

VMP学习笔记之壳的重定位修复（五）

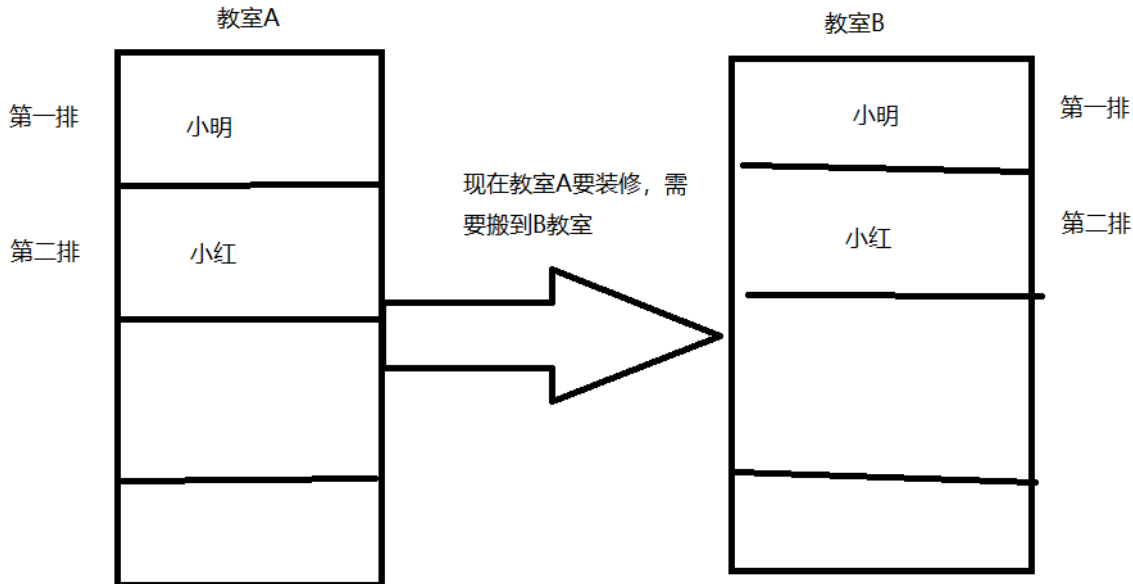
理论知识：

1、为什么需要重定位

因为特殊情况这块地址不给用了你要搬到另外一个地方，你本来是在哪里的搬到另外一个地方你还是在哪里

如下图所示：

小明在教室A是第一排，搬到教室B小明还是第一排



2、待修复数据如下：

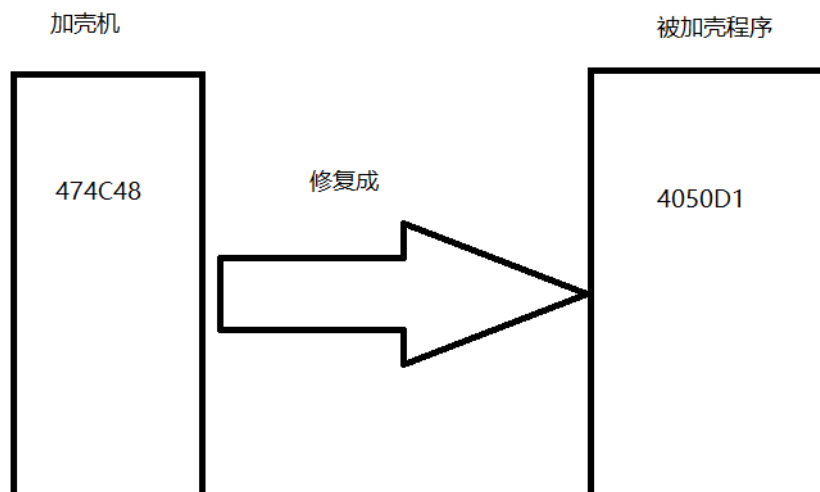
```
push 0xFACE0002      ----->修复后是: 0
mov edi,0xFACE0003    ----->修复后是: 新区段首地址VMcontext
jmp dword ptr ds:[eax*4+0x474FCF] ----->修复后是: Handle块
jmp short 00474984     ----->修复后是: VMDispatcher
```

总结：

- 1、这些地址直接使用肯定会报错，因为这些地址是基于加壳机的，并非我们被加壳程序的。
- 2、我们要将所有地址修复成基本我们被加壳程序的
- 3、我们要修改两处：struct_DisassemblyFunction->LODWORD_VMP_Address跟struct_DisassemblyFunction->ReadHexAddress
- 4、基本要修复都是有：Displacement、Immediate这些

3、如图所示

第一种：修复struct_DisassemblyFunction->LODWORD_VMP_Address



修复前：

地址	HEX 数据	ASCII
01445CE0	AC DC 47 00 A8 21 3F 01 E8 5D 44 01 00 00 00 00	G.?? 墨 D f..
01445CF0	48 4C 47 00 00 00 00 00 49 4C 47 00 00 00 00 00	HLG....ILG..
01445D00	00 00 00 00 02 00 00 01 04 00 00 00 00 00 00 00 f!....
01445D10	00 00 00 00 00 00 00 00 00 00 FF FF FF FF 00 00uuuu
01445D20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01445D30	00 FF FF FF FF 00 00 00 00 00 00 00 00 00 00	iiiiiii

修复后:

