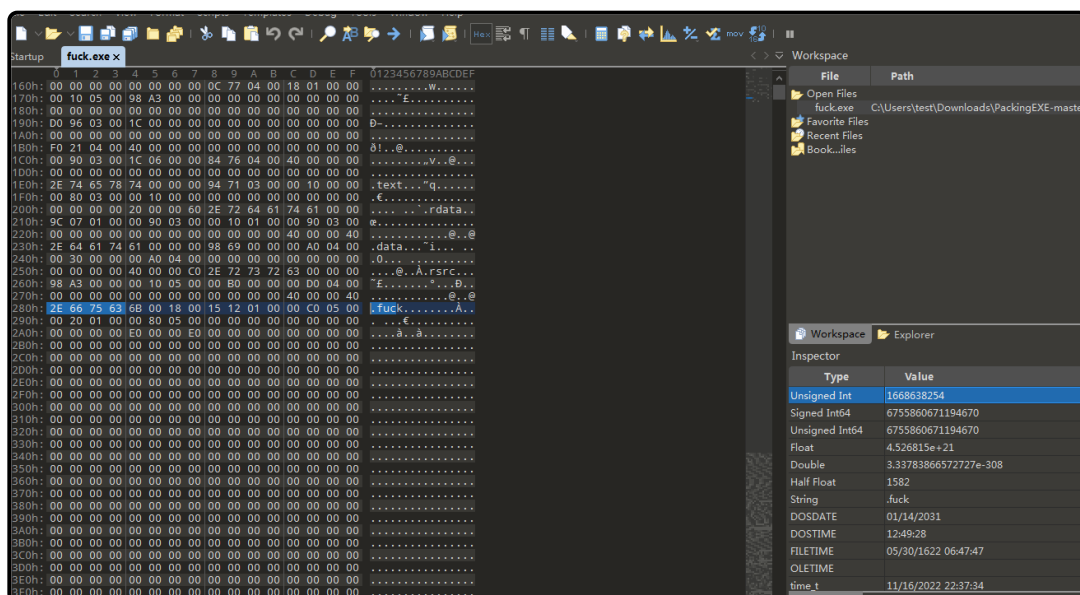
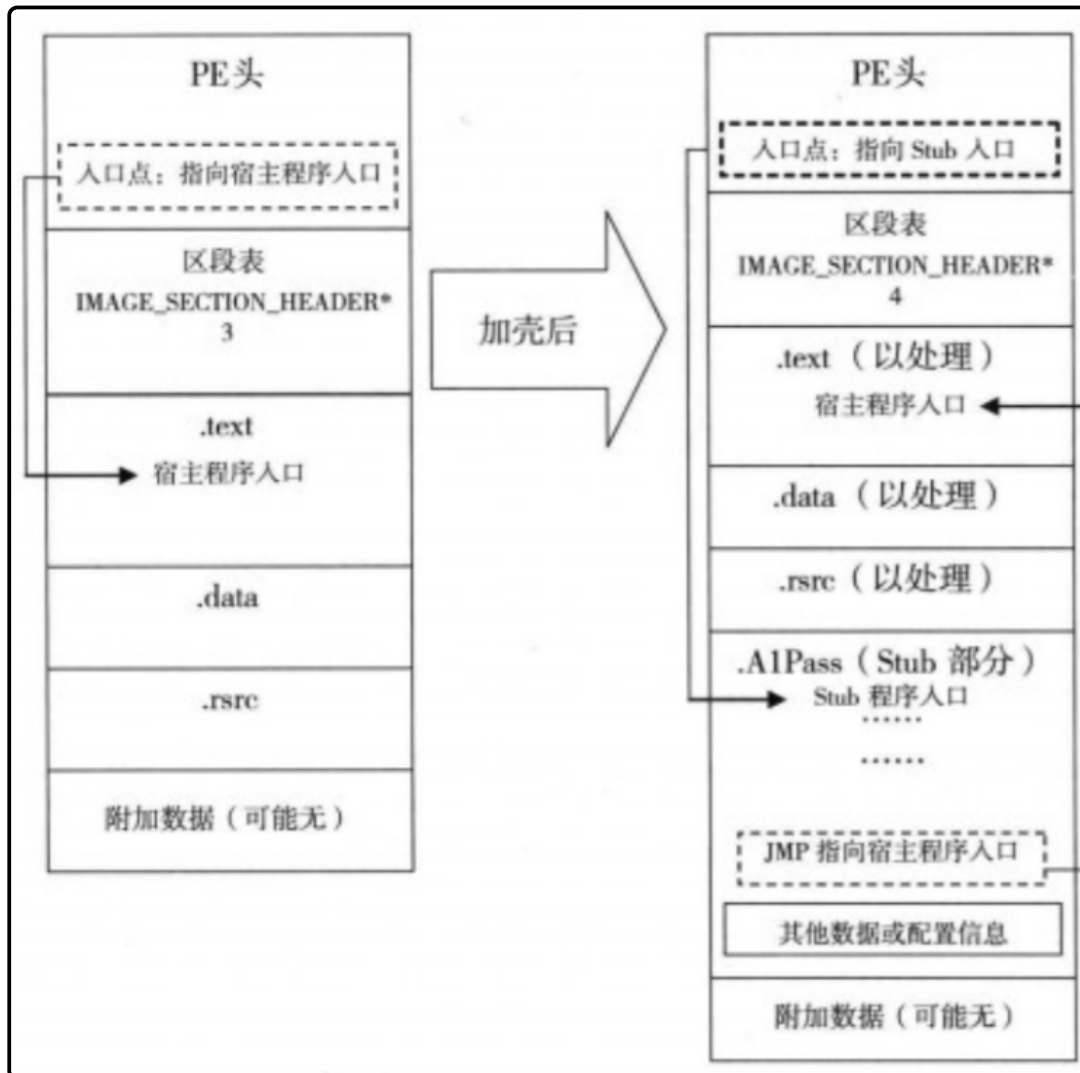


