[2018看雪] - 第四题 - 对抗反汇编+密码

原创

Flying_Fatty 于 2018-07-20 20:09:39 发布 716 收藏 1
分类专栏： reverse

本文链接：https://blog.csdn.net/kevin66654/article/details/81135577

逆向断章取义

```
.data:004… 0000003E     C     Base must be binary (MR_ALWAYS_BINARY defined in mirdef.h ?)\n
.data:004… 0000002B     C     Unable to control Floating-point rounding\n
.data:004… 00000024     C     Specified basis is NOT irreducible\n
.data:004… 00000032     C     Specified double length type isn't double length\n
.data:004… 00000020     C     Number Base must be power of 2\n
.data:004… 00000012     C     Exponent too big\n
.data:004… 00000014     C     No modulus defined\n
.data:004… 00000012     C     Illegal modulus \n
.data:004… 0000002E     C     MIRACL not initialised - no call to mirsys()\n
.data:004… 00000015     C     I/O buffer overflow\n
.data:004… 00000024     C     Flash to double conversion failure\n

.data:004… 0000001C     C     \nMIRACL error from routine
```

MIRACL大数库了解一下

在OD中调试下断的时候，看到了这个字符串

```
serial:GetLastError()=87
some problem!0.0123456789abcdefghijklmn
```

```
.rdata:0048606C aSomeProblem00  db 'some problem!0.0',0 ; DATA XREF: sub_402070:loc_402163↑o
.rdata:0048607D                 align 10h
.rdata:00486080 aExplorerExe    db 'explorer.exe',0     ; DATA XREF: sub_402070+8F↑o
.rdata:0048608D                 align 10h
```

```
17  if ( (unsigned __int8)sub_4010FA((int)&v5, "explorer.exe") )
18  {
19    v3 = 0;
20    v6 = -1;
21    sub_401249((int)&v5);
22    result = v3;
23  }
24  else if ( (unsigned __int8)sub_401262(&v5, &unk_495640) )
25  {
26    if ( !sub_401271(dword_495650) )
27      printf("some problem!0.0");
28    v2 = 1;
29    v6 = -1;
30    sub_401249((int)&v5);
31    result = v2;
32  }
33  else
34  {
35    v1 = 1;
36    v6 = -1;
37    sub_401249((int)&v5);
38    result = v1;
39  }
```

有个反调试原理：查看当前进程的父进程是否为explorer.exe

双击运行和cmd运行的程序的父进程都是explorer.exe，在OD下调试的程序的父进程是OD，这可以作为一个反动态调试的原理根据，奇怪的是对OD没啥用，还是可以正常下断正常调试（除了输出的提示字符串不同，不影响正常功能）

