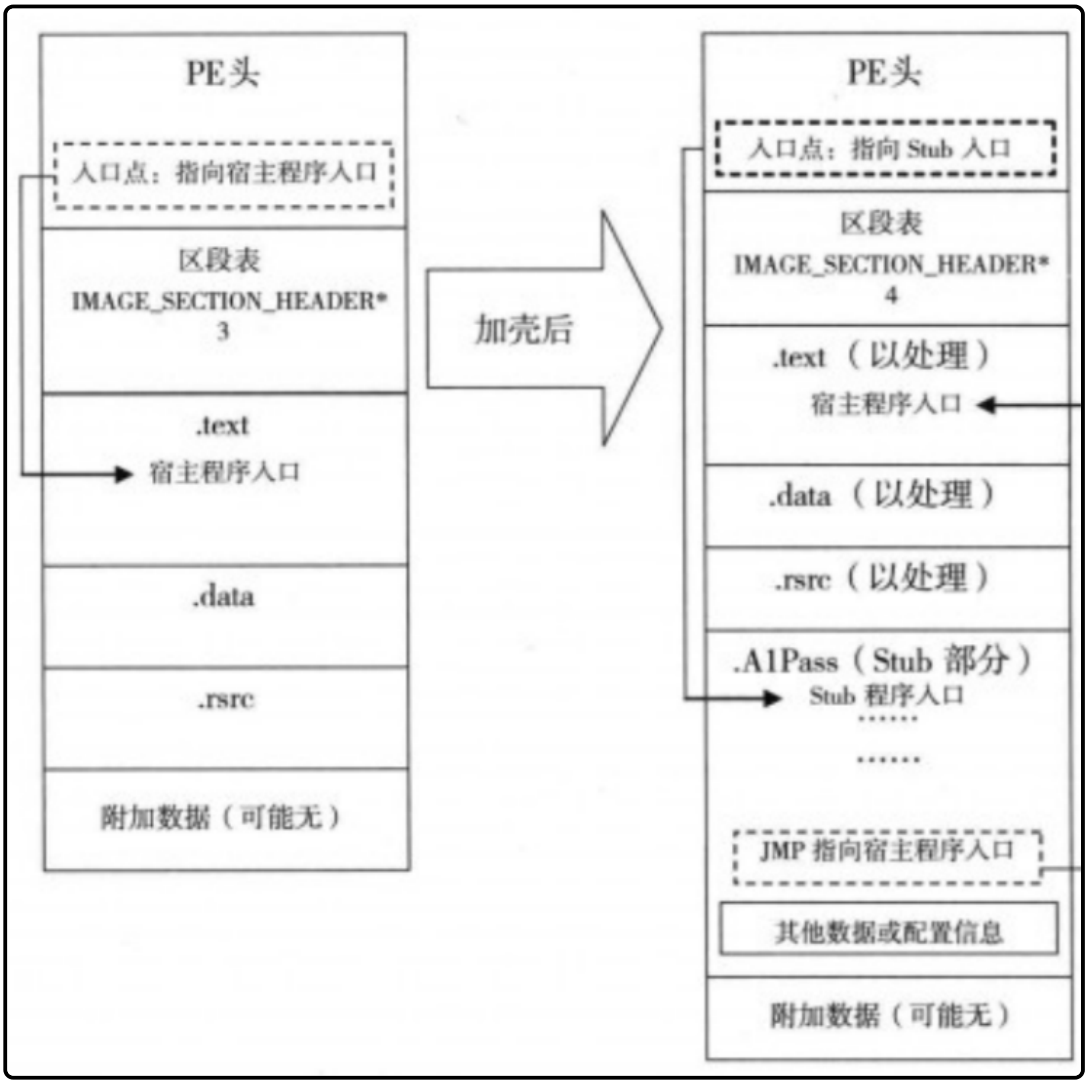
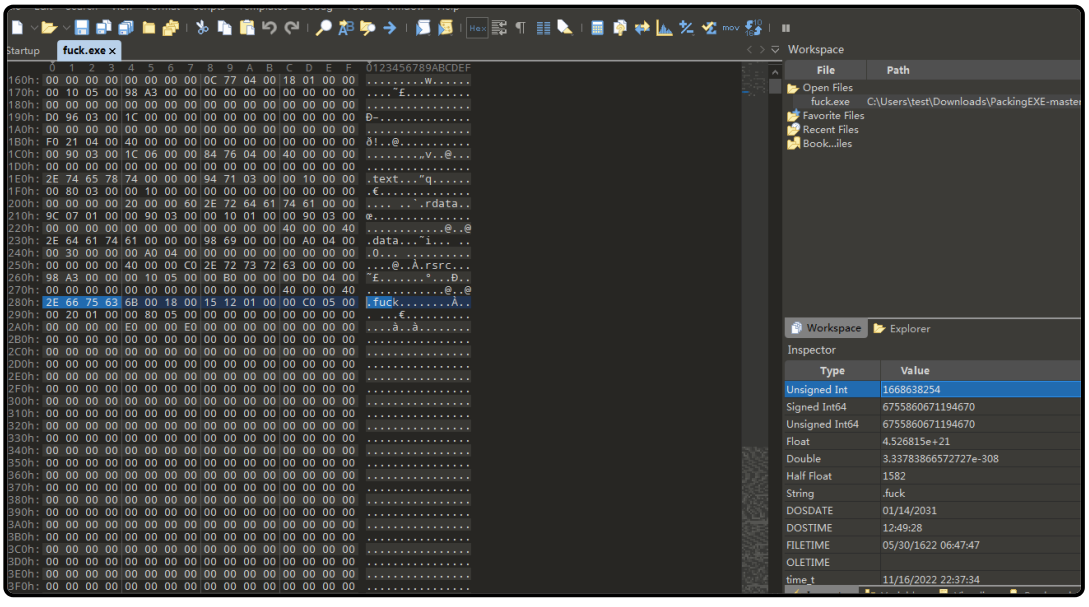
完整高级免杀壳开发原理（一）