完整高级免杀壳开发原理（一）

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) .rsrc(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。

```
1 #include <stdio.h>
2
3 int main() {
4     printf("hello");
5 }
```

我们目前要干的事情是:以手动形式向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。那我们就来吧

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, 所以我们将加1变成6

struct IMAGE_DOS_HEADER DosHeader		0h	40h	Fg:	Bg:	
struct IMAGE_DOS_STUB DosStub		40h	A8h	Fg:	Bg:	
struct IMAGE_NT_HEADERS NtHeader		F0h	F8h	Fg:	Bg:	
DWORD Signature	4550h	F0h	4h	Fg:	Bg:	IMAGE_NT_SIGNATURE = 0x...
struct IMAGE_FILE_HEADER FileHeader		F4h	14h	Fg:	Bg:	
enum IMAGE_MACHINE Machine	1386 (14Ch)	F4h	2h	Fg:	Bg:	WORD
WORD NumberOfSections	6	F6h	2h	Fg:	Bg:	Section num
time_t TimeDateStamp	10/28/2020 05:14:...	F8h	4h	Fg:	Bg:	DWORD, from 01/01/1970 12:00:00
DWORD PointerToSymbolTable	0	FCh	4h	Fg:	Bg:	
DWORD NumberOfSymbols	0	100h	4h	Fg:	Bg:	
WORD SizeOfOptionalHeader	224	104h	2h	Fg:	Bg:	
struct FILE_CHARACTERISTICS Characteristics		106h	2h	Fg:	Bg:	WORD
struct IMAGE_OPTIONAL_HEADER32 OptionalHeader		108h	E0h	Fg:	Bg:	
struct IMAGE_SECTION_HEADER SectionHeaders[6]		1E8h	F0h	Fg:	Bg:	

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

struct IMAGE_SECTION_HEADER SectionHeaders[6]		1E8h	F0h	Fg:	Bg:	
struct IMAGE_SECTION_HEADER SectionHeaders[0]	.text	1E8h	28h	Fg:	Bg:	
struct IMAGE_SECTION_HEADER SectionHeaders[1]	.rdata	210h	28h	Fg:	Bg:	
struct IMAGE_SECTION_HEADER SectionHeaders[2]	.data	238h	28h	Fg:	Bg:	
struct IMAGE_SECTION_HEADER SectionHeaders[3]	.rsrc	260h	28h	Fg:	Bg:	
struct IMAGE_SECTION_HEADER SectionHeaders[4]	.reloc	288h	28h	Fg:	Bg:	
struct IMAGE_SECTION_HEADER SectionHeaders[5]		2B0h	28h	Fg:	Bg:	
BYTE Name[8]		2B0h	8h	Fg:	Bg:	can end without zero
union Misc		2B8h	4h	Fg:	Bg:	
DWORD PhysicalAddress	0	2B8h	4h	Fg:	Bg:	
DWORD VirtualSize	0	2B8h	4h	Fg:	Bg:	
DWORD VirtualAddress	0h	2BCh	4h	Fg:	Bg:	
DWORD SizeOfRawData	0h	2C0h	4h	Fg:	Bg:	
DWORD PointerToRawData	0h	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:	

