> 周泽峻 20184228

1. 加密壳的主要机制：

通过修改原程序的执行文件结构，对源程序的各个块进行加密、修改源程序文件的输入表、隐藏原程序的入口点等，以保护程序免受逆向分析。

加壳程序执行过程：
（1）保存入口参数
（2）获取所需要的API地址
（3）解密原程序的各个区块的数据
（4）初始化程序的IAT表
（5）对重定位项进行处理
（6）跳转到程序的原入口点

2. 简述源码级混淆的主要方法：

(1) 标识符重命名：
把变量，函数，类的名字改写成无意义的名字，使得阅读的人无法根据名字猜测其用途。

(2) 等价表达式：
重写代码中的部分逻辑，将其变成功能上等价，但是更难理解的形式。比如将循环改成递归，精简中间变量等。

(3) 代码重排：
打乱原有代码格式。比如将多行代码挤到一行代码中。

(4) 花指令：
通过构造字节码插入程序的适当位置，使得反汇编器出错，产生无法反编译或者反编译出错的情况。

(5) 自解密：
通过对程序部分进行加密，在即将运行时代码进行自解密，然后执行解密之后的代码。

3. 简述SEH链结构：

(1) SEH链的节点是异常处理器 `_EXCEPTION_REGISTRATION_RECORD` 结构体，定义为：

```
typedef struct _EXCEPTION_REGISTRATION_RECORD{
    PEXCEPTION_REGISTRATION_RECORD Next;
    PEXCEPTION_DISPOSITION Handler;
}EXCEPTION_REGISTRATION_RECORD,*PEXCEPTION_REGISTRATION_RECORD;
```

其中，`Next` 用于指向下一个异常处理器，如果 `Next` 的值为 `FFFFFFFF`，则表示SEH链到此结束。

`Handler` 则指向异常处理函数。异常处理函数是一个回调函数，由系统调用。

(2) 异常处理函数有四个参数，这四个参数用来传递与异常相关的信息，包括异常类型、发生异常的代码地址、异常发生时CPU寄存器的状态等。结构体定义为：

```
EXCEPTION_DISPOSITION _except_handler{
    EXCEPTION_RECORD *pRecord,
    EXCEPTION_REGISTRATION_RECORD *pFrame,
    CONTEXT *pContext,
    PVOID pValue
};
```

(3) 异常处理函数的返回为 `EXCEPTION_DISPOSITION` 的枚举类型，用于告知系统异常处理完成后程序应如何继续运行。结构体定义为：

```
typedef enum _EXCEPTION_DISPOSITION{
   ExceptionContinueExecution =0, //继续执行异常代码
   ExceptionContinueSearch =1, //运行下一个异常处理器
   ExceptionNestedException =2, //在OS内部使用
   ExceptionCollidedUnwind =3 //在OS内部使用
}EXCEPTION_DISPOSITION;
```

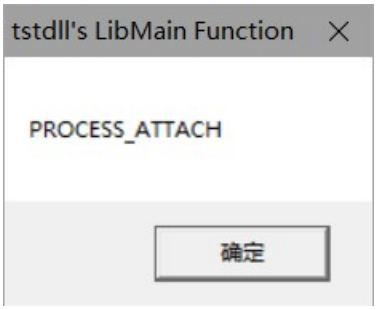
4.