用010editor手工加壳了解下原理：





加壳处理流程

1读取加壳文件，外壳DLL

2:选择加壳文件需要压缩的地方，对于资源段选择不压缩，其他区段都进行压缩。

3:重新构造区段表，分别有这么几个区段 .OldDat(原始压缩的数据) .Shell(外壳DLL代码) .tls(用来支持加壳tls程序) .CRT(用来支持加壳tls程序) .reloc(外壳DLL重定位信息) .rcsc(资源如果有的话)

4:利用aPLib进行压缩，将压缩之后的数据复制到目标文件.OldDat区段缓冲区。

5:对于外壳DLL进行重定位，资源数据修复
6:设置导出变量的数据，在外壳DLL中将使用到的变量
7:写入文件

完整高级免杀壳手工分析（二）

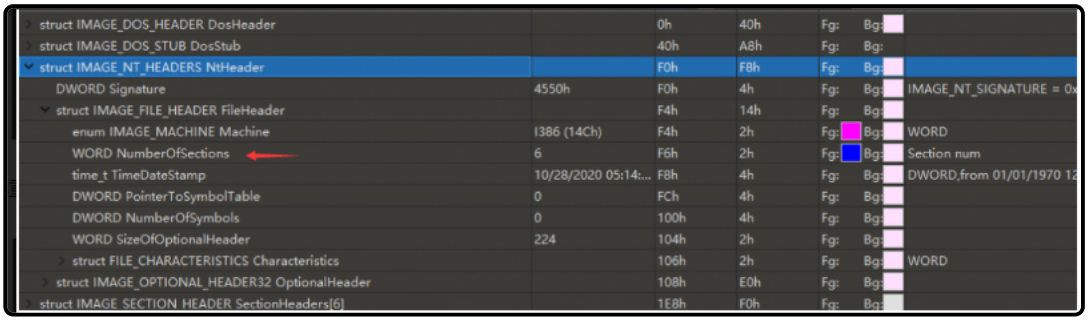
即我们向PE文件添加一个区段并将其设置为入口点，这样PE文件最开始执行的命令就是我们添加的区段也就是壳的指令，壳对加密区进行解密，对压缩区进行解压，将原本的EXE文件还原出来，然后跳转至原程序入口，程序照常运行。

