__int64 __fastcall sub_401192(__int64 a1)
{
  int v1; // eax
  int i; // [rsp+Ch] [rbp-Ch]
  int v4; // [rsp+10h] [rbp-8h]
  int v5; // [rsp+14h] [rbp-4h]

  v5 = 1;
  v4 = 0;
  for ( i = 0; i <= 14; ++i )
  {
    v1 = *(unsigned __int8 *)(i + a1);
    if ( v1 == 'w' )
    {
      --v4;
    }
    else if ( *(unsigned __int8 *)(i + a1) <= (unsigned int)'w' )
    {
      if ( v1 == 's' )
      {
        ++v4;
      }
      else if ( *(unsigned __int8 *)(i + a1) <= (unsigned int)'s' )
      {
        if ( v1 == 'a' )
        {
          --v5;
        }
        else if ( v1 == 'd' )
        {
          ++v5;
        }
      }
    }
  if ( aAB[11 * v4 + v5] == 'B' )
    return 1LL;
  if ( aAB[11 * v4 + v5] != '.' )
    return 0LL;
  }
  return 0LL;
}
```

前面是对前15个字符的逐个字符比较

后面是对应的字符要满足条件才行，这里我是直接手扒的因为我比较菜，不会用算法来算。

// *A**********.***.....**.....***.*********..*********B**

.

.

然后上半部分条件就过了，开始分析后半部分：

```
  int i; // [rsp+24Ch] [rbp-4h]

  printf("your flag:");
  __isoc99_scanf("%s", v6);
  putchar(10);
  if ( v6[15] == '-' )
  {
    for ( i = 0; i <= 14; ++i )
      v5[i] = v6[i];
    for ( j = 0; j <= 17; ++j )
      v4[j] = v6[j + 16];
    v10 = sub_401192(v5);
    if ( v10 == 1 )
    {
      dest = mmap(0LL, 0x1000uLL, 7, 33, -1, 0LL);
      src = (void *)sub_40126F();
      memcpy(dest, src, 0x28CuLL);
```

dest开辟空间，src关键自定义函数运算，然后复制给dest

```
29    ((void (__fastcall *)(char *, char *, void *))dest)(v7, v4, &unk_404320);
30    v10 = sub_401342(v7);
31    if ( v10 )
32      puts("congratulations!");
33    else
34      printf("nonono");
35    result = 0LL;
36  }
37  else
38  {
39    printf("nonono");
40    result = 0LL;
41  }
42 }
43 else
44 {
45   printf("nonono");
46   result = 0LL;
47 }
48 return result;
49}
```

又一个自定义函数

明显的自修改代码，可以动态调试，可以静态修补，后面的都是传入参数。

.

.

先跟踪分析 `src = (void *)sub_40126F();` 函数：

```
1 _BYTE *sub_40126F()
2 {
3   _BYTE *v1; // [rsp+8h] [rbp-18h]
4   int j; // [rsp+14h] [rbp-Ch]
5   int i; // [rsp+18h] [rbp-8h]
6   int v4; // [rsp+1Ch] [rbp-4h]
7
8   v1 = malloc(0x200uLL);
9   v4 = 0;
0   for ( i = 0; i < 652; ++i )
1   {
2     if ( v4 == 64 )
3       v4 = 0;
4     v1[i] = byte_404080[i] ^ byte_404360[v4++];
5   }
6   for ( j = 0; j <= 63; ++j )
7     byte_404320[j] ^= byte_404360[j];
8   return v1;
9 }
```

两个原始数组异或，赋值给v1，v1再赋值给dest，这里的异或应该构成了dest的自修改函数了。

这里对byte_404320重新构造了，所以后面作为dest传入参数是修改过的。

.

.

（这里积累第二个经验）

这里的 `dest` 是在内存中的，这里静态修补我不太会，因为 `dest` 和 `src` 都是在栈中的所以直接用动态调试，用前面得出的前半部分搭配其它字符来运行 `ssddddwddddssas-666666666666666666` ：

```
14  printf("your flag:");
15  __isoc99_scanf("%s", v6);
16  putchar(10);
17  if ( v6[15] == '-' )
18  {
19    for ( i = 0; i <= 14; ++i )
20      v5[i] = v6[i];
21    for ( j = 0; j <= 17; ++j )
22      v4[j] = v6[j + 16];
23    v10 = sub_401192(v5);
24    if ( v10 == 1 )
25    {
26      dest = mmap(0LL, 0x1000uLL, 7, 33, -1, 0LL);
27      src = (void *)sub_40126F();
28      memcpy(dest, src, 0x28CuLL);
29      ((void (__fastcall *)(char *, char *, void *))dest)(v7, v4, &byte_404320);
30      v10 = sub_401342(v7);
31      if ( v10 )
```

单步F7走起


