

## 2016-pediy-ctf-3

笔记本： CTF

创建时间： 2019/3/18 17:44

更新时间： 2019/3/20 10:36

作者： 2278411737@qq.com

URL： <https://bbs.pediy.com/thread-213746.htm>

---

# 2016-pediy-ctf-3

---

IDA打开看到混淆，本打算带混淆直接搞，不过发现自己可能功力不够，看代码逻辑并不是很好，所以写脚本过混淆。

做完反混淆后，之后的工作就是IDA的F5直接查看代码了

前边的输入输出什么的就不讲了，还有一些赋值检测，这里检测

长度的检测

```
if ( length <= 256 && length >= 8 )
    break;
Sleep(0xBB8u);
v24 = -1;
sub_401C30(&v23, v10);
```

输入的检测

```
v19 = 0;
for ( i = 0; ; ++i )
{
    if ( i > length )
        goto LABEL_10;
    if ( *((char *)InputString + i) > 'Z' ) // CheckInput
        break;
}
v19 = 1;
```

主要看三个线程代码

```
sub_401C30(&v23, v10);
v12 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, 0, 0, &ThreadId);
v10 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_403120, 0, 0, &v18);
v11 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_402FA0, 0, 0, &v19);
Sleep(0xBB8u);
if ( *((BYTE *)pMalloc + 1) == 1 )
```

查看第一个和第三个线程的代码

发现并没有对结果产生影响，而且这两个线程的代码基本是一致的，不过还是分析其中一个，就当练功

首先根据时间得到0x20大小的随机字符串

```

} v10 = GetTickCount();
} v9 = v10 % 0x300;
L GetString_0(&v13, (_BYTE *)pMalloc + v10 % 0x300, 0x20u);
2 v15 = 0;
} v8 = GetTickCount();
L v7 = v8 % 0x208;
5 GetString_0(&v12, (_BYTE *)pMalloc + v8 % 0x208, 0x20u);
5 LOBYTE(v15) = 1;
7 v6 = &v2;
3 v11 = SetString(&v2, &v12);
} v5 = v11;
} LOBYTE(v15) = 2;
L v4 = &v1;
2 v3 = SetString(&v1, &v13);
3 LOBYTE(v15) = 1;

```

根据AES将其中一个作为Key一个作为Sn，计算AES得到的值放回到之前的一个大的数组中。

```

v10 = 1;
sub_401B40(&v8);
LOBYTE(v10) = 2;
length_ = GetLen(&a3);
string = (unsigned __int8 *)GetString(&a3);
Box(string, length_, 0, v7);
v5 = (unsigned __int8 *)GetString(&a2);
if ( AES(v5, &v9, v7) )
{
    sub_401B60(a1, ::a2);
}
else
{
    mov     [ebp+var_80], eax
    lea     ecx, [ebp+var_28] ; this
    call    GetLen
    push    eax                ; size_t
    lea     ecx, [ebp+var_28]
    call    GetString          ; Microsoft VisualC 2-14/net runtime
    push    eax                ; void *
    mov     ecx, pMalloc
    add     ecx, 100h
    push    ecx                ; void *
    call    memmove_0
    add     esp, 0Ch
    mov     [ebp+var_6C], 0
    lea     ecx, [ebp+var_28]
}

```

主要看第二个线程，首先获取K anxue-Crackme-CTF2016 字符串，并对其进行sha256和md5操作

```

sub_401B60(&v6, a2);
v7 = 0;
v1 = GetLen(&stru_43E5E0);
md5_(&v3, sha256); // Kanxue-Crackme-CTF2016