地址	HEX 数据	ASCII
1445CE0	AC DC 47 00 A8 21 3F 01 E8 5D 44 01 00 00 00 00	G.?? 墨 D f...
1445CF0	D1 50 40 00 00 00 00 00 49 4C 47 00 00 00 00 00	烟@....ILG....
1445D00	00 00 00 00 02 00 00 01 04 00 00 00 00 00 00 00 f!....
1445D10	00 00 00 00 00 00 00 00 00 00 FF FF FF FF 00 00
1445D20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1445D30	00 FF FF FF FF 00 00 00 00 00 00 00 00 00 00	iiiiiii

第二种：修复struct_DisassemblyFunction->ReadHexAddress

典型例子1：

00474991 FF2485 CF4F700 jmp dword ptr ds:[eax*4+0x474FCF]这一句不修复肯定执行会错误

我们要修复这一句的Displacement，改成一个被加壳程序存在的地址就OK了

例如：jmp dword ptr ds:[eax*4+0x4051BB]

典型例子2：

Jmp VMDispatcher

公式：jmp后的偏移量=目标地址-原来地址-5 5是指令长度

正文：

1、筛选Esi_Addr[X]结构数据保存到ruct_VmpOpcodePY_50结构

将struct_DisassemblyFunction所在的数组下标保存到struc_69结构里

```

*(DWORD *) (TList::Add(v3->struct_VmpOpcodePY_50) + 8) = 0;
v5 = 0;
do
{
    v6 = *(DWORD *) &v3->Esi_Addr[4 * v5]; // 循环作用：1、将Esi_Addr[X]的struct_DisassemblyFunction添加到保存起来
    if (v6)
    {
        ArrayNumber = TList::IndexOf(v6, v4); // 返回第几项找到的，返回该struct_DisassemblyFunction结构在struc_SaveAllDisasmFunData->ArrayNumber
        if (DateTimeToStr_5(
            (struct_VmpOpcode *) v3->struct_VmpOpcodePY_50,
            ArrayNumber,
            (unsigned __int64) ArrayNumber >> 0x20) == -1) // 列表中找不到就添加
        {
            *(DWORD *) (TList::Add(v3->struct_VmpOpcodePY_50) + 8) = ArrayNumber; // 保存数组下标
        }
        ++v5;
    }
    while (v5 != 0xCC); // Esi大小
}
挑选jmp ret指令保存起来
ArrayNumber_1 = TCollection::GetCount_0(v3->struct_VmpOpcodePY_50) - 1;
if (ArrayNumber_1 >= 0)
{
    ArrayNumber_2 = ArrayNumber_1 + 1;
    v10 = 0;
    do
    {
        v83 = (struct_69 *) TCollection::GetItem_7(v3->struct_VmpOpcodePY_50, v10, v9);
        v11 = v83->ArrayNumber;
        v12 = TCollection::GetCount_0((int) v3); // struc_SaveAllDisasmFunData->ArrayNumber (基本版)
        if (v12 - 1 >= v11) // 判断数组下标合法性
        {
            v13 = v12 - 1 - v11 + 1;
            while (1)
            {
                VMOpcode = *(DWORD *) (GetItem_7((int) v3, v11, v9) + 0x24) - 8;
                v15 = VMOpcode < 2u;
                v16 = VMOpcode - 2;
                if (v15 || v16 == 2) // VMOpcode: 8 9 =ret C=jmp
                {
                    break;
                }
                ++v11;
                if (v11 == v13)
                {
                    goto LABEL_14;
                }
                v83->FilterArrayNumber = v11; // 符合条件VMOpcode保存起来
            }
        }
        LABEL_14:
        ++v10;
        --ArrayNumber_2;
    }
    while (ArrayNumber_2); // 找到struct_DisassemblyFunction->VMOpcode值是：8 9 =ret C=jmp，然后将该结构所在的数组下标保存起来
}

```

struc_69结构定义如下：

```

00000000 struc_69 struct ; (sizeof=0x34, mappedto_353)
00000000 This dd ?
00000004 prev_node dd ?
00000008 ArrayNumber dd ? ; 指向struct_VmpOpcodePY_50
0000000C FilterArrayNumber dd ? ; 返回第几项找到该struct_DisassemblyFunction结构在struc_SaveAllDisasmFunData->ArrayNumber
00000010 field_10 dd ? ; 找到struct_DisassemblyFunction->VMOpcode值是：8 9 =ret C=jmp，然后将该结构所在的数组下标保存起来
00000014 field_14 dd ?
00000018 field_18 dd ?
0000001C field_1C dd ?
00000020 field_20 dd ?
00000024 field_24 dd ?
00000028 field_28 dd ?
0000002C field_2C dd ?
00000030 field_30 dd ?
00000034 struc_69 ends

```

总结：

1、struc_69结构

struc_69->ArrayNumber == Handle块头

struc_69->FilterArrayNumber == Handle块结束地址, 也就是jmp或则ret

2、找到第一组Handle块在数组的元素下标, 作为结尾标识

符合条件的是: 解析JmpAddr跳转表, 填充的VMOpcode都是0x23

```
// v17 = TCollection::GetCount_0((int)v3) - 1; // struc_SaveAllDisasmFunData->ArrayNumber (基本版)
if ( v17 >= 0 ) // 功能: 1、找到解析JmpAddr跳转表的解析结构, 并找到该结构在数组的第几个下标, 取出并保存起来
{
    ArrayNumber_2a = v17 + 1;
    v19 = 0;
    while ( 1 )
    {
        v20 = (struct_DisassemblyFunction *)GetItem_7((int)v3, v19, v18);
        v21 = v20;
        v22 = (struct_UnFunctionAddr *)v20->struct_UnFunctionAddr;
        if ( v22 )
        {
            if ( v22->CheckDisassemblyFunction && *(_WORD *) (v22->CheckDisassemblyFunction + 0x24) == 0x23 )// 解析JmpAddr跳转表, 填充的VMOpcode都是0x23
                break;
        }
        ++v19;
        if ( !--ArrayNumber_2a )
            goto LABEL_22;
    }
    v23 = (struct_69 *)TList::Add(v3->struct_UnopcodePY_50);
    v23->ArrayNumber = TList::Index0F(*(_DWORD *) (v21->struct_UnFunctionAddr + 0x1C), v24);// 该struc_SaveAllDisasmFunData->ArrayNumber (基本版)在数组第几项
    v23->FilterArrayNumber = TCollection::GetCount_0((int)v3) - 1; // 保存struc_SaveAllDisasmFunData->ArrayNumber (基本版)
}
LABEL_22:
v25 = TCollection::GetCount_0(v3->struct_UnopcodePY_50) - 1;
```