```
zero:00007F5492908000
zero:00007F5492908000 ; Attributes: bp-based frame
zero:00007F5492908000
zero:00007F5492908000 sub_7F5492908000 proc near
zero:00007F5492908000
zero:00007F5492908000 var_38= qword ptr -38h
zero:00007F5492908000 var_30= qword ptr -30h
zero:00007F5492908000 var_28= qword ptr -28h
zero:00007F5492908000 var_18= dword ptr -18h
zero:00007F5492908000 var_14= dword ptr -14h
zero:00007F5492908000 var_10= qword ptr -10h
zero:00007F5492908000 var_8= qword ptr -8
zero:00007F5492908000
zero:00007F5492908000 push    rbp
zero:00007F5492908001 mov     rbp, rsp
zero:00007F5492908004 mov     [rbp+var_28], rdi
zero:00007F5492908008 mov     [rbp+var_30], rsi
zero:00007F549290800C mov     [rbp+var_38], rdx
```

按P即可反汇编生成函数


```
1  __int64 __fastcall sub_7F5492908000(__int64 a1, __int64 a2, __int64 a3)
2  {
3    __int64 result; // rax
4    int v4; // [rsp+20h] [rbp-18h]
5    int v5; // [rsp+24h] [rbp-14h]
6    __int64 v6; // [rsp+28h] [rbp-10h]
7    __int64 v7; // [rsp+30h] [rbp-8h]
8
9    while ( *(_BYTE *)(v6 + a2) )
10     ++v6;
11   if ( v6 % 3 )
12     v7 = 4 * (v6 / 3 + 1);
13   else
14     v7 = 4 * (v6 / 3);
15   *(_BYTE *)(v7 + a1) = 0;
16   v5 = 0;
17   v4 = 0;
18   while ( v5 < v7 - 2 )
19   {
20     *(_BYTE *)(v5 + a1) = *(_BYTE *)((*(_BYTE *)(v4 + a2) >> 2) + a3);
21     *(_BYTE *)(v5 + 1LL + a1) = *(_BYTE *)((((16 * *(_BYTE *)(v4 + a2)) & 0x30 | (unsigned int)(*(_BYTE *)(v4 + 1LL + a2) >> 4))
22                                          + a3);
23     *(_BYTE *)(v5 + 2LL + a1) = *(_BYTE *)(((4 * *(_BYTE *)(v4 + 1LL + a2)) & 0x3C | (unsigned int)(*(_BYTE *)(v4 + 2LL + a2) >> 6))
24                                          + a3);
25     *(_BYTE *)(v5 + 3LL + a1) = *(_BYTE *)((*(_BYTE *)(v4 + 2LL + a2) & 0x3F) + a3);
26     v4 += 3;
27     v5 += 4;
28   }
29   result = v6 % 3;
30   if ( v6 % 3 == 1 )
31   {
32     *(_BYTE *)(v5 - 2LL + a1) = '=';
33     result = v5 - 1LL + a1;
34     *(_BYTE *)result = '=';
35   }
36   else if ( result == 2 )
37   {
38     result = v5 - 1LL + a1;
39     *(_BYTE *)result = '=';
```

a3是变形加密表单，是前面src函数中被修改过的数组

典型的base64加密

```
40  }
41  return result;
42}
```


.

.

所以这题就是base64变码加密了，继续跟踪最后的自定义函数`v10 = sub_401342(v7);`：

```
1  __int64 __fastcall sub_401342(__int64 a1)
2{
3  int v2; // [rsp+1Ch] [rbp-14h]
4  int i; // [rsp+2Ch] [rbp-4h]
5
6  v2 = strlen("r60ihyZ/m4lseHt+m4t+mIkc");
7  for ( i = 0; i < v2; ++i )
8  {
9    if ( *(_BYTE *)(i + a1) != aR60ihyzM4lseht[i] )
10     return 0LL;
11 }
12 return 1LL;
13}
```

变形base64加密密文

.

.

密文也有了，可以写逻辑了，首先求出base64变形码表，从
`src = (void *)sub_40126F();`处导出数组：