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）

```

1
2
3 流程部分:
4 #include <Windows.h>
5 // #include "stdafx.h"
6 #include <tchar.h>
7 #include <stdio.h>
8 #include "PE.h"
9
10 int main() {
11     // 打开被加壳文件
12     char PePath[] = "C:\\Users\\test\\Downloads\\PackingEXE-master\\PackingEXE-
master\\jiake\\Release\\Server.exe";
13     DWORD PeSize;
14     char* PeHmoudle = GetFileHmoudle(PePath,&PeSize);
15     // 加载stub
16     StubInfo pstub = { 0 };
17     LoadStub(&pstub);
18
19     // 加密代码段
20     DWORD textRVA = GetSecByName(PeHmoudle, ".text")->VirtualAddress;
21     DWORD textSize = GetSecByName(PeHmoudle, ".text")->Misc.VirtualSize;
22     Encry(PeHmoudle,pstub);
23
24     // 添加新区段
25     char SecName[] = ".fuck";
26     char* PeNewHmoudle = AddSec(PeHmoudle, PeSize, SecName, GetSecByName(pstub.dllbase,
".text")->Misc.VirtualSize);
27
28     // stub重定位修复
29     FixStub(GetOptHeader(PeNewHmoudle)->ImageBase,
30             (DWORD)pstub.dllbase,
31             GetLastSec(PeNewHmoudle)->VirtualAddress,
32             GetSecByName(pstub.dllbase, ".text")->VirtualAddress);
33     auto b = (DWORD*)GetProcAddress((HMODULE)pstub.dllbase, "OriginEntry");
34     pstub.pStubConf->srcOep = GetOptHeader(PeNewHmoudle)->AddressOfEntryPoint; // 获取原
入口点
35
36     // stub移植
37     memcpy(GetLastSec(PeNewHmoudle)->PointerToRawData+ PeNewHmoudle,
38            GetSecByName(pstub.dllbase, ".text")->VirtualAddress+pstub.dllbase,
39            GetSecByName(pstub.dllbase, ".text")->Misc.VirtualSize);
40
41     /// 入口点修改
42     GetOptHeader(PeNewHmoudle)->AddressOfEntryPoint =

```

```

43     pstub.pfnStart-(DWORD)pstub.dllbase-GetSecByName(pstub.dllbase, ".text")-
>VirtualAddress+GetLastSec(PeNewHmoudle)->VirtualAddress;
44     auto a =pstub.pfnStart-(DWORD)pstub.dllbase-GetSecByName(pstub.dllbase, ".text")-
>VirtualAddress+GetLastSec(PeNewHmoudle)->VirtualAddress;
45     auto d =GetProcAddress((HMODULE)pstub.dllbase, "OriginEntry");
46
47     //去随机基址
48     GetOptHeader(PeNewHmoudle)->DllCharacteristics &= (~0x40);
49
50     //保存文件
51     SaveFile("C:\\Users\\test\\Downloads\\PackingEXE-master\\PackingEXE-
master\\jiake\\Release\\fuck.exe", PeNewHmoudle, PeSize);
52
53     return 0;
54 }
55 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
56
57
58 #include <Windows.h>
59
60 typedef struct _StubConf
61 {
62     DWORD srcOep;           //入口点
63     DWORD textScnRVA;       //代码段RVA
64     DWORD textScnSize;      //代码段的大小
65     DWORD key;              //解密密钥
66 }StubConf;
67
68 struct StubInfo
69 {
70     char* dllbase;          //stub.dll的加载基址
71     DWORD pfnStart;         //stub.dll(start)导出函数的地址
72     StubConf* pStubConf;    //stub.dll(g_conf)导出全局变量的地址
73 };
74
75
76 //*****
77 //PE信息获取函数簇
78 //time:2020/11/2
79 //*****
80 PIMAGE_DOS_HEADER GetDosHeader(_In_ char* pBase) {
81     return PIMAGE_DOS_HEADER(pBase);
82 }
83
84 PIMAGE_NT_HEADERS GetNtHeader(_In_ char* pBase) {
85     return PIMAGE_NT_HEADERS(GetDosHeader(pBase)->e_lfanew+(SIZE_T)pBase);
86 }
87
88 PIMAGE_FILE_HEADER GetFileHeader(_In_ char* pBase) {
89     return &(GetNtHeader(pBase)->FileHeader);
90 }
91
92 PIMAGE_OPTIONAL_HEADER32 GetOptHeader(_In_ char* pBase) {
93     return &(GetNtHeader(pBase)->OptionalHeader);
94 }
95
96 PIMAGE_SECTION_HEADER GetLastSec(_In_ char* pBase) {
97     DWORD SecNum = GetFileHeader(pBase)->NumberOfSections;
98     PIMAGE_SECTION_HEADER FirstSec = IMAGE_FIRST_SECTION(GetNtHeader(pBase));
99     PIMAGE_SECTION_HEADER LastSec = FirstSec + SecNum - 1;
100     return LastSec;
101 }
102
103 PIMAGE_SECTION_HEADER GetSecByName(_In_ char* pBase, _In_ const char* name) {
104     DWORD Secnum = GetFileHeader(pBase)->NumberOfSections;
105     PIMAGE_SECTION_HEADER Section = IMAGE_FIRST_SECTION(GetNtHeader(pBase));
106     char buf[10] = { 0 };
107     for (DWORD i = 0; i < Secnum; i++) {
108         memcpy_s(buf, 8, (char*)Section[i].Name, 8);
109         if (!strcmp(buf, name)) {
110             return Section + i;
111         }
112     }