然后对struc_VmpOpcodePY_50结构进行数组乱序

```
64 LABEL_22: |
65 v25 = TCollection::GetCount_0(v3->struct_UnopcodePY_50) - 1;
66 if ( v25 >= 0 )
67 {
68     ArrayNumber_2b = v25 + 1;
69     v26 = 0;
70     do
71     {
72         v27 = v3->struct_UnopcodePY_50;
73         TCollection::GetCount_0(v3->struct_UnopcodePY_50);
74         v28 = __linkproc_RandInt(); // 以ArrayNumber作为随机数种子
75         TList::Exchange(v27, v26++, v28);
76         --ArrayNumber_2b;
77     }
78     while ( ArrayNumber_2b ); // 乱序数组排列顺序
79 }
```

3、循环填充struc_SaveAllDisasmFunData->ArrayNumber (基本版) Vmp_Address(0x10)替换成实际壳的真实地址

```
v29 = TCollection::GetCount_0(v3->struct_UnopcodePY_50) - 1;
if ( v29 >= 0 ) // 循环填充struc_SaveAllDisasmFunData->ArrayNumber (基本版) Vmp_Address(0x10)替换成实际壳的地址
{
    ArrayNumber_2c = v29 + 1;
    v31 = 0;
    do
    {
        v84 = (struct_69 *)TCollection::GetItem_7(v3->struct_UnopcodePY_50, v31, (int)v30);
        ReadHexLen = 0;
        v33 = v84->ArrayNumber; // 该handle块首地址
        v34 = v84->FilterArrayNumber; // 该handle块结束地址
        v35 = __OFSUB__(v33, v34); // 结束的 - 起始的 = 获取一共有多少组Opcode
        v36 = v34 - v33;
        if ( !(v36 < 0) ^ v35 ) // 计算出该handle块指令长度是多少, 第一条指令到jmp Dispatcher (jmp Dispatcher不算)为结束
        {
            v37 = v36 + 1;
            do
            {
                ReadHexLen += *(_DWORD *) (GetItem_7((int)v3, v33++, v32) + 0x6C); // ReadHexLen dd ? ; 读取Opcode长度, 每次执行ReadHex函数便会累加
                --v37;
            } while ( v37 );
        }
        UserVmpStartAddr = sub_480BE0(a1a, ReadHexLen, v3->Characteristics & 0x20, 0i64); // 找到有足够空间存放该VmpHandle块的区段, 并返回该区段的开始地址
        if ( !v84->ArrayNumber )
            *(_QWORD *) &v3->VmpStubStart = UserVmpStartAddr; // 保存壳的跳到解析器的地址 jmp UstartVM
        v38 = v84->ArrayNumber;
        v39 = v84->FilterArrayNumber;
        v35 = __OFSUB__(v39, v38); // 结束的 - 起始的 = 获取一共有多少组Opcode
        v40 = v39 - v38;
        if ( !(v40 < 0) ^ v35 ) // 将找到的struc_SaveAllDisasmFunData->ArrayNumber (基本版)的Vmp_Address(0x10)替换成前面找到的UserVmpStartAddr (0x8)
        {
            v41 = v40 + 1;
            do
            {
                v30 = (struct_DisassemblyFunction *)GetItem_7((int)v3, v38, (int)v30);
                *(_QWORD *) &v3->LODWORD_VMP_Address = UserVmpStartAddr;
                UserVmpStartAddr += v30->ReadHexLen;
                ++v38;
                --v41;
            }
        }
    } while ( v29 >= 0 );
}
```

每次执行完后sub_480BE0(a1a, ReadHexLen, v3->Characteristics & 0x20, 0i64)函数

struc_59->LODWORD_UserVmpStartAddr+=ReadHexLen; (默认的壳起始OEP (新区段的起始地址))

地址	HEX 数据	ASCII
014113C0	3C DD 47 00 7C D7 3F 01 4B 50 40 00 00 00 00 00	<薑. ?AkPG....
014113D0	FF FF FF FF FF FF FF 7F 00 00 00 00 0A 00 00 00	????????
014113E0	4F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

然后替换掉struc_DisassemblyFunction->LODWORD_VMP_Address为真正的地址

```

7  v41 = v40 - v40;
8  if ( !(v40 < 0) ^ v35 ) // 将找到的struct_SaveAllDisasmFunData->ArrayNumber (基本版)的Ump_Address(0x10)替换成前面找到的UserUmpStartAddr(0x8)
9  {
10     v41 = v40 + 1;
11     do
12     {
13         v38 = (struct_DisassemblyFunction *)GetItem_7((int)v3, v38, (int)v30);
14         *(_QWORD *)&v38->L0DWORD_Ump_Address = UserUmpStartAddr;
15         UserUmpStartAddr += v38->ReadHexLen;
16         ++v38;
17         --v41;
18     }
19     while ( v41 ); // Opcode有几行就循环几次
20 }
21 ++v31;
22 --ArrayNumber_2c;
23 while ( ArrayNumber_2c );