代码：

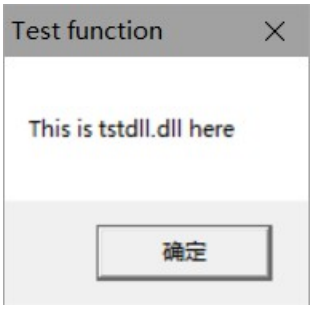
```
.386
.model flat, stdcall
option casemap :none

include kernel32.inc
include user32.inc
include masm32.inc

includelib kernel32.lib
includelib user32.lib
includelib masm32.lib

.data
    libName  db "tstdll.dll",0
    FuncName db "TestProc",0
    hLib dd 0
    szText db "HelloWorld",0
    szTitle db "title",0
    NULL = 0
    MB_OK = 0
.code
main PROC
    ; 装载动态链接库
    invoke LoadLibrary, ADDR libName
    mov hLib, eax

    ; 获取函数VA
    invoke GetProcAddress, hLib, ADDR FuncName
    call eax

    ; 释放动态链接库
    invoke FreeLibrary, hLib

    ; 静态调用MessageBox函数
    invoke MessageBox, NULL, ADDR szText, NULL, MB_OK

    invoke ExitProcess, 0
main ENDP
END main
```

执行结果：

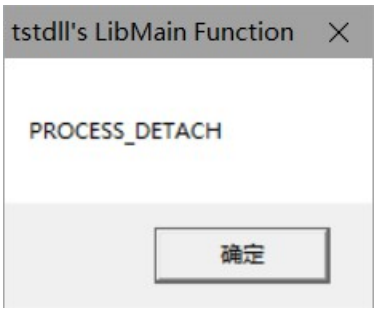
1. 加载动态链接库



2. 调用链接库的函数



3. 释放动态链接库



OD分析：

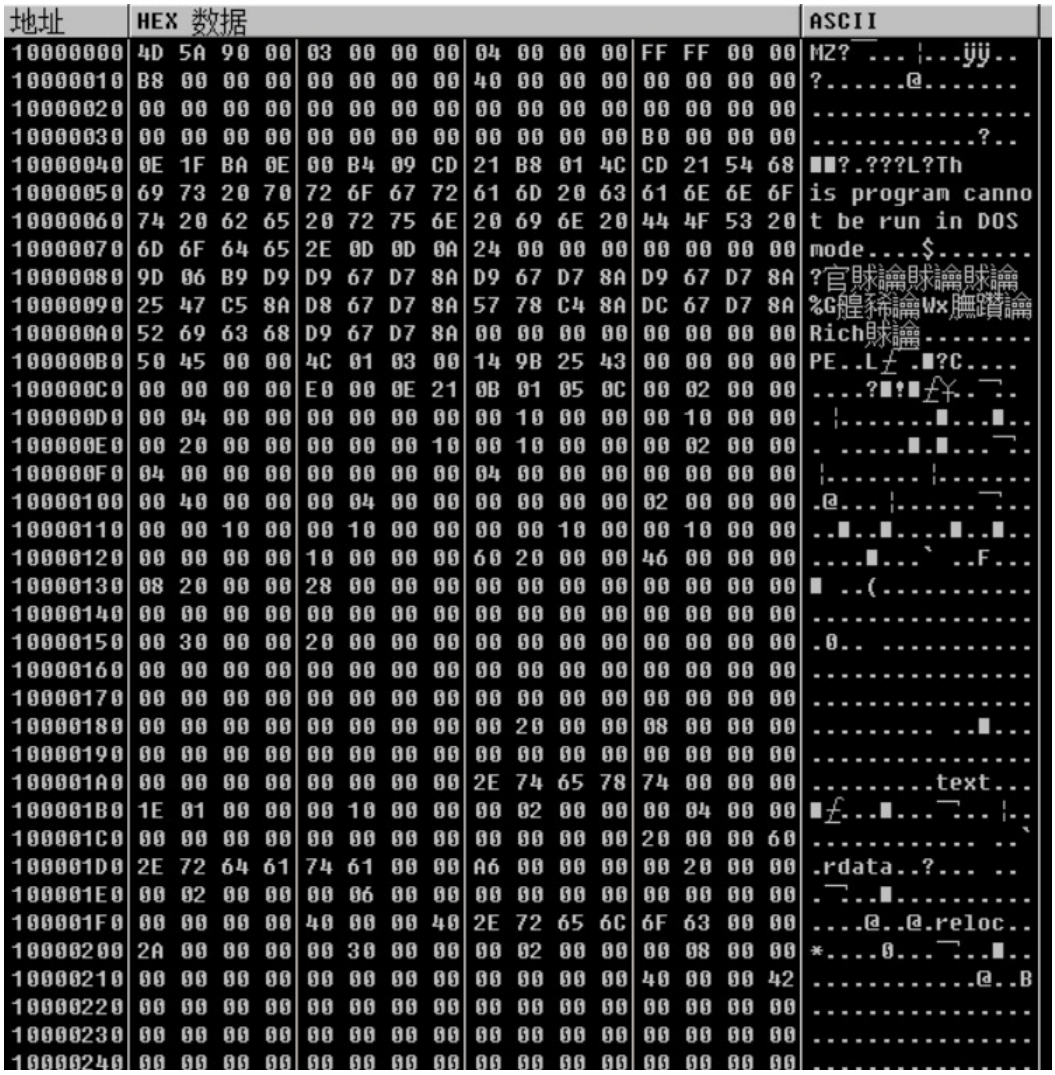
> 1. main函数

程序入口在 `0x401020` ：

```
00401020  r$  68 00304000   push code.00403000                  ; FileName = "tstdll.dll"
00401025  .   E8 E8FFFFFF   call <jmp.&kernel32.LoadLibraryA>   ; LoadLibraryA
0040102A  .   A3 14304000   mov dword ptr ds:[0x403014],eax
0040102F  .   68 0B304000   push code.0040300B                  ; ProcNameOrOrdinal = "TestProc"
00401034  .   FF35 1430400  push dword ptr ds:[0x403014]        ; hModule = NULL
0040103A  .   E8 CDFFFFFF   call <jmp.&kernel32.GetProcAddress> ; GetProcAddress
0040103F  .   FFD0          call eax
00401041  .   FF35 1430400  push dword ptr ds:[0x403014]
00401047  .   E8 BAFFFFFF   call <jmp.&kernel32.FreeLibrary>    ; hLibModule = NULL
0040104C  .   6A 00         push 0x0                            ; FreeLibrary
0040104E  .   6A 00         push 0x0                            ; Style = MB_OK|MB_APPLMODAL
00401050  .   68 18304000   push code.00403018                  ; Title = NULL