```

```

113     return nullptr;
114 }
115
116 //*****
117 //打开文件返回句柄
118 //time:2020/11/2
119 //*****
120 char* GetFileHmoudle(_In_ const char* path,_Out_opt_ DWORD* nFileSize) {
121     //打开一个文件并获得文件句柄
122     HANDLE hFile = CreateFileA(path,
123         GENERIC_READ,
124         FILE_SHARE_READ,
125         NULL,
126         OPEN_ALWAYS,
127         FILE_ATTRIBUTE_NORMAL,
128         NULL);
129     //获得文件大小
130     DWORD FileSize = GetFileSize(hFile, NULL);
131     //返回文件大小到变量nFileSize
132     if(nFileSize)
133         *nFileSize = FileSize;
134     //申请一片大小为FileSize的内存并将指针置于首位
135     char* pFileBuf = new CHAR[FileSize]{ 0 };
136     //给刚刚申请的内存读入数据
137     DWORD dwRead;
138     ReadFile(hFile, pFileBuf, FileSize, &dwRead, NULL);
139     CloseHandle(hFile);
140     return pFileBuf;
141 }
142
143 //*****
144 //对齐处理
145 //time:2020/11/5
146 //*****
147 int AlignMent(_In_ int size, _In_ int alignment) {
148     return (size) % (alignment)==0 ? (size) : ((size) / alignment+1) * (alignment);
149 }
150
151 //*****
152 //增添区段
153 //time:2020/11/6
154 //*****
155 char* AddSec(_In_ char*& hpe, _In_ DWORD& filesize, _In_ const char* secname, _In_
const int secsize) {
156     GetFileHeader(hpe)->NumberOfSections++;
157     PIMAGE_SECTION_HEADER pesec = GetLastSec(hpe);
158     //设置区段表属性
159     memcpy(pesec->Name, secname, 8);
160     pesec->Misc.VirtualSize = secsize;
161     pesec->VirtualAddress = (pesec - 1)->VirtualAddress + AlignMent((pesec - 1)-
>SizeOfRawData,GetOptHeader(hpe)->SectionAlignment);
162     pesec->SizeOfRawData = AlignMent(secsize, GetOptHeader(hpe)->FileAlignment);
163     pesec->PointerToRawData = AlignMent(filesize,GetOptHeader(hpe)->FileAlignment);
164     pesec->Characteristics = 0xE0000E0;
165     //设置OPT头映像大小
166     GetOptHeader(hpe)->SizeOfImage = pesec->VirtualAddress + pesec->SizeOfRawData;
167     //扩充文件数据
168     int newSize = pesec->PointerToRawData + pesec->SizeOfRawData;
169     char* nhpe = new char [newSize] {0};
170     //向新缓冲区录入数据
171     memcpy(nhpe, hpe, filesize);
172     //缓存区更替
173     delete hpe;
174     filesize = newSize;
175     return nhpe;
176 }
177
178 //*****
179 //保存文件
180 //time:2020/11/6
181 //*****
182 void SaveFile(_In_ const char* path, _In_ const char* data, _In_ int FileSize) {
183     HANDLE hFile = CreateFileA(
184         path,