```

总结:

1、前面所有工作都是找到所有使用到的struct_DisassemblyFunction结构, 然后对其地址进行替换成真实壳地址

替换前后对比:

前:

地址	HEX 数据	ASCII
01445CE0	AC DC 47 00 A8 21 3F 01 E8 5D 44 01 00 00 00 00	G.?? 墨 D f..
01445CF0	48 4C 47 00 00 00 00 00 49 4C 47 00 00 00 00 00	HLG.....ILG..
01445D00	00 00 00 00 02 00 00 01 04 00 00 00 00 00 00 00 f!.....
01445D10	00 00 00 00 00 00 00 00 00 00 FF FF FF FF 00 00 yyy!
01445D20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01445D30	00 FF FF FF FF 00 00 00 00 00 00 00 00 00 00	iiiiiii

后:

地址	HEX 数据	ASCII
1445CE0	AC DC 47 00 A8 21 3F 01 E8 5D 44 01 00 00 00 00	G.?? 墨 D f..
1445CF0	D1 50 40 00 00 00 00 00 49 4C 47 00 00 00 00 00	烟@.....ILG....
1445D00	00 00 00 00 02 00 00 01 04 00 00 00 00 00 00 00 f!.....
1445D10	00 00 00 00 00 00 00 00 00 00 FF FF FF FF 00 00 yyy!

4、修复特定值

执行前	转换	执行后
push 0xFACE0002	----- >	Push 重定位值
mov edi,0xFACE0003	----- >	Mov edi,VMContext

```

7  TlistArrayClear(v3->struct_UnpOpcodePV_50, (int)v30); // 内容全部释放
8  FixArray((int)v3); // 修复数组排列顺序, 从小到大
9  v42 = TCollection::GetCount_0(v3->struct_UnpOpcodePV_3D4) - 1; // 该结构体保存数据分别是: 1、修复重定位地址(push XXXX<-push 0xFACE0002) 2、edi (VMContext<-mov edi,0xFACE
10 if ( v42 >= 0 ) // 修复指令重定位信息
11 {
12     ArrayNumber_2d = v42 + 1;
13     v44 = 0;
14     do
15     {
16         v45 = v3->struct_UnpOpcodePV_3D4;
17         v46 = (struct_Edi_Addr *)TCollection::GetItem_10(v43, v44);
18         v48 = (struct_DisassemblyFunction *)v46->struct_DisassemblyFunction;
19         v49 = 0x17 * v46->Number;
20         v50 = (struct_Disasm *)((char *)&v48->First + v49);
21         v51 = v46->Constant_Byte - 1; // 保存作者设计Handle里面的有特殊含义常量(0xFACExxxx)最低字节, 例如push 0xFACE0002的02
22         if ( v51 ) // 1
23         {
24             v52 = v51 - 1;
25             if ( v52 ) // 2
26             {
27                 v53 = v52 - 1;
28                 if ( v53 ) // 3
29                 {
30                     if ( v53 == 1 ) // 4 call dword ptr ds:[0xFACE0004]
31                     {
32                         v56 = v3->prev_node;
33                         v57 = *(_DWORD *)(&v56 + 0x10) + 0x34;
34                         v74 = *(_QWORD *)(&v56 + 0x38) + 8;
35                         if ( v50->Mod_Rm )
36                         {
37                             *(_QWORD *)&v50->L0DWORD_RestHex_Lval_Displacement_Immediate = v74 - Unp_GetNextAddressStart(v48);
38                         }
39                         else
40                         {
41                             *(_QWORD *)&v50->L0DWORD_RestHex_Lval_Displacement_Immediate = v74;
42                             v50->Tag = -2;
43                         }
44                     }
45                 }
46             }
47         }
48     }
49 }

```

5、修复所有 JMP VMDispatcher的地址

公式: jmp后的偏移量=目标地址-原来地址-5 5是指令长度

```

++u/2;
}
else if ( v61->UMOpcode != 0xC || 2 != v61->First.ModRM_mod_Or_Size )// UMopcode == 0xC是jmp系列
{
    v67 = (struct_UnFunctionAddr *)v61->struct_UnFunctionAddr;// 判断是否是Handler块函数
    if ( v67 && LOBYTE(v67->Un_Mnemonic) == 0xC && v67->CheckDisassemblyFunction )// 解析JmpAddr跳转表结束后,再生成一组以0xC标记结尾
    {
        v68 = 0;
        while ( !*((_BYTE *)&v62->First.ModRM_mod_Or_Size + 0x17 * v68) & 2 )
        {
            if ( ++v68 == 3 )
                goto LABEL_71;
        }
        v75 = *(_QWORD *)((_DWORD *)v62->struct_UnFunctionAddr + 0x1C) + 0x10;
        if ( *((&v62->First.Mod_Rm + 0x17 * v68) )
        {
            v69 = Unp_GetNextAddressStart(v62);
            v59 = (int)(v62->First.About_Lval_Byte_Word_Dword + 0x17 * v68);
            *(_QWORD *)((char *)&v62->First.LODWORD_RestHex_Lval_Displacement_Immediate + 0x17 * v68) = v75 - v69;
        }
        else
        {
            *(_QWORD *)((char *)&v62->First.LODWORD_RestHex_Lval_Displacement_Immediate + 0x17 * v68) = v75;
            *(int *)((char *)&v62->First.Tag + 0x17 * v68) = -2;
        }
    }
LABEL_71:
    SetDisassemblyFunction(v62, v59); // 修复重定位 Jmp dword ptr [eax*4+JumpAddr];跳到Handler执行处,由加壳引擎填充
}
else
{
    v65 = *(_QWORD *)((_DWORD *)v61->struct_UnFunctionAddr + 0x1C) + 0x10;
    *(_QWORD *)&v61->First.LODWORD_RestHex_Lval_Displacement_Immediate = v65 - Unp_GetNextAddressStart(v61);
    SetDisassemblyFunction(v62, v66); // 修复重定位 Jmp UMDispatcher
}
++v68;
--ArrayNumber_2e;
}
while ( ArrayNumber_2e );