00401055  .   6A 00         push 0x0                            ; Text = "HelloWorld"
00401057  .   E8 BCFFFFFF   call <jmp.&user32.MessageBoxA>      ; hOwner = NULL
0040105C  .   6A 00         push 0x0                            ; MessageBoxA
0040105E  L.  E8 9DFFFFFF   call <jmp.&kernel32.ExitProcess>    ; ExitCode = 0x0
                                                                ; ExitProcess
```

> 2. 动态加载链接库 `tstdll.dll`

加载完动态链接库 `tstdll.dll` 后(`0x401025`)，返回了该链接库加载到内存的地址，在 `0x10000000` 处：



执行完 `GetProcAddress` 后(`0x40103A`)，返回了该链接库的 `TestProc` 函数的地址，在 `0x100010DB` 处。

找到该地址，即可看到 `TestProc` 函数的内容：

```
100010DB   EB 26            jmp short tstdll.10001103
100010DD   54               push esp
100010DE   65:73 74         jae short 10001155
100010E1   2066 75          and byte ptr ds:[esi+0x75],ah
100010E4   6e               outs dx,byte ptr ds:[esi]
100010E5   637469 6F        arpl word ptr ds:[ecx+ebp*2+0x6F],si
100010E9   6e               outs dx,byte ptr ds:[esi]
100010EA   005468 69        add byte ptr ds:[eax+ebp*2+0x69],dl
100010EE   73 20            jnb short tstdll.10001110
100010F0   6973 20 747374   imul esi,dword ptr ds:[ebx+0x20],0x64747374
100010F7   6c               ins byte ptr es:[edi],dx
100010F8   6c               ins byte ptr es:[edi],dx
100010F9   2e               cs:
100010FA   64:6c            ins byte ptr es:[edi],dx
100010FC   6c               ins byte ptr es:[edi],dx
100010FD   2068 65          and byte ptr ds:[eax+0x65],ch
10001100   72 65            jb short tstdll.10001167
10001102   006A 00          add byte ptr ds:[edx],ch
10001105   68 DD100010      push tstdll.100010DD
1000110A   68 EB100010      push tstdll.100010EB
1000110F   6A 00            push 0x0
10001111   E8 02000000      call <jmp.&user32.MessageBoxA>
10001116   C3               retn
```