首先生成一个打印hello的exe文件。

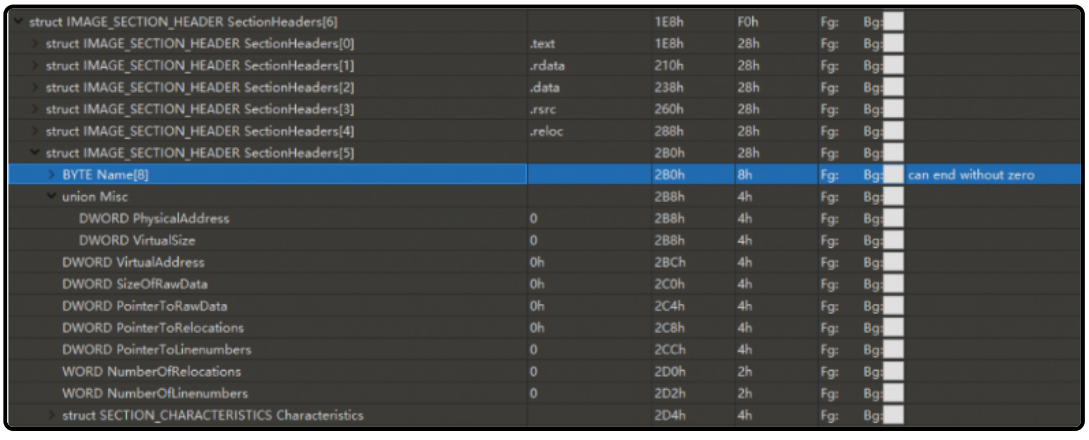
```c
#include <stdio.h>

int main() {
printf("hello");
}
```

我们目前要干的事情是:以手动的形式向PE文件添加一个壳部分并设为程序入口，并使其能跳转回原入口。 那就来吧

用010editor打开我们的exe文件，启用exe模板分析。 我们首先修改其文件头numverofsection属性，这个属性用来定义当前PE文件存在多少个区段,因为我们要添加一个壳区段，所以我们将其加1变成6