https://bbs.pediy.com/thread-228792.htm

这里有个IDA-python脚本可以学习（用来patch的）

https://blog.csdn.net/kevin66654

根据对抗反汇编的思路，在main中修改IDA的反汇编（Data和Code的识别，把干扰分析的patch掉）

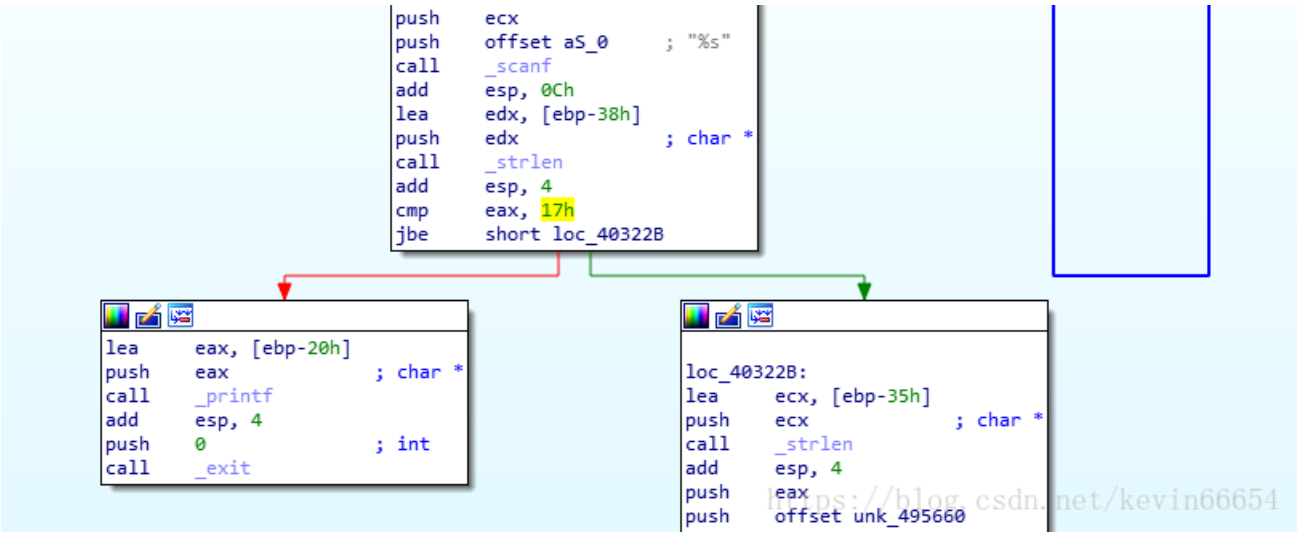
```
.text:0045C106 74 04                    jz      short loc_45C10C
.text:0045C108 75 02                    jnz     short loc_45C10C

.text:0045C10C EB 01                    jmp     short loc_45C10F
.text:0045C10C                  ; ---------------------------------------------------------------------------
.text:0045C10E 81               byte_45C10E     db 81h                  ; CODE
.text:0045C10F                  ; ---------------------------------------------------------------------------
.text:0045C10F
.text:0045C10F                  loc_45C10F:                             ; CODE
.text:0045C10F 55                       push    ebp
.text:0045C110 74 04                    jz      short loc_45C116
.text:0045C112 75 02                    jnz     short loc_45C116

.text:0045C120                                                          ;
.text:0045C120 83 EC 7C                 sub     esp, 7Ch
.text:0045C123 E9 0E 70 FA FF           jmp     loc_403136
.text:0045C123                  ; ---------------------------------------------------------------------------

.text:0040315B E8                       db 0E8h
.text:0040315C                  ; ---------------------------------------------------------------------------
.text:0040315C
.text:0040315C                  loc_40315C:                             ; CODE XREF: sub_403160↓p
.text:0040315C EB 07                    jmp     short loc_403165
.text:0040315C                  ; ---------------------------------------------------------------------------
.text:0040315E 00                       db    0
.text:0040315F 00                       db    0
.text:00403160
.text:00403160                  ; =============== S U B R O U T I N E =======================================
.text:00403160
.text:00403160                  sub_403160      proc near               ; CODE XREF: .text:00403156↑p
.text:00403160
.text:00403160                  arg_0           = dword ptr  4
.text:00403160
.text:00403160 E8 F7 FF FF FF           call    loc_40315C
.text:00403165
.text:00403165                  loc_403165:                             ; CODE XREF: .text:loc_40315C↑j
.text:00403165                                                          ; sub_403160+1A↓j
.text:00403165 83 C4 08                 add     esp, 8
.text:00403168 0F 31                    rdtsc
.text:0040316A 50                       push    eax
.text:0040316B 0F 31                    rdtsc
.text:0040316D 2B 04 24                 sub     eax, [esp-4+arg_0]
.text:00403170 83 C4 04                 add     esp, 4
.text:00403173 3D FF 0F 00 00           cmp     eax, 0FFFh
.text:00403178 76 02                    jbe     short loc_40317C
.text:0040317A EB E9                    jmp     short loc_403165
```

绕过了这些，可以看到之后开始有对于输入的判断了



flag长度为0x17 = 23

402220函数：把输入的字符转换为对应的hex形式，如 'a' = 0x61，即变成61

```
.text:00402AD7 68 A0 60 48 00                   push    offset a3e9 ; "3e9"
```

看到数据0x3e9 = 1001

分析402580函数：

```
.text:00402588 57                               push    edi
.text:00402589 8D 7D B8                         lea     edi, [ebp+var_48]
.text:0040258C B9 12 00 00 00                   mov     ecx, 12h
.text:00402591 B8 CC CC CC CC                   mov     eax, 0CCCCCCCCh
.text:00402596 F3 AB                            rep stosd
.text:00402598 E8 05 00 00 00                   call    sub_4025A2
.text:00402598                  sub_402580      endp ; sp-analysis failed
.text:00402598
.text:00402598                  ; ---------------------------------------------------------
.text:0040259D E8               byte_40259D     db 0E8h
.text:0040259E                  ; ---------------------------------------------------------
.text:0040259E
.text:0040259E                  loc_40259E:                             ; CODE XREF:
.text:0040259E EB 07                            jmp     short loc_4025A7
.text:0040259E                  ; ---------------------------------------------------------
.text:004025A0 EB                               db 0EBh
.text:004025A1 00                               db    0
.text:004025A2
.text:004025A2                  ; =============== S U B R O U T I N E ===============
.text:004025A2
.text:004025A2                  sub_4025A2      proc near               ; CODE XREF:
.text:004025A2 E8 F7 FF FF FF                   call    loc_40259E
.text:004025A7
.text:004025A7                  loc_4025A7:                             ; CODE XREF:
.text:004025A7 83 C4 08                         add     esp, 8
.text:004025AA C7 45 FC 01 00 00 00             mov     dword ptr [ebp-4], 1
.text:004025B1 C7 45 F8 00 00 00 00             mov     dword ptr [ebp-8], 0
.text:004025B8 EB 09                            jmp     short loc_4025C3
```