与 `tstdll.asm` 里的代码对应：

```
TestProc proc

    jmp @F
      MbTitle db "Test function",0
      MbMsg db "This is tstdll.dll here",0
    @@:

    invoke MessageBox,NULL,addr MbMsg,addr MbTitle,MB_OK

    ret

TestProc endp
```

> 3. 静态加载链接库 `user32.dll`

静态加载的 `user32.dll` 的IAT表在 `0x2014` (RVA) → `0x402014` (VA):

```
0640h: 14 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . ..............
0650h: 00 00 00 00 00 00 00 00 82 20 00 00 90 20 00 00  ........, ... ..
0660h: A2 20 00 00 74 20 00 00 00 00 00 00 C0 20 00 00  ¢ ..t ......À ..
0670h: 00 00 00 00 9B 00 45 78 69 74 50 72 6F 63 65 73  ....›.ExitProces
0680h: 73 00 D1 00 46 72 65 65 4C 69 62 72 61 72 79 00  s.Ñ.FreeLibrary.
0690h: 53 01 47 65 74 50 72 6F 63 41 64 64 72 65 73 73  S.GetProcAddress
06A0h: 00 00 EA 01 4C 6F 61 64 4C 69 62 72 61 72 79 41  .ê.LoadLibraryA
06B0h: 00 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C 00 00  ..kernel32.dll..
06C0h: B1 01 4D 65 73 73 61 67 65 42 6F 78 41 00 75 73  ±.MessageBoxA us
06D0h: 65 72 33 32 2E 64 6C 6C 00 00 00 00 00 00 00 00  er32.dll........
```

地址	HEX 数据
00402014	30 19 A3 75 00 00 00 00

IAT表中第一个DWORD即为 `MessageBox` 的地址（`0x75A31930`）。

找到该地址，即可看到 `MessageBox` 的内容:

```
75A31930   8BFF            mov edi,edi                          code.<ModuleEn
75A31932   55              push ebp
75A31933   8BEC            mov ebp,esp
75A31935   833D 783CA575   cmp dword ptr ds:[0x75A53C78],0x0
75A3193C   74 22           je short user32.75A31960
75A3193E   64:A1 18000000  mov eax,dword ptr fs:[0x18]
75A31944   BA 9C40A575     mov edx,user32.75A5409C
75A31949   8B48 24         mov ecx,dword ptr ds:[eax+0x24]
75A3194C   33C0            xor eax,eax
75A3194E   f0:0fb10a       lock cmpxchg dword ptr ds:[edx],ecx
75A31952   85C0            test eax,eax
75A31954   75 0A           jnz short user32.75A31960
75A31956   C705 CC3CA575   mov dword ptr ds:[0x75A53CCC],0x1
75A31960   6A FF           push -0x1
75A31962   6A 00           push 0x0
75A31964   FF75 14         push dword ptr ss:[ebp+0x14]
75A31967   FF75 10         push dword ptr ss:[ebp+0x10]
75A3196A   FF75 0C         push dword ptr ss:[ebp+0xC]
75A3196D   FF75 08         push dword ptr ss:[ebp+0x8]
75A31970   E8 3B020000     call user32.MessageBoxTimeoutA
75A31975   5D              pop ebp                              code.0040105C
75A31976   C2 1000         retn 0x10
```