```

6、修复完毕后的样子

05053	55	push ebp	
05054	68 00000000	push 0x0	
05059	8B7424 28	mov esi,dword ptr ss:[esp+0x28]	
0505D	BF 00504000	mov edi,HelloASM.00405000	
05062	89F3	mov ebx,esi	HelloASM.<ModuleEntryPo
05064	033424	add esi,dword ptr ss:[esp]	kernel132.77998484
05067	8A06	mov al,byte ptr ds:[esi]	
05069	0008	add al,bl	
0506B	FEC0	inc al	
0506D	C0C0 05	rol al,0x5	
05070	F6D0	not al	
05072	2C B0	sub al,0xB0	
05074	F6D0	not al	
05076	34 7A	xor al,0x7A	
05078	0D76 01	lea esi,dword ptr ds:[esi+0x1]	
0507B	00C3	add bl,al	
0507D	0FB6C0	movzx eax,al	
05080	FF3485 BB51400	push dword ptr ds:[eax*4+0x4051BB]	
05087	C3	ret	
05088	54	push esp	
05089	E9 D9FFFFFF	jmp HelloASM.00405067	
0508E	58	pop eax	kernel132.77998484
0508F	8F00	pop dword ptr ds:[eax]	kernel132.77998484
05091	E9 D1FFFFFF	jmp HelloASM.00405067	
05096	58	pop eax	kernel132.77998484
05097	5A	pop edx	kernel132.77998484
05098	66 59	pop cx	

下一篇文章介绍:

1、伪代码构造

2、伪代码加密 (前面构成出Add的解密代码,所以这里要对称整出一套加密代码)

```

0000074 Field_74 dd ?
0000078 Field_78 dd ?
000007C Field_7C dd ?
0000080 struct_UnOpcodePY_80 dd ?
0000084 struct_UnOpcodePY_84 dd ?
0000088 struct_UnOpcodePY_88 dd ?
000008C struct_UnOpcodePY_8C dd ?
0000090 struct_UnOpcodePY_90 dd ?
0000094 struct_UnOpcodePY_94 dd ?
0000098 struct_UnOpcodePY_98 dd ?
000009C struct_UnOpcodePY_9C dd ?
00000A0 struct_UnOpcodePY_A0 dd ?
00000A4 Esi_addr db 816 dup(?)
00000D4 struct_UnOpcodePY_3D4 dd ?
00000D8 struc_PushRegister dd ?
00000DC struct_UnOpcode ends
00000E0 ;
00000E4 ;
00000E8 ;
00000EC ;
00000F0 ;
00000F4 ;
00000F8 ;
00000FC ;
0000100 ;
0000104 ;
0000108 ;
000010C ;
0000110 ;
0000114 ;
0000118 ;
000011C ;
0000120 ;
0000124 ;
0000128 ;
000012C ;
0000130 ;
0000134 ;
0000138 ;
000013C ;
0000140 ;
0000144 ;
0000148 ;
000014C ;
0000150 ;
0000154 ;
0000158 ;
000015C ;
0000160 ;
0000164 ;
0000168 ;
000016C ;
0000170 ;
0000174 ;
0000178 ;
000017C ;
0000180 ;
0000184 ;
0000188 ;
000018C ;
0000190 ;
0000194 ;
0000198 ;
000019C ;
00001A0 ;
00001A4 ;
00001A8 ;
00001AC ;
00001B0 ;
00001B4 ;
00001B8 ;
00001BC ;
00001C0 ;
00001C4 ;
00001C8 ;
00001CC ;
00001D0 ;
00001D4 ;
00001D8 ;
00001DC ;
00001E0 ;
00001E4 ;
00001E8 ;
00001EC ;
00001F0 ;
00001F4 ;
00001F8 ;
00001FC ;
0000200 ;
0000204 ;
0000208 ;
000020C ;
0000210 ;
0000214 ;
0000218 ;
000021C ;
0000220 ;
0000224 ;
0000228 ;
000022C ;
0000230 ;
0000234 ;
0000238 ;
000023C ;
0000240 ;
0000244 ;
0000248 ;
000024C ;
0000250 ;
0000254 ;
0000258 ;
000025C ;
0000260 ;
0000264 ;
0000268 ;
000026C ;
0000270 ;
0000274 ;
0000278 ;
000027C ;
0000280 ;
0000284 ;
0000288 ;
000028C ;
0000290 ;
0000294 ;
0000298 ;
000029C ;
00002A0 ;
00002A4 ;
00002A8 ;
00002AC ;
00002B0 ;
00002B4 ;
00002B8 ;
00002BC ;
00002C0 ;
00002C4 ;
00002C8 ;
00002CC ;
00002D0 ;
00002D4 ;
00002D8 ;
00002DC ;
00002E0 ;
00002E4 ;
00002E8 ;
00002EC ;
00002F0 ;
00002F4 ;
00002F8 ;
00002FC ;
0000300 ;
0000304 ;
0000308 ;
000030C ;
0000310 ;
0000314 ;
0000318 ;
000031C ;
0000320 ;
0000324 ;
0000328 ;
000032C ;
0000330 ;
0000334 ;
0000338 ;
000033C ;
0000340 ;
0000344 ;
0000348 ;
000034C ;
0000350 ;
0000354 ;
0000358 ;
000035C ;
0000360 ;
0000364 ;
0000368 ;
000036C ;
0000370 ;
0000374 ;
0000378 ;
000037C ;
0000380 ;
0000384 ;
0000388 ;
000038C ;
0000390 ;
0000394 ;
0000398 ;
000039C ;
00003A0 ;
00003A4 ;
00003A8 ;
00003AC ;
00003B0 ;
00003B4 ;
00003B8 ;
00003BC ;
00003C0 ;
00003C4 ;
00003C8 ;
00003CC ;
00003D0 ;
00003D4 ;
00003D8 ;
00003DC ;
00003E0 ;
00003E4 ;
00003E8 ;
00003EC ;
00003F0 ;
00003F4 ;
00003F8 ;
00003FC ;
0000400 ;
0000404 ;
0000408 ;
000040C ;
0000410 ;
0000414 ;
0000418 ;
000041C ;
0000420 ;
0000424 ;
0000428 ;
000042C ;
0000430 ;
0000434 ;
0000438 ;
000043C ;
0000440 ;
0000444 ;
0000448 ;
000044C ;
0000450 ;
0000454 ;
0000458 ;
000045C ;
0000460 ;
0000464 ;
0000468 ;
000046C ;
0000470 ;
0000474 ;
0000478 ;
000047C ;
0000480 ;
0000484 ;
0000488 ;
000048C ;
0000490 ;
0000494 ;
0000498 ;
000049C ;
00004A0 ;
00004A4 ;
00004A8 ;
00004AC ;
00004B0 ;
00004B4 ;
00004B8 ;
00004BC ;
00004C0 ;
00004C4 ;
00004C8 ;
00004CC ;
00004D0 ;
00004D4 ;
00004D8 ;
00004DC ;
00004E0 ;
00004E4 ;
00004E8 ;
00004EC ;
00004F0 ;
00004F4 ;
00004F8 ;
00004FC ;
0000500 ;
0000504 ;
0000508 ;
000050C ;
0000510 ;
0000514 ;
0000518 ;
000051C ;
0000520 ;
0000524 ;
0000528 ;
000052C ;
0000530 ;
0000534 ;
0000538 ;
000053C ;
0000540 ;
0000544 ;
0000548 ;
000054C ;
0000550 ;
0000554 ;
0000558 ;
000055C ;
0000560 ;
0000564 ;
0000568 ;
000056C ;
0000570 ;
0000574 ;
0000578 ;
000057C ;
0000580 ;
0000584 ;
0000588 ;
000058C ;