```

```

185         GENERIC_WRITE,
186         FILE_SHARE_READ,
187         NULL,
188         CREATE_ALWAYS,
189         FILE_ATTRIBUTE_NORMAL,
190         NULL
191     );
192     DWORD Buf = 0;
193     WriteFile(hFile, data, FileSize, &Buf, NULL);
194     CloseHandle(hFile);
195 }
196
197 //*****
198 //加载stub
199 //time:
200 //*****
201 //void StubLoad (_In_ StubInfo* pStub) {
202 //
203 //}
204
205
206 void FixStub(DWORD targetDllbase, DWORD stubDllbase, DWORD targetNewScnRva, DWORD
stubTextRva )
207 {
208     //找到stub.dll的重定位表
209     DWORD dwRelRva = GetOptHeader((char*)stubDllbase)->DataDirectory[5].VirtualAddress;
210     IMAGE_BASE_RELOCATION* pRel = (IMAGE_BASE_RELOCATION*)(dwRelRva + stubDllbase);
211
212     //遍历重定位表
213     while (pRel->SizeOfBlock)
214     {
215         struct TypeOffset
216         {
217             WORD offset : 12;
218             WORD type : 4;
219
220         };
221         TypeOffset* pTypeOffset = (TypeOffset*)(pRel + 1);
222         DWORD dwCount = (pRel->SizeOfBlock - 8) / 2;    //需要重定位的数量
223         for (int i = 0; i < dwCount; i++)
224         {
225             if (pTypeOffset[i].type != 3)
226             {
227                 continue;
228             }
229             //需要重定位的地址
230             DWORD* pFixAddr = (DWORD*)(pRel->VirtualAddress + pTypeOffset[i].offset +
stubDllbase);
231
232             DWORD dwOld;
233             //修改属性为可写
234             VirtualProtect(pFixAddr, 4, PAGE_READWRITE, &dwOld);
235             //去掉dll当前加载基址
236             *pFixAddr -= stubDllbase;
237             //去掉默认的段首RVA
238             *pFixAddr -= stubTextRva;
239             //换上目标文件的加载基址
240             *pFixAddr += targetDllbase;
241             //加上新区段的段首RVA
242             *pFixAddr += targetNewScnRva;
243             //把属性修改回去
244             VirtualProtect(pFixAddr, 4, dwOld, &dwOld);
245         }
246         //切换到下一个重定位块
247         pRel = (IMAGE_BASE_RELOCATION*)((DWORD)pRel + pRel->SizeOfBlock);
248     }
249
250 }
251 //*****
252 //加密代码段
253 //time:
254 //*****
255 void Encry(_In_ char* hpe, _In_ StubInfo pstub)
256 {

```

```

257 //获取代码段首地址
258 BYTE* TargetText = GetSecByName(hpe, ".text")->PointerToRawData + (BYTE*)hpe;
259 //获取代码段大小
260 DWORD TargetTextSize = GetSecByName(hpe, ".text")->Misc.VirtualSize;
261 //加密代码段
262 for (int i = 0; i < TargetTextSize; i++) {
263     TargetText[i] ^= 0x15;
264 }
265 pstub.pStubConf->textScnRVA = GetSecByName(hpe, ".text")->VirtualAddress;
266 pstub.pStubConf->textScnSize = TargetTextSize;
267 pstub.pStubConf->key = 0x15;
268 }
269
270 void LoadStub(_In_ StubInfo* pstub)
271 {
272     pstub->dllbase = (char*)LoadLibraryEx(L"stubdll.dll", NULL,
DONT_RESOLVE_DLL_REFERENCES);
273     pstub->pfnStart = (DWORD)GetProcAddress((HMODULE)pstub->dllbase, "Start");
274     pstub->pStubConf = (StubConf*)GetProcAddress((HMODULE)pstub->dllbase, "g_conf");
275 }