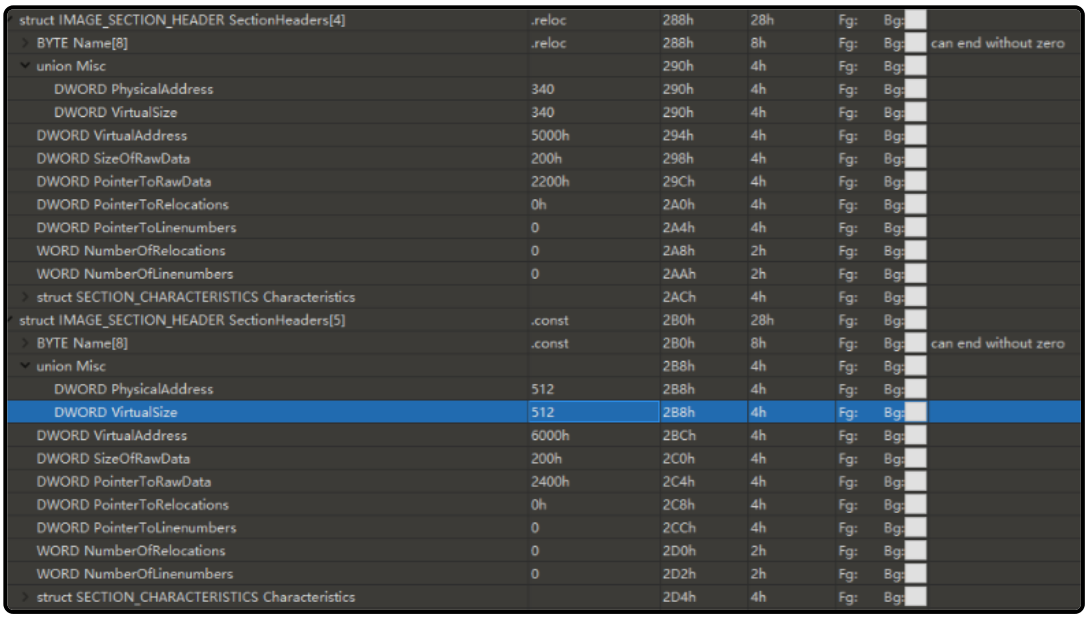


在我们重载模板后我们就会在区段表发现多出来一个空的区段表



从上到下各个比较重要字段的意思是 \1. Name 表示该区段的名字 2.VirtualSize 表示在内存中的大小(一般内存对齐为0x1000) 3.virtualaddress 虚拟地址 即上一个区段的VirtualAddress + 上一个区段经内存对齐粒度对齐后的大小 4.sizeofdata 表示在文件中的大小（一般文件对齐为0x200) 5.pointertorawdata 文件的偏移 即 上一个区段的PointerToRawData + 上一个区段的SizeOfRawData

然后我们通过修改以上各值来定义一个新区段（壳区段)的属性



struct IMAGE_SECTION_HEADER SectionHeaders[4]	.reloc	288h	28h	Fg:	Bg:	
BYTE Name[8]	.reloc	288h	8h	Fg:	Bg:	can end without zero
union Misc		290h	4h	Fg:	Bg:	
DWORD PhysicalAddress	340	290h	4h	Fg:	Bg:	
DWORD VirtualSize	340	290h	4h	Fg:	Bg:	
DWORD VirtualAddress	5000h	294h	4h	Fg:	Bg:	
DWORD SizeOfRawData	200h	298h	4h	Fg:	Bg:	
DWORD PointerToRawData	2200h	29Ch	4h	Fg:	Bg:	
DWORD PointerToRelocations	0h	2A0h	4h	Fg:	Bg:	
DWORD PointerToLinenumbers	0	2A4h	4h	Fg:	Bg:	
WORD NumberOfRelocations	0	2A8h	2h	Fg:	Bg:	
WORD NumberOfLinenumbers	0	2AAh	2h	Fg:	Bg:	
struct SECTION_CHARACTERISTICS Characteristics		2ACh	4h	Fg:	Bg:	
struct IMAGE_SECTION_HEADER SectionHeaders[5]	.const	2B0h	28h	Fg:	Bg:	
BYTE Name[8]	.const	2B0h	8h	Fg:	Bg:	can end without zero
union Misc		2B8h	4h	Fg:	Bg:	
DWORD PhysicalAddress	512	2B8h	4h	Fg:	Bg:	
DWORD VirtualSize	512	2B8h	4h	Fg:	Bg:	
DWORD VirtualAddress	6000h	2BCh	4h	Fg:	Bg:	
DWORD SizeOfRawData	200h	2C0h	4h	Fg:	Bg:	
DWORD PointerToRawData	2400h	2C4h	4h	Fg:	Bg:	
DWORD PointerToRelocations	0h	2C8h	4h	Fg:	Bg:	
DWORD PointerToLinenumbers	0	2CCh	4h	Fg:	Bg:	
WORD NumberOfRelocations	0	2D0h	2h	Fg:	Bg:	
WORD NumberOfLinenumbers	0	2D2h	2h	Fg:	Bg:	
struct SECTION_CHARACTERISTICS Characteristics		2D4h	4h	Fg:	Bg:	