```
key2=[  0x4D, 0x20, 0x07, 0x05, 0x43, 0x15, 0x7A, 0x73, 0x39, 0x01,
  0x7F, 0x53, 0x66, 0x4E, 0x0D, 0x18, 0x60, 0x76, 0x75, 0x00,
  0x58, 0x15, 0x00, 0x32, 0x68, 0x3F, 0x78, 0x7F, 0x7B, 0x64,
  0x4E, 0x49, 0x0F, 0x2E, 0x3F, 0x0D, 0x0D, 0x0D, 0x64, 0x66,
  0x61, 0x53, 0x06, 0x44, 0x34, 0x6E, 0x69, 0x2F, 0x20, 0x14,
  0x37, 0x6A, 0x49, 0x55, 0x36, 0x37, 0x23, 0x23, 0x2A, 0x6B,
  0x73, 0x06, 0x78, 0x0B]

key3=[  0x3E, 0x7A, 0x40, 0x64, 0x27, 0x25, 0x48, 0x04, 0x4F, 0x63,
  0x19, 0x60, 0x0B, 0x3A, 0x75, 0x5D, 0x11, 0x4E, 0x07, 0x44,
  0x30, 0x4C, 0x4B, 0x06, 0x5F, 0x73, 0x0D, 0x1A, 0x38, 0x08,
  0x34, 0x78, 0x45, 0x47, 0x58, 0x3B, 0x74, 0x7D, 0x2C, 0x2B,
  0x4A, 0x3C, 0x29, 0x13, 0x01, 0x3F, 0x03, 0x61, 0x70, 0x52,
  0x65, 0x09, 0x22, 0x00, 0x7F, 0x59, 0x6C, 0x77, 0x72, 0x3D,
  0x32, 0x55, 0x41, 0x49]


for i in range(len(key3)):
 key2[i]^=key3[i]
print(key2)
print(''.join(map(chr,key2)))  #base64变表码
```

结果：

```
└─$ python 4.py
[115, 90, 71, 97, 100, 48, 50, 119, 118, 98, 102, 51, 109, 116, 120, 69, 113, 56, 114, 68,
104, 89, 75, 52, 55, 76, 117, 101, 67, 108, 122, 49, 74, 105, 103, 54, 121, 112, 72, 77, 43
, 111, 47, 87, 53, 81, 106, 78, 80, 70, 82, 99, 107, 85, 73, 110, 79, 84, 88, 86, 65, 83, 5
7, 66]3,
sZGad02wvbf3mtxEq8rDhYK47LueClz1Jig6ypHM+o/W5QjNPFRckUInOTXVAS9B
```

.

.

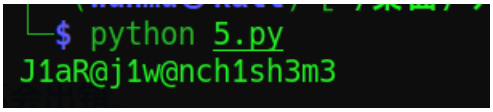
然后把变形码表替换传统`base64`加密码表，这里用的是我以前写过的base64 python编码实现：

```
base64="sZGad02wvbf3mtxEq8rDhYK47LueClz1Jig6ypHM+o/W5QjNPFRckUInOTXVAS9B"   #准备好base64的基表
def encryption(inputstring):  #定义加密函数
 ascii=['{:0>8}'.format(str(bin(ord(i))).replace('0b','')) for i in inputstring] #把每个输入字符保证8位一个，才能3*8变4*6。
#{:0>8}是右对齐8位然后左边补0,因为python是自己判断数据大小类型的，所以必须强制满足8位。bin转化二进制会带0b前缀，所以要用replace('0b','')去掉。
 encrystr='' #while外的变量，返回base64加密后的字符串
 equalnumber=0 #while外的变量，记录拆分后不足4的倍数时需要补齐的等号个数
 while ascii:
  subascii=ascii[:3] #用一个子列表subascii每次取输入的三位进行操作,前面操作后每位都是8位
  while len(subascii)<3:   #这里其实是最后一段截取中才会用上的，不满足3位时要单独取出，记录equalnumber数量用于后期补'='号，然后补齐8位的0免得干扰后面3*8拆分成4*6
   equalnumber+=1  #计算要补‘=’的个数
   subascii+=['0'*8] #补8个0来填充够3的倍数，这然后面就不会出错。
  substring=''.join(subascii)#用substring合并subascii的3个8位，准备进行拆分操作
  encrystringlist=[substring[x:x+6] for x in [0,6,12,18]]   #开始进行3*8变4*6的拆分，每次拆分一组24位。
  encrystringlist=[int(x,2) for x in encrystringlist]  #把前面拆分的6位一组转成10进制，就不用进行位数补齐操作了，这是用来后面对应base64基表的下标。
  if equalnumber:
   encrystringlist=encrystringlist[0:4-equalnumber] #如果前面不足3字符补了0,比如2个8位字符16位，拆分后就要用3个6位共18位，所以有效位是4-equalnumber
  encrystr+=''.join(base64[x] for x in encrystringlist) #这里encrystringlist已经在前面拆分成4*6且转换成10进制了，所以对应基表的下标。
  ascii=ascii[3:]  #每次向后取3个列表元素，对应while循环条件
 encrystr+='='*equalnumber  #因为前面encrystringlist[0:4-equalnumber]去掉了补0位，所以这里最后补齐'='号
 return encrystr