0000590 ;
0000594 ;
0000598 ;
000059C ;
00005A0 ;
00005A4 ;
00005A8 ;
00005AC ;
00005B0 ;
00005B4 ;
00005B8 ;
00005BC ;
00005C0 ;
00005C4 ;
00005C8 ;
00005CC ;
00005D0 ;
00005D4 ;
00005D8 ;
00005DC ;
00005E0 ;
00005E4 ;
00005E8 ;
00005EC ;
00005F0 ;
00005F4 ;
00005F8 ;
00005FC ;
0000600 ;
0000604 ;
0000608 ;
000060C ;
0000610 ;
0000614 ;
0000618 ;
000061C ;
0000620 ;
0000624 ;
0000628 ;
000062C ;
0000630 ;
0000634 ;
0000638 ;
000063C ;
0000640 ;
0000644 ;
0000648 ;
000064C ;
0000650 ;
0000654 ;
0000658 ;
000065C ;
0000660 ;
0000664 ;
0000668 ;
000066C ;
0000670 ;
0000674 ;
0000678 ;
000067C ;
0000680 ;
0000684 ;
0000688 ;
000068C ;
0000690 ;
0000694 ;
0000698 ;
000069C ;
00006A0 ;
00006A4 ;
00006A8 ;
00006AC ;
00006B0 ;
00006B4 ;
00006B8 ;
00006BC ;
00006C0 ;
00006C4 ;
00006C8 ;
00006CC ;
00006D0 ;
00006D4 ;
00006D8 ;
00006DC ;
00006E0 ;
00006E4 ;
00006E8 ;
00006EC ;
00006F0 ;
00006F4 ;
00006F8 ;
00006FC ;
0000700 ;
0000704 ;
0000708 ;
000070C ;
0000710 ;
0000714 ;
0000718 ;
000071C ;
0000720 ;
0000724 ;
0000728 ;
000072C ;
0000730 ;
0000734 ;
0000738 ;
000073C ;
0000740 ;
0000744 ;
0000748 ;
000074C ;
0000750 ;
0000754 ;
0000758 ;
000075C ;
0000760 ;
0000764 ;
0000768 ;
000076C ;
0000770 ;
0000774 ;
0000778 ;
000077C ;
0000780 ;
0000784 ;
0000788 ;
000078C ;
0000790 ;
0000794 ;
0000798 ;
000079C ;
00007A0 ;
00007A4 ;
00007A8 ;
00007AC ;
00007B0 ;
00007B4 ;
00007B8 ;
00007BC ;
00007C0 ;
00007C4 ;
00007C8 ;
00007CC ;
00007D0 ;
00007D4 ;
00007D8 ;
00007DC ;
00007E0 ;
00007E4 ;
00007E8 ;
00007EC ;
00007F0 ;
00007F4 ;
00007F8 ;
00007FC ;
0000800 ;
0000804 ;
0000808 ;
000080C ;
0000810 ;
0000814 ;
0000818 ;
000081C ;
0000820 ;
0000824 ;
0000828 ;
000082C ;
0000830 ;
0000834 ;
0000838 ;
000083C ;
0000840 ;
0000844 ;
0000848 ;
000084C ;
0000850 ;
0000854 ;
0000858 ;
000085C ;
0000860 ;
0000864 ;
0000868 ;
000086C ;
0000870 ;
0000874 ;
0000878 ;
000087C ;
0000880 ;
0000884 ;
0000888 ;
000088C ;
0000890 ;
0000894 ;
0000898 ;
000089C ;
00008A0 ;
00008A4 ;
00008A8 ;
00008AC ;
00008B0 ;
00008B4 ;
00008B8 ;
00008BC ;
00008C0 ;
00008C4 ;
00008C8 ;
00008CC ;
00008D0 ;
00008D4 ;
00008D8 ;
00008DC ;
00008E0 ;
00008E4 ;
00008E8 ;
00008EC ;
00008F0 ;
00008F4 ;
00008F8 ;
00008FC ;
0000900 ;
0000904 ;
0000908 ;
000090C ;
0000910 ;
0000914 ;
0000918 ;
000091C ;
0000920 ;
0000924 ;
0000928 ;
000092C ;
0000930 ;
0000934 ;
0000938 ;
000093C ;
0000940 ;
0000944 ;
0000948 ;
000094C ;
0000950 ;
0000954 ;
0000958 ;
000095C ;
0000960 ;
0000964 ;
0000968 ;
000096C ;
0000970 ;
0000974 ;
0000978 ;
000097C ;
0000980 ;
0000984 ;
0000988 ;
000098C ;
0000990 ;
0000994 ;
0000998 ;
000099C ;
00009A0 ;
00009A4 ;
00009A8 ;
00009AC ;
00009B0 ;
00009B4 ;
00009B8 ;
00009BC ;
00009C0 ;
00009C4 ;
00009C8 ;
00009CC ;
00009D0 ;
00009D4 ;
00009D8 ;
00009DC ;
00009E0 ;
00009E4 ;
00009E8 ;
00009EC ;
00009F0 ;
00009F4 ;
00009F8 ;
00009FC ;
0000A00 ;
0000A04 ;
0000A08 ;
0000A0C ;
0000A10 ;
0000A14 ;
0000A18 ;
0000A1C ;
0000A20 ;
0000A24 ;
0000A28 ;
0000A2C ;
0000A30 ;
0000A34 ;
0000A38 ;
0000A3C ;
0000A40 ;
0000A44 ;
0000A48 ;
0000A4C ;
0000A50 ;
0000A54 ;
0000A58 ;
0000A5C ;
0000A60 ;
0000A64 ;
0000A68 ;
0000A6C ;
0000A70 ;
0000A74 ;
0000A78 ;
0000A7C ;
0000A80 ;
0000A84 ;
0000A88 ;
0000A8C ;
0000A90 ;
0000A94 ;
0000A98 ;
0000A9C ;
0000AA0 ;
0000AA4 ;
0000AA8 ;
0000AAC ;
0000AB0 ;
0000AB4 ;
0000AB8 ;
0000ABC ;
0000AC0 ;
0000AC4 ;
0000AC8 ;
0000ACC ;
0000AD0 ;
0000AD4 ;
0000AD8 ;
0000ADC ;
0000AE0 ;
0000AE4 ;
0000AE8 ;
0000AEC ;
0000AF0 ;
0000AF4 ;
0000AF8 ;
0000AFC ;
0000B00 ;
0000B04 ;
0000B08 ;
0000B0C ;
0000B10 ;
0000B14 ;
0000B18 ;