中间的jmp都nop掉，可以发现功能为：

```
size_t __cdecl sub_402580(char *a1)
{
  size_t result; // eax
  unsigned int i; // [esp+4Ch] [ebp-8h]
  signed int v3; // [esp+50h] [ebp-4h]

  v3 = 1;
  for ( i = 0; ; ++i )
  {
    result = strlen(a1);
    if ( i >= result )
      break;
    a1[i] ^= v3++;
  }
  return result;
}
```


一个简单的XOR，可以猜想是对数据进行处理

然后看到402630函数，做同样的nop操作之后，发现：

```
  memset(&v70, 0, 0x88u);
  sub_401078(&v79);
  sub_401078(&v6);
  *(_DWORD *)(v144 + 564) = 16;
  v5 = sub_409350(0);
  v4 = sub_409350(0);
  v2 = sub_409350(0);
  v3 = sub_409350(0);
  sub_40D1E0(v2, &unk_495660);
  sub_40D1E0(v5, &v6);
  sub_40D1E0(v4, "3e9");
  if ( sub_40A2C0(v2, v5) != -1 )
    return 0;
  sub_40C110(v2, v4, v5, v3);
  sub_40B280(0, v3, &v75, 0);
  sub_409CA0(v5);
  sub_409CA0(v4);
  sub_409CA0(v2);
  sub_409CA0(v3);
  sub_409CC0();
  v0 = strlen(&v75);
  sub_40100F(&v75, &v71, v0);
  return strcmp(&v79, &v71) == 0;
}
```

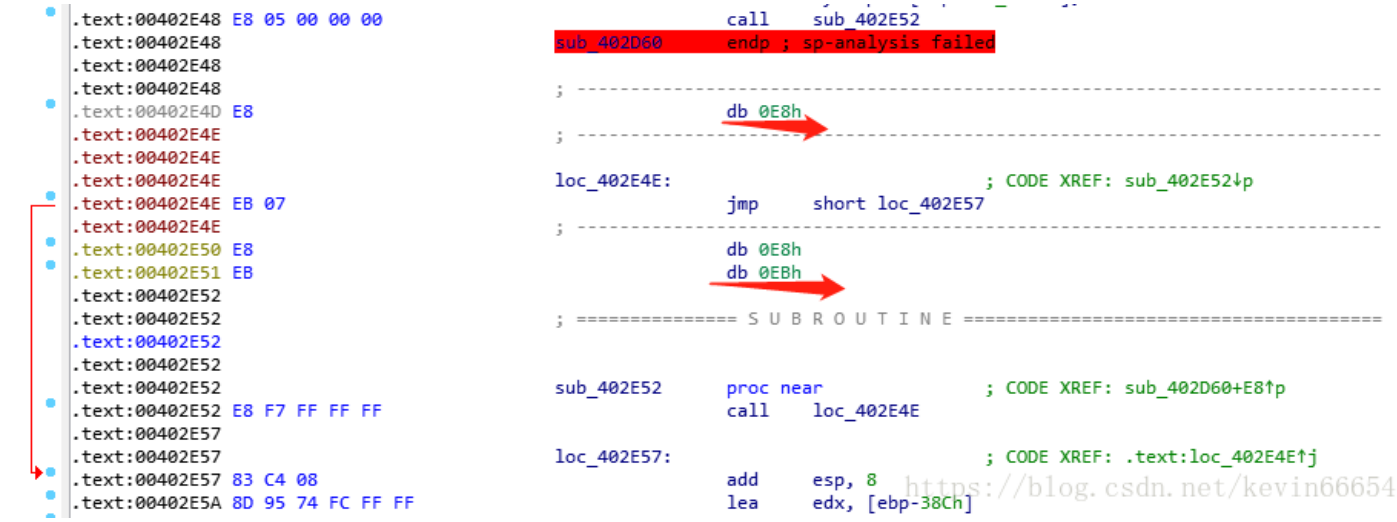

根据0x3e9，得知40D1E0函数为初始化函数

分析逻辑，v2，v4，v5在函数40C110的计算之下会变成c3，根据某种转化会变成v75，然后进行比对

根据算法分析猜想RSA，从初始化过程中提取数据

第二部分check在402D60函数中：IDA解析仍然出错

```
  BYTE2(v2) = 32;
  sub_402E52();
  MEMORY[0xEC28363D]();
  return sub_402E52();
}
```