完整高级免杀壳源码开发（三）

```

流程部分：
#include <Windows.h>
//#include "stdafx.h"
#include <tchar.h>
#include <stdio.h>
#include "PE.h"

int main() {
    //打开被加壳文件
    char PePath[] = "C:\\Users\\test\\Downloads\\PackingEXE-master\\PackingEXE-master\\jiake\\Release\\Server.exe";
    DWORD PeSize;
    char* PeHmoudle = GetFileHmoudle(PePath,&PeSize);
    //加载stub
    StubInfo pstub = { 0 };
    LoadStub(&pstub);

    //加密代码段
    DWORD textRVA = GetSecByName(PeHmoudle, ".text")->VirtualAddress;
    DWORD textSize = GetSecByName(PeHmoudle, ".text")->Misc.VirtualSize;
    Encry(PeHmoudle,pstub);

    //添加新区段
    char SecName[] = ".fuck";
    char* PeNewHmoudle = AddSec(PeHmoudle, PeSize, SecName, GetSecByName(pstub.dllbase, ".text")->Misc.VirtualSize);

    //stub重定位修复
    FixStub(GetOptHeader(PeNewHmoudle)->ImageBase,
        (DWORD)pstub.dllbase,
        GetLastSec(PeNewHmoudle)->VirtualAddress,
        GetSecByName(pstub.dllbase,".text")->VirtualAddress);
    auto b = (DWORD*)GetProcAddress((HMODULE)pstub.dllbase, "OriginEntry");
    pstub.pStubConf->srcOep = GetOptHeader(PeNewHmoudle)->AddressOfEntryPoint;  //获取原入口点

    //stub移植
    memcpy(GetLastSec(PeNewHmoudle)->PointerToRawData+ PeNewHmoudle,
        GetSecByName(pstub.dllbase, ".text")->VirtualAddress+pstub.dllbase,
        GetSecByName(pstub.dllbase,".text")->Misc.VirtualSize);

    ////入口点修改
    GetOptHeader(PeNewHmoudle)->AddressOfEntryPoint =
```

```
        pstub.pfnStart-(DWORD)pstub.dllbase-GetSecByName(pstub.dllbase,".text")->VirtualAddress+GetLastSec(PeNewHmoudle)->VirtualAddress;
    auto a =pstub.pfnStart-(DWORD)pstub.dllbase-GetSecByName(pstub.dllbase,".text")->VirtualAddress+GetLastSec(PeNewHmoudle)->VirtualAddress;
    auto d =GetProcAddress((HMODULE)pstub.dllbase, "OriginEntry");

    //去随机基址
    GetOptHeader(PeNewHmoudle)->DllCharacteristics &= (~0x40);

    //保存文件
    SaveFile("C:\\Users\\test\\Downloads\\PackingEXE-master\\PackingEXE-master\\jiake\\Release\\fuck.exe", PeNewHmoudle, PeSize);

    return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////


#include <Windows.h>

typedef struct _StubConf
{
    DWORD srcOep;       //入口点
    DWORD textScnRVA;   //代码段RVA
    DWORD textScnSize;  //代码段的大小
    DWORD key;          //解密密钥
}StubConf;

struct StubInfo
{
    char* dllbase;          //stub.dll的加载基址
    DWORD pfnStart;         //stub.dll(start)导出函数的地址
    StubConf* pStubConf;    //stub.dll(g_conf)导出全局变量的地址
};


//***********************
//PE信息获取函数簇
//time:2020/11/2
//***********************
PIMAGE_DOS_HEADER GetDosHeader(_In_ char* pBase) {
    return PIMAGE_DOS_HEADER(pBase);
}

PIMAGE_NT_HEADERS GetNtHeader(_In_ char* pBase) {
return PIMAGE_NT_HEADERS(GetDosHeader(pBase)->e_lfanew+(SIZE_T)pBase);
}

PIMAGE_FILE_HEADER GetFileHeader(_In_ char* pBase) {
    return &(GetNtHeader(pBase)->FileHeader);