def decryption(inputstring):
 ascii=['{0:0>6}'.format(str(bin(base64.index(i))).replace('0b',''))for i in inputstring if i!='=']#从加密字符中取除补位'='之外加密字符，即6位生成的base64基表下标的数，按6位一组排列，准备拆分
 decrystr=''#准备while外的解密后的字符
 equalnumber=inputstring.count('=')#这里计数补位的'='号的个数，后面不够8位时会根据'='号补加位数。
 while ascii:
  subascii=ascii[:4]#取加密字符的4个6位一组共24位准备拆分合并成3*8
  substring=''.join(subascii)#先连成一串24位
  if len(substring)%8!=0:
   substring=substring[0:-1*equalnumber*2]
#截取到倒数第equalnumber*2个元素。对不足8位的组补位，因为加密时1个8位要来2个6位，两个'='号，截取到8位就是倒数第4位1*2*2。2个8位要3个6位，要一个'='号，截取到16位就是倒数第2位1*1*2。
  decrystringlist=[substring[x:x+8] for x in [0,8,16]]#开始进行4*6变3*8的拆分，每次拆分4个6位一组24位。
  decrystringlist=[int(x,2) for x in decrystringlist if x]#把前面拆分的8位一组转成10进制，用来对应十进制ASCII码，if x功能不清楚，但不可缺少，应该是要排除空格吧。
  decrystr+=''.join([chr(x) for x in decrystringlist])#这里decrystringlist已经在前面拆分成3*8且转换成10进制了，现在转换成ASCII码。
  ascii=ascii[4:]#每次向后取4个列表元素，对应while循环条件
 return decrystr

if __name__=="__main__":
 #print(encryption('abcd'))
 print(decryption('r60ihyZ/m4lseHt+m4t+mIkc'))
```

.
结果：



.
.
所以最终 `flag`：

> flag{ssddddwddddssas-J1aR@j1w@nch1sh3m3}

.
.
总结：

> 1：
> （这里积累第一个经验）
> 上图分析了前半部分，现在跟踪那个关键自定义函数 `v10 = sub_401192(v5);` 这是对前 `15` 个字符组成的迷宫进行操作。
> .
> 点进去查看逻辑，是一个走迷宫，输入15步的wasd，要从A开始走，必须走到.上，最后必须走到B上。
> .
> 根据 `11` 可以判断是二维数组，这也是我第一次学到迷宫的 `维数判断` 。因为一个走 `11` 步，另一个一步一步走，所以 `一维` 字符串最后得出 `二维迷宫` ，长11宽5的 `5*11` 型， `s` 和 `w` 字符一下走 `11` 步就是 `上下移动` ， `a` 和 `d` 一下走1步就是 `左右移动` 。也就是说走得多的就是行移动，走得少的就是列移动。
> .
> *A********
> .*...
> ...*.
> *******...
> ******B
> .
> 手扒得到 `ssddddwddddssas`

```
__int64 __fastcall sub_401192(__int64 a1)
{
  int v1; // eax
  int i; // [rsp+Ch] [rbp-Ch]
  int v4; // [rsp+10h] [rbp-8h]
  int v5; // [rsp+14h] [rbp-4h]

  v5 = 1;
  v4 = 0;
  for ( i = 0; i <= 14; ++i )
  {
    v1 = *(unsigned __int8 *)(i + a1);
    if ( v1 == 'w' )
    {
      --v4;
    }
    else if ( *(unsigned __int8 *)(i + a1) <= (unsigned int)'w' )
    {
      if ( v1 == 's' )
      {
        ++v4;
      }
      else if ( *(unsigned __int8 *)(i + a1) <= (unsigned int)'s' )
      {
        if ( v1 == 'a' )
        {
          --v5;
        }
        else if ( v1 == 'd' )
        {
          ++v5;
        }
```

前面是对前15个字符的逐个字符比较

后面是对应的字符要满足条件才行，这里我是直接手扒的