0000B1C ;
0000B20 ;
0000B24 ;
0000B28 ;
0000B2C ;
0000B30 ;
0000B34 ;
0000B38 ;
0000B3C ;
0000B40 ;
0000B44 ;
0000B48 ;
0000B4C ;
0000B50 ;
0000B54 ;
0000B58 ;
0000B5C ;
0000B60 ;
0000B64 ;
0000B68 ;
0000B6C ;
0000B70 ;
0000B74 ;
0000B78 ;
0000B7C ;
0000B80 ;
0000B84 ;
0000B88 ;
0000B8C ;
0000B90 ;
0000B94 ;
0000B98 ;
0000B9C ;
0000BA0 ;
0000BA4 ;
0000BA8 ;
0000BAC ;
0000BB0 ;
0000BB4 ;
0000BB8 ;
0000BBC ;
0000BC0 ;
0000BC4 ;
0000BC8 ;
0000BCC ;
0000BD0 ;
0000BD4 ;
0000BD8 ;
0000BDC ;
0000BE0 ;
0000BE4 ;
0000BE8 ;
0000BEC ;
0000BF0 ;
0000BF4 ;
0000BF8 ;
0000BFC ;
0000C00 ;
0000C04 ;
0000C08 ;
0000C0C ;
0000C10 ;
0000C14 ;
0000C18 ;
0000C1C ;
0000C20 ;
0000C24 ;
0000C28 ;
0000C2C ;
0000C30 ;
0000C34 ;
0000C38 ;
0000C3C ;
0000C40 ;
0000C44 ;
0000C48 ;
0000C4C ;
0000C50 ;
0000C54 ;
0000C58 ;
0000C5C ;
0000C60 ;
0000C64 ;
0000C68 ;
0000C6C ;
0000C70 ;
0000C74 ;
0000C78 ;
0000C7C ;
0000C80 ;
0000C84 ;
0000C88 ;
0000C8C ;
0000C90 ;
0000C94 ;
0000C98 ;
0000C9C ;
0000CA0 ;
0000CA4 ;
0000CA8 ;
0000CAC ;
0000CB0 ;
0000CB4 ;
0000CB8 ;
0000CBC ;
0000CC0 ;
0000CC4 ;
0000CC8 ;
0000CCC ;
0000CD0 ;
0000CD4 ;
0000CD8 ;
0000CDC ;
0000CE0 ;
0000CE4 ;
0000CE8 ;
0000CEC ;
0000CF0 ;
0000CF4 ;
0000CF8 ;
0000CFC ;
0000D00 ;
0000D04 ;
0000D08 ;
0000D0C ;
0000D10 ;
0000D14 ;
0000D18 ;
0000D1C ;
0000D20 ;
0000D24 ;
0000D28 ;
0000D2C ;
0000D30 ;
0000D34 ;
0000D38 ;
0000D3C ;
0000D40 ;
0000D44 ;
0000D48 ;
0000D4C ;
0000D50 ;
0000D54 ;
0000D58 ;
0000D5C ;
0000D60 ;
0000D64 ;
0000D68 ;
0000D6C ;
0000D70 ;
0000D74 ;
0000D78 ;
0000D7C ;
0000D80 ;
0000D84 ;
0000D88 ;
0000D8C ;
0000D90 ;
0000D94 ;
0000D98 ;
0000D9C ;
0000DA0 ;
0000DA4 ;
0000DA8 ;
0000DAC ;
0000DB0 ;
0000DB4 ;
0000DB8 ;
0000DBC ;
0000DC0 ;
0000DC4 ;
0000DC8 ;
0000DCC ;
0000DD0 ;
0000DD4 ;
0000DD8 ;
0000DDC ;
0000DE0 ;
0000DE4 ;
0000DE8 ;
0000DEC ;
0000DF0 ;
0000DF4 ;
0000DF8 ;
0000DFC ;
0000E00 ;
0000E04 ;
0000E08 ;
0000E0C ;
0000E10 ;
0000E14 ;
0000E18 ;
0000E1C ;
0000E20 ;
0000E24 ;
0000E28 ;
0000E2C ;
0000E30 ;
0000E34 ;
0000E38 ;
0000E3C ;
0000E40 ;
0000E44 ;
0000E48 ;
0000E4C ;
0000E50 ;
0000E54 ;
0000E58 ;
0000E5C ;
0000E60 ;
0000E64 ;
0000E68 ;
0000E6C ;
0000E70 ;
0000E74 ;
0000E78 ;
0000E7C ;
0000E80 ;
0000E84 ;
0000E88 ;
0000E8C ;
0000E90 ;
0000E94 ;
0000E98 ;
0000E9C ;
0000EA0 ;
0000EA4 ;
0000EA8 ;
0000EAC ;
0000EB0 ;
0000EB4 ;
0000EB8 ;
0000EBC ;
0000EC0 ;
0000EC4 ;
0000EC8 ;
0000ECC ;
0000ED0 ;
0000ED4 ;
0000ED8 ;
0000EDC ;
0000EE0 ;
0000EE4 ;
0000EE8 ;
0000EEC ;
0000EF0 ;
0000EF4 ;
0000EF8 ;
0000EFC ;
0000F00 ;
0000F04 ;
0000F08 ;
0000F0C ;
0000F10 ;
0000F14 ;
0000F18 ;
0000F1C ;
0000F20 ;
0000F24 ;
0000F28 ;
0000F2C ;
0000F30 ;
0000F34 ;
0000F38 ;
0000F3C ;
0000F40 ;
0000F44 ;
0000F48 ;
0000F4C ;
0000F50 ;
0000F54 ;
0000F58 ;
0000F5C ;
0000F60 ;
0000F64 ;
0000F68 ;
0000F6C ;
0000F70 ;
0000F74 ;
0000F78 ;
0000F7C ;
0000F80 ;
0000F84 ;
0000F88 ;
0000F8C ;
0000F90 ;
0000F94 ;
0000F98 ;
0000F9C ;
0000FA0 ;
0000FA4 ;
0000FA8 ;
0000FAC ;
0000FB0 ;
0000FB4 ;
0000FB8 ;
0000FBC ;
0000FC0 ;
0000FC4 ;
0000FC8 ;
0000FCC ;
0000FD0 ;
0000FD4 ;
0000FD8 ;
0000FDC ;
0000FE0 ;
0000FE4 ;
0000FE8 ;
0000FEC ;
0000FF0 ;
0000FF4 ;
0000FF8 ;
0000FFC ;

```

; 随机填充与Encoding of a p-code加密有关
; 与上面同理使用的结构体与80相同, 4个一组84~90

; 与上面同理使用的结构体与80相同, 4个一组94~A0

; 保存的都是struct_DisassemblyFunction结构
; 该结构体保存数据分别是: 1、修复重定位地址(push XXXX<-push 0xFACE0002) 2、edi (UHcontext<-r
; 主要是保存push XX(寄存器环境)对应的数字,后面会进行乱序操作

正向就是解密,取反就是加密了
0x4=add ---->加密用sub

0x5=xor	---->xor不变
0x34=sub	---->解密用add
0x43=rol	循环左移
0x44=ror	逻辑右移
0x5C=not	单操作数
0x29=inc	单操作数
0x2A=dec	单操作数
0x5D=neg	单操作数
0x31=bswap	单操作数