}

PIMAGE_OPTIONAL_HEADER32 GetOptHeader(_In_ char* pBase) {
    return &(GetNtHeader(pBase)->OptionalHeader);
}

PIMAGE_SECTION_HEADER GetLastSec(_In_ char* pBase) {
    DWORD SecNum = GetFileHeader(pBase)->NumberOfSections;
    PIMAGE_SECTION_HEADER FirstSec = IMAGE_FIRST_SECTION(GetNtHeader(pBase));
    PIMAGE_SECTION_HEADER LastSec = FirstSec + SecNum - 1;
    return LastSec;
}

PIMAGE_SECTION_HEADER GetSecByName(_In_ char* pBase,_In_ const char* name) {
    DWORD Secnum = GetFileHeader(pBase)->NumberOfSections;
    PIMAGE_SECTION_HEADER Section = IMAGE_FIRST_SECTION(GetNtHeader(pBase));
    char buf[10] = { 0 };
    for (DWORD i = 0; i < Secnum; i++) {
        memcpy_s(buf, 8, (char*)Section[i].Name, 8);
        if (!strcmp(buf, name)) {
                return Section + i;
        }
    }
```

```
    return nullptr;
}

//**********************
//打开文件返回句柄
//time:2020/11/2
//**********************
char* GetFileHmoudle(_In_ const char* path,_Out_opt_ DWORD* nFileSize) {
    //打开一个文件并获得文件句柄
    HANDLE hFile = CreateFileA(path,
        GENERIC_READ,
        FILE_SHARE_READ,
        NULL,
        OPEN_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
    //获得文件大小
    DWORD FileSize = GetFileSize(hFile, NULL);
    //返回文件大小到变量nFileSize
    if(nFileSize)
        *nFileSize = FileSize;
    //申请一片大小为FileSize的内存并将指针置于首位
    char* pFileBuf = new CHAR[FileSize]{ 0 };
    //给刚刚申请的内存读入数据
    DWORD dwRead;
    ReadFile(hFile, pFileBuf, FileSize, &dwRead, NULL);
    CloseHandle(hFile);
    return pFileBuf;
}

//****************
//对齐处理
//time:2020/11/5
//****************
int AlignMent(_In_ int size, _In_ int alignment) {
    return (size) % (alignment)==0 ? (size) : ((size) / alignment+1) * (alignment);
}

//**********************
//增添区段
//time:2020/11/6
//**********************
char* AddSec(_In_ char*& hpe, _In_ DWORD& filesize, _In_ const char* secname, _In_
const int secsize) {
    GetFileHeader(hpe)->NumberOfSections++;
    PIMAGE_SECTION_HEADER pesec = GetLastSec(hpe);
    //设置区段表属性
    memcpy(pesec->Name, secname, 8);
    pesec->Misc.VirtualSize = secsize;
    pesec->VirtualAddress = (pesec - 1)->VirtualAddress + AlignMent((pesec - 1)-
>SizeOfRawData,GetOptHeader(hpe)->SectionAlignment);
    pesec->SizeOfRawData = AlignMent(secsize, GetOptHeader(hpe)->FileAlignment);
    pesec->PointerToRawData = AlignMent(filesize,GetOptHeader(hpe)->FileAlignment);
    pesec->Characteristics = 0xE00000E0;
    //设置OPT头映像大小
    GetOptHeader(hpe)->SizeOfImage = pesec->VirtualAddress + pesec->SizeOfRawData;
    //扩充文件数据
    int newSize = pesec->PointerToRawData + pesec->SizeOfRawData;
    char* nhpe = new char [newSize] {0};
    //向新缓冲区录入数据
    memcpy(nhpe, hpe, filesize);
    //缓存区更替
    delete hpe;
    filesize = newSize;
    return nhpe;
}

//******************