```
    }
  }
  if ( aAB[11 * v4 + v5] == 'B' )
    return 1LL;
  if ( aAB[11 * v4 + v5] != '.' )
    return 0LL;
}
return 0LL;
}
```

后面是对应的字符要满足条件才行，这里我是直接手扒的因为我比较菜，不会用算法来算。

// *A**********.***.....**.....***.*********..*********B**


```
int i; // [rsp+24Ch] [rbp-4h]

printf("your flag:");
__isoc99_scanf("%s", v6);
putchar(10);
if ( v6[15] == '-' )
{
  for ( i = 0; i <= 14; ++i )
    v5[i] = v6[i];
  for ( j = 0; j <= 17; ++j )
    v4[j] = v6[j + 16];
  v10 = sub_401192(v5);
  if ( v10 == 1 )
  {
    dest = mmap(0LL, 0x1000uLL, 7, 33, -1, 0LL);
    src = (void *)sub_40126F();
    memcpy(dest, src, 0x28CuLL);
    ((void (__fastcall *)(char *, char *, void *))dest)(v7, v4, &unk_404320);
    v10 = sub_401342(v7);
    if ( v10 )
      puts("congratulations!");
    else
      printf("nonono");
    result = 0LL;
  }
  else
  {
    printf("nonono");
    result = 0LL;
  }
}
else
{
  printf("nonono");
  result = 0LL;
}
return result;
}
```

dest开辟空间，src关键自定义函数运算，然后复制给dest

又一个自定义函数

明显的自修改代码，可以动态调试，可以静态修补，后面的都是传入参数。

2：

（这里积累第二个经验）

这里的 `dest` 是在内存中的，这里静态修补我不太会，因为 `dest` 和 `src` 都是在栈中的所以直接用动态调试，用前面得出的前半部分搭配其它字符来运行 `ssdddddwdddddssas-666666666666666666` ：

```
__int64 __fastcall sub_7F5492908000(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 result; // rax
  int v4; // [rsp+20h] [rbp-18h]
  int v5; // [rsp+24h] [rbp-14h]
  __int64 v6; // [rsp+28h] [rbp-10h]
  __int64 v7; // [rsp+30h] [rbp-8h]

  while ( *(_BYTE *)(v6 + a2) )
    ++v6;
  if ( v6 % 3 )
    v7 = 4 * (v6 / 3 + 1);
  else
    v7 = 4 * (v6 / 3);
  *(_BYTE *)(v7 + a1) = 0;
  v5 = 0;
  v4 = 0;
  while ( v5 < v7 - 2 )
  {
    *(_BYTE *)(v5 + a1) = *(_BYTE *)((*(_BYTE *)(v4 + a2) >> 2) + a3);
    *(_BYTE *)(v5 + 1LL + a1) = *(_BYTE *)(((16 * *(_BYTE *)(v4 + a2)) & 0x30 | (unsigned int)(*(_BYTE *)(v4 + 1LL + a2) >> 4))
                                         + a3);
    *(_BYTE *)(v5 + 2LL + a1) = *(_BYTE *)(((4 * *(_BYTE *)(v4 + 1LL + a2)) & 0x3C | (unsigned int)(*(_BYTE *)(v4 + 2LL + a2) >> 6))
                                         + a3);
    *(_BYTE *)(v5 + 3LL + a1) = *(_BYTE *)((*(_BYTE *)(v4 + 2LL + a2) & 0x3F) + a3);
    v4 += 3;
    v5 += 4;
  }
  result = v6 % 3;
  if ( v6 % 3 == 1 )
  {
    *(_BYTE *)(v5 - 2LL + a1) = '=';
    result = v5 - 1LL + a1;
    *(_BYTE *)result = '=';
  }
  else if ( result == 2 )
  {
    result = v5 - 1LL + a1;
    *(_BYTE *)result = '=';
  }
  return result;
}
```

a3是变形加密表单，是前面src函数中被修改过的数组

典型的base64加密

解毕！敬礼！