在IDA里使用Data和Code的转换，可以明显看出逻辑，把中间的全部nop掉即可

```
52  memcpy(&v5, "123567389:;<=>?", 0xFu);
53  BYTE2(v7) = 32;
54  XOR(v1, (int)&v20);
55  XOR(v2, (int)&v5);
56  v5 = *a1;
57  LOBYTE(v6) = a1[1];
58  BYTE1(v6) = dword_495728 + a1[2];
59  memset(&v28, 0, 0x1FCu);
60  sub_40D760((int *)&v28, 0, 16, (int)&v5, 0);
61  sub_40DC40(&v28, &v12);
62  v3 = strlen(&v12);
63  sub_40100F(&v12, &v24, v3);
64  return strcmp(&v20, &v24) == 0;
```

在函数中翻到了如下的数据：

```
.rdata:0048637C byte_48637C     db 63h
.rdata:0048637C
.rdata:0048637D                 db  7Ch ; |
.rdata:0048637E                 db  77h ; w
.rdata:0048637F                 db  7Bh ; {
.rdata:00486380                 db 0F2h
.rdata:00486381                 db  6Bh ; k
.rdata:00486382                 db  6Fh ; o
.rdata:00486383                 db 0C5h
```

搜索一下为AES加密中的Sbox，初步判断为AES加密

```
52  memcpy(&key, "123567389:;<=>?", 0xFu);
53  BYTE2(v7) = 32;
54  XOR(v1, (int)&v20);
55  XOR(v2, (int)&key);
56  key = *a1;
57  LOBYTE(v6) = a1[1];
58  BYTE1(v6) = dword_495728 + a1[2];
59  memset(v28, 0, 0x1FCu);
60  sub_40D760(v28, 0, 16, (int)&key, 0);
61  sub_40DC40(v28, &Message);
62  v3 = strlen(&Message);
63  NumberToHex((int)&Message, (int)&Calc, v3);
64  return strcmp(&v20, &Calc) == 0;
```

```
char key; // [esp+4Ch] [ebp-530h]
int v6; // [esp+4Dh] [ebp-52Fh]
int v7; // [esp+59h] [ebp-523h]
char v8; // [esp+60h] [ebp-51Ch]
char v9; // [esp+61h] [ebp-51Bh]
__int16 v10; // [esp+125h] [ebp-457h]
char v11; // [esp+127h] [ebp-455h]
```

可见，key与我们输入flag的前三个字节有关

Message为明文pediy，也就是说，pediy经过某个密钥AES加密之后，等于一个值

因为就三个字符，第一想法是爆破：62 * 62 * 62 = 238328，也是可行方法

```
.text:00403267                call    DigitCheck
.text:0040326C                add     esp, 4
.text:0040326F                and     eax, 0FFh
.text:00403274                test    eax, eax
.text:00403276                jz      short loc_403289
.text:00403278                lea     eax, [ebp-3Ch]
.text:0040327B                push    eax
.text:0040327C                call    j_AESCheck
.text:00403281                add     esp, 4
.text:00403284                mov     [ebp-8], eax
.text:00403287                jmp     short loc_403299
```

在IDA中发现了AESCheck之前，对这三个字符进行了判断：必须都是数字

所以爆破总数为1000！

先单步调试确认一下数据

```
00402EBA  .  8D95 D0FAFFFF    lea edx, [local.332]
堆栈地址=0018F97C, (ASCII "12413140000000000")
edx=00000000
```

```
EDI 0018FA58 ASCII "pediy"
```

```
**********看雪-京东CTF2018**********
serial:GetLastError()=87
some problem!0,0123456789abcdefghijklmn
```

对于AES的加密部分，密钥为前三位数字相关，明文pediy

爆破的几种姿势：

```
正在比较文件 a.txt 和 B.TXT
***** a.txt
**********看雪-京东CTF2018**********
serial:error
***** B.TXT
**********看雪-京东CTF2018**********
serial:success
*****
```

https://bbs.pediy.com/thread-228754.htm

运行批处理，因为正确输入和错误输入的返回结果不同，用来暴力，记录一下代码：

```
set a=100
echo %a%iamahandsomeguyhaha1 | CrackMe.exe > a.txt
:retry
set /a a=a+1
echo %a%iamahandsomeguyhaha1 | CrackMe.exe > b.txt
fc a.txt b.txt
if %ERRORLEVEL%==0 goto retry
```

还有这样的py姿势：

https://bbs.pediy.com/thread-228772.htm

```
# -*- coding: ascii -*-
import subprocess
sn3="iamahandsomeguyhaha1"
for i in range(100,999):
    p = subprocess.Popen("CrackMe.exe", stdin = subprocess.PIPE,\
    stdout = subprocess.PIPE, stderr = subprocess.PIPE, shell = True)
    sn=str(i)+sn3+'\n'
    outs,errs=p.communicate(bytes(sn.encode("ascii")))
    if(b"success" in outs):
        print(sn)
        break
```