//保存文件
//time:2020/11/6
//******************
void SaveFile(_In_ const char* path, _In_ const char* data, _In_ int FileSize) {
    HANDLE hFile = CreateFileA(
        path,
```

```
        GENERIC_WRITE,
        FILE_SHARE_READ,
        NULL,
        CREATE_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL
    );
    DWORD Buf = 0;
    WriteFile(hFile, data, FileSize, &Buf,NULL);
    CloseHandle(hFile);
}

//******************
//加载stub
//time:
//******************
//void StubLoad (_In_ StubInfo* pStub) {
//
//}


void FixStub(DWORD targetDllbase, DWORD stubDllbase,DWORD targetNewScnRva,DWORD stubTextRva )
{
    //找到stub.dll的重定位表
    DWORD dwRelRva = GetOptHeader((char*)stubDllbase)->DataDirectory[5].VirtualAddress;
    IMAGE_BASE_RELOCATION* pRel = (IMAGE_BASE_RELOCATION*)(dwRelRva + stubDllbase);

    //遍历重定位表
    while (pRel->SizeOfBlock)
    {
        struct TypeOffset
        {
            WORD offset : 12;
            WORD type : 4;

        };
        TypeOffset* pTypeOffset = (TypeOffset*)(pRel + 1);
        DWORD dwCount = (pRel->SizeOfBlock - 8) / 2;    //需要重定位的数量
        for (int i = 0; i < dwCount; i++)
        {
            if (pTypeOffset[i].type != 3)
            {
                continue;
            }
            //需要重定位的地址
            DWORD* pFixAddr = (DWORD*)(pRel->VirtualAddress + pTypeOffset[i].offset + stubDllbase);

            DWORD dwOld;
            //修改属性为可写
            VirtualProtect(pFixAddr, 4, PAGE_READWRITE, &dwOld);
            //去掉dll当前加载基址
            *pFixAddr -= stubDllbase;
            //去掉默认的段首RVA
            *pFixAddr -= stubTextRva;
            //换上目标文件的加载基址
            *pFixAddr += targetDllbase;
            //加上新区段的段首RVA
            *pFixAddr += targetNewScnRva;
            //把属性修改回去
            VirtualProtect(pFixAddr, 4, dwOld, &dwOld);
        }
        //切换到下一个重定位块
        pRel = (IMAGE_BASE_RELOCATION*)((DWORD)pRel + pRel->SizeOfBlock);
    }

}
//******************
//加密代码段
//time:
//******************
void Encry(_In_ char* hpe,_In_ StubInfo pstub)
{
```

```
    //获取代码段首地址
    BYTE* TargetText = GetSecByName(hpe, ".text")->PointerToRawData + (BYTE*)hpe;
    //获取代码段大小
    DWORD TargetTextSize = GetSecByName(hpe, ".text")->Misc.VirtualSize;
    //加密代码段
    for (int i = 0; i < TargetTextSize; i++) {
        TargetText[i] ^= 0x15;
    }
    pstub.pStubConf->textScnRVA = GetSecByName(hpe, ".text")->VirtualAddress;
    pstub.pStubConf->textScnSize = TargetTextSize;
    pstub.pStubConf->key = 0x15;
}

void LoadStub(_In_ StubInfo* pstub)
{
    pstub->dllbase = (char*)LoadLibraryEx(L"stubdll.dll", NULL,
DONT_RESOLVE_DLL_REFERENCES);
    pstub->pfnStart = (DWORD)GetProcAddress((HMODULE)pstub->dllbase, "Start");
    pstub->pStubConf = (StubConf*)GetProcAddress((HMODULE)pstub->dllbase, "g_conf");
}
```