```

完整高级免杀壳源码开发（四）

```

1 stubdll shellcode模块部分:
2
3
4
5 // 合并.data到.text段
6 #pragma comment(linker, "/merge:.data=.text")
7 // 合并.rdata到.text段
8 #pragma comment(linker, "/merge:.rdata=.text")
9 // 将.text改成可读可写可执行
10 #pragma comment(linker, "/section:.text,RWE")
11
12 #include <Windows.h>
13 typedef struct _StubConf
14 {
15     DWORD srcOep;        //入口点
16     DWORD textScnRVA;    //代码段RVA
17     DWORD textScnSize;   //代码段的大小
18     DWORD key;           //解密密钥
19 }StubConf;
20
21 //导出一个全局变量
22 extern "C" __declspec(dllexport)StubConf g_conf = { 0 };
23
24 //定义函数指针和变量
25 typedef void* (WINAPI* FnGetProcAddress)(HMODULE, const char*);
26 FnGetProcAddress MyGetProcAddress;
27
28 typedef void* (WINAPI* FnLoadLibraryA)(char*);
29 FnLoadLibraryA MyLoadLibraryA;
30
31 typedef void* (WINAPI* FnVirtualProtect)(LPVOID, SIZE_T, DWORD, PDWORD);
32 FnVirtualProtect MyVirtualProtect;
33
34
35 void Decrypt()
36 {
37     unsigned char* pText = (unsigned char*)g_conf.textScnRVA + 0x400000;
38     //修改代码段的属性
39     DWORD old = 0;
40     MyVirtualProtect(pText, g_conf.textScnSize, PAGE_READWRITE, &old);
41     //解密代码段
42     for (DWORD i = 0; i < g_conf.textScnSize; i++)
43     {
44         pText[i] ^= g_conf.key;
45     }

```

```

46 //把属性修改回去
47 MyVirtualProtect(pText, g_conf.textScnSize, old, &old);
48
49 }
50
51
52 void GetApis()
53 {
54     HMODULE hKernel32;
55
56     __asm
57     {
58         pushad;
59         ; //获取kernel32.dll的加载基址;
60         ;// 1. 找到PEB的首地址;
61         mov eax, fs: [0x30] ;
62         mov eax, [eax + 0ch];
63         mov eax, [eax + 0ch];
64         mov eax, [eax];
65         mov eax, [eax];
66         mov eax, [eax + 018h];
67         mov hKernel32, eax;
68         mov ebx, [eax + 03ch];
69         add ebx, eax;
70         add ebx, 078h;
71         mov ebx, [ebx];
72         add ebx, eax;
73         lea ecx, [ebx + 020h];
74         mov ecx, [ecx]; // ecx => 名称表的首地址(rva);
75         add ecx, eax; // ecx => 名称表的首地址(va);
76         xor edx, edx; // 作为index来使用.
77     _WHILE;;
78         mov esi, [ecx + edx * 4];
79         lea esi, [esi + eax];
80         cmp dword ptr[esi], 050746547h; 47657450 726F6341 64647265 7373;
81         jne _LOOP;
82         cmp dword ptr[esi + 4], 041636f72h;
83         jne _LOOP;
84         cmp dword ptr[esi + 8], 065726464h;
85         jne _LOOP;
86         cmp word ptr[esi + 0ch], 07373h;
87         jne _LOOP;
88         mov edi, [ebx + 024h];
89         add edi, eax;
90
91         mov di, [edi + edx * 2];
92         and edi, 0FFFFh;
93         mov edx, [ebx + 01ch];
94         add edx, eax;
95         mov edi, [edx + edi * 4];
96         add edi, eax; ;
97         mov MyGetProcAddress, edi;
98         jmp _ENDWHILE;
99     _LOOP;;
100         inc edx; // ++index;
101         jmp _WHILE;
102     _ENDWHILE;;
103         popad;
104     }
105
106     MyLoadLibraryA = (FnLoadLibraryA)MyGetProcAddress(hKernel32, "LoadLibrary");
107     MyVirtualProtect = (FnVirtualProtect)MyGetProcAddress(hKernel32, "VirtualProtect");
108
109 }
110
111 extern "C" __declspec(dllexport) __declspec(naked)
112 void Start()
113 {
114     //获取函数的API地址
115     GetApis();
116     //解密代码段
117     Decrypt();
118     //跳转到原始OEP
119     __asm

```



```
120     {  
121         mov eax, g_conf.src0ep;  
122         add eax, 0x400000  
123         jmp eax  
124     }  
125 }  
126
```