```

之后IDA的F5反汇编混乱，直接看代码

首先取输入每4个字节为一组对其进行RSA运算，其中

```

.text:0040325C      push    8
.text:0040325E      mov     eax, [ebp+var_BC]
.text:00403264      push    eax
.text:00403265      push    offset stru_43E5E0
.text:0040326A      lea     ecx, [ebp+var_88]
.text:00403270      call    Get8B
.text:00403270 ;   } // starts at 4031DD
.text:00403275 ;   try {
.text:00403275      mov     byte ptr [ebp+var_4], 2
.text:00403279      nop
.text:0040327A      nop
.text:004032B3      nop
.text:004032B4      lea     ecx, [ebp+var_88]
.text:004032BA      push    ecx
.text:004032BB      lea     edx, [ebp+var_58]
.text:004032BE      push    edx
.text:004032BF      call    SetStringHex
-----
.text:004032C4 ;   add     esp, 8
.text:004032C4 ;   } // starts at 403275
.text:004032C7 ;   try {
.text:004032C7      mov     byte ptr [ebp+var_4], 3
.text:004032CB      lea     ecx, [ebp+var_28]
.text:004032CE      call    sub_401B40
.text:004032CE ;   } // starts at 4032C7
.text:004032D3 ;   try {
.text:004032D3      mov     byte ptr [ebp+var_4], 4
.text:004032D7      lea     eax, [ebp+var_28]
.text:004032DA      push    eax
.text:004032DB      lea     ecx, [ebp+var_58]
.text:004032DE      push    ecx
.text:004032DF      call    RSA
.text:004032E4      add     esp, 8
.text:004032E7      mov     [ebp+var_DC], eax
.text:004032ED      nop

```

每次取4个字节

N = F574FD11

在线分解大素数

63587 \* 64763

利用rsatool得D

注意这里只获取密文的的前两个字节

这里记做

注：程序用了libtommath的库进行RSA的运算，并且鉴于当前笔者的水平，并没有对其中的RSA进行研究，所得的N是网上得来的，不过之后笔者会用该库学习rsa，进行更新，所以如果读者是看rsa的，就不用浪费时间了，这里没讲

最后程序将通过RSA得到的作为明文，以之前的sha256得到为key，得到密文sn,与之前的MD5作比较

所以逻辑为

MD5 == AES(RSA(), OUT, SHA256)

所以反过来: 首先aes解密得到RSA明文

|         |   |     |        |
|---------|---|-----|--------|
| 密钥 (K): | 0f5f7bcd0660d587c86d726b1c6aad103b21285c103a351bbc5733b3ad43635 | 长度: | 256Bit |
| 明文 (M): | 0D48ED5EF769E2ABD68BFD6C76DD795D                                |     |        |
| 密文 (C): | c8e5e2c3c439fc0448d80f8b5f738ca9                                |     |        |

然后根据RSA解密明文注意这里是没两个字节一组的，比较无语的是，我找到的解密工具全部是解密出可现实字符，所以这也是我决定学一下上边的理由之一

|                                       |          |  |
|---------------------------------------|----------|--|
| 进 制:                                  | 16       | 16进制   |
| 指数 (E):                               | 10001    |  |
| 模数 (N):                               | F574FD11 |  |
| 私钥 (D):                               | 2BBE7481 |  |
| 明文 (M):                               | <0?      |  |
| 密文 (C):                               | 0D48     |  |
| <input type="button" value="加密 (E)"/> |          | <input type="button" value="解密 (D)"/>              |
|                                       |          | 加密: $c = (m^e) \bmod n$<br>解密: $m = (c^d) \bmod n$ |

把他复制到

|         |                               |
|---------|-------------------------------|
| ASCII   | <0? . . . .                   |
| UNICODE | 4? . . . .                    |
| HEX +03 | 3C 4F 3<br>00 00 0<br>00 00 0 |

结果为:

3C4F963B039A2C377E02291E3C157AE591BCC1CA0A8F528EB2700AC021FB958D

反混淆代码如下

笔者水平有限，该脚本逻辑混乱,并且有些地方可能存在错误，需要在动态调试时进行更改，写来只是做个存档，请勿指责，如果读者有更好的脚本，还请告知，更新鸣谢

<https://github.com/ZEROSHE/ctf/blob/master/ctf/Pediy/2016%20ctf/3/test.py>

参考链接:

<https://ctf.pediy.com/game-fight-4.htm>