完整高级免杀壳源码开发（四）

```
stubdll shellcode模块部分:



// 合并.data到.text段
#pragma comment(linker,"/merge:.data=.text")
// 合并.rdata到.text段
#pragma comment(linker,"/merge:.rdata=.text")
// 将.text改成可读可写可执行
#pragma comment(linker, "/section:.text,RWE")

#include <Windows.h>
typedef struct _StubConf
{
    DWORD srcOep;       //入口点
    DWORD textScnRVA;   //代码段RVA
    DWORD textScnSize;  //代码段的大小
    DWORD key;          //解密密钥
}StubConf;

//导出一个全局变量
extern "C" __declspec(dllexport)StubConf g_conf = { 0 };

//定义函数指针和变量
typedef void* (WINAPI* FnGetProcAddress)(HMODULE, const char*);
FnGetProcAddress MyGetProcAddress;

typedef void* (WINAPI* FnLoadLibraryA)(char*);
FnLoadLibraryA MyLoadLibraryA;

typedef void* (WINAPI* FnVirtualProtect)(LPVOID, SIZE_T, DWORD, PDWORD);
FnVirtualProtect MyVirtualProtect;



void Decrypt()
{
    unsigned char* pText = (unsigned char*)g_conf.textScnRVA + 0x400000;
    //修改代码段的属性
    DWORD old = 0;
    MyVirtualProtect(pText, g_conf.textScnSize, PAGE_READWRITE, &old);
    //解密代码段
    for (DWORD i = 0; i < g_conf.textScnSize; i++)
    {
        pText[i] ^= g_conf.key;
    }
```

```
    //把属性修改回去
    MyVirtualProtect(pText, g_conf.textScnSize, old, &old);

}


void GetApis()
{
    HMODULE hKernel32;

    _asm
    {
        pushad;
        ; //获取kernel32.dll的加载基址;
        ;// 1. 找到PEB的首地址;
        mov eax, fs: [0x30] ;
        mov eax, [eax + 0ch];
        mov eax, [eax + 0ch];
        mov eax, [eax];
        mov eax, [eax];
        mov eax, [eax + 018h];
        mov hKernel32, eax;
        mov ebx, [eax + 03ch];
        add ebx, eax;
        add ebx, 078h;
        mov ebx, [ebx];
        add ebx, eax;
        lea ecx, [ebx + 020h];
        mov ecx, [ecx]; // ecx => 名称表的首地址(rva);
        add ecx, eax; // ecx => 名称表的首地址(va);
        xor edx, edx; // 作为index来使用.
    _WHILE:;
        mov esi, [ecx + edx * 4];
        lea esi, [esi + eax];
        cmp dword ptr[esi], 050746547h; 47657450 726F6341 64647265 7373;
        jne _LOOP;
        cmp dword ptr[esi + 4], 041636f72h;
        jne _LOOP;
        cmp dword ptr[esi + 8], 065726464h;
        jne _LOOP;
        cmp word  ptr[esi + 0ch], 07373h;
        jne _LOOP;
        mov edi, [ebx + 024h];
        add edi, eax;

        mov di, [edi + edx * 2];
        and edi, 0FFFFh;
        mov edx, [ebx + 01ch];
        add edx, eax;
        mov edi, [edx + edi * 4];
        add edi, eax; ;
        mov MyGetProcAddress, edi;
        jmp _ENDWHILE;
    _LOOP:;
        inc edx; // ++index;
        jmp _WHILE;
    _ENDWHILE:;
        popad;
    }

    MyLoadLibraryA = (FnLoadLibraryA)MyGetProcAddress(hKernel32, "LoadLibrary");
    MyVirtualProtect = (FnVirtualProtect)MyGetProcAddress(hKernel32, "VirtualProtect");


}
extern "C" __declspec(dllexport) __declspec(naked)
void Start()
{
    //获取函数的API地址
    GetApis();
    //解密代码段
    Decrypt();
    //跳转到原始OEP
    __asm
```

```
    {
        mov eax, g_conf.srcOep;
        add eax, 0x400000
            jmp eax
    }
}

```