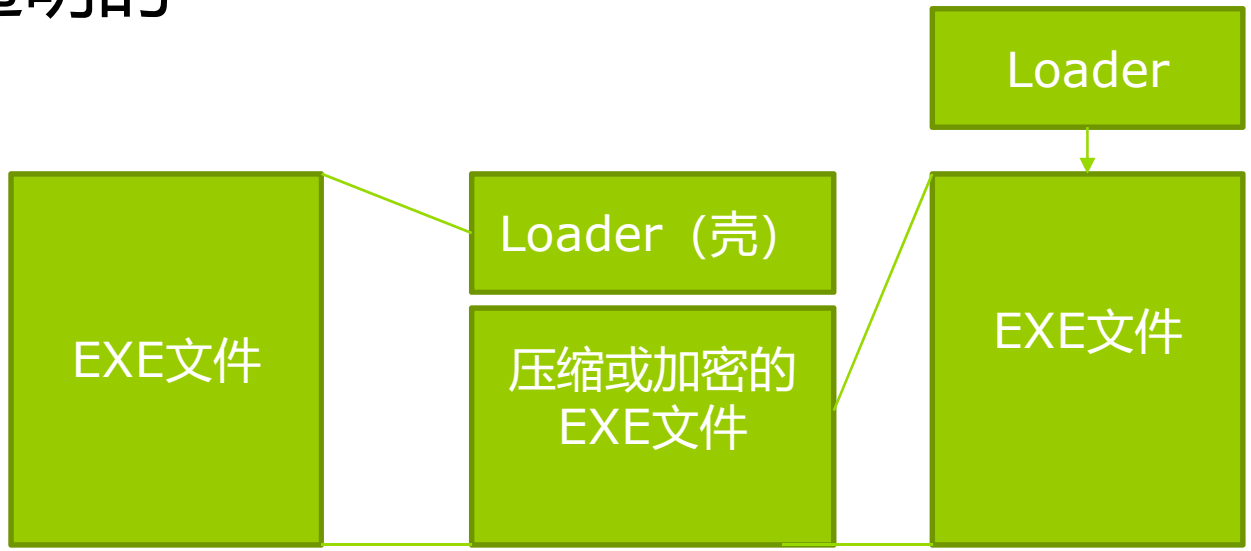
八、软件加壳与脱壳

壳的概念

- 壳是指在软件外面包裹上另一段代码，专门负责保护软件不被非法修改或反编译
- 先于软件执行，之后还原软件，并执行
- 用户执行的实际上是外壳程序，负责把用户原来的程序在内存中解压缩，并把控制权交还给解开后的真正程序，这一切工作都是在内存中运行的，整个过程对用户是透明的



壳的作用

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- **加壳改变其原来的特征码，隐藏一些字符串等，使软件不能被修改**
  - 防止被识别
  - 防止被篡改
  - 压缩可执行文件（压缩壳）
  - 对运行软件进行保护（防调试）

壳的发展历史

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 最早提出壳概念的人是脱壳软件RCOPY 3的作者熊焰
- 在DOS时代，壳一般是指磁盘加密软件中的一段加密程序，壳与需要加密的程序之间总有一条比较明显的“分界线”
- 脱壳技术的进步推动了当时加壳技术的发展，LOCK95和BITLOK等所谓“壳中带籽”的加密程序纷纷出现

壳的发展历史

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 在国内的加壳软件和脱壳软件较量得激烈时，国外的壳类软件早已发展到LZEXE之类的压缩壳。这类软件其实就是一个标准的加壳软件，把EXE文件压缩之后，在文件上加上一层在软件执行时自动将文件解压缩的壳，以达到压缩EXE文件的目的
- 国外淘汰磁盘加密，转向使用软件序列号加密，保护EXE文件不被动态跟踪和静态反编译变得非常重要，专门用于实现这类功能的加壳程序应运而生，MESS、CRACKSTOP、HACKSTOP、TRAP、UPS等是这类软件的代表

壳的发展历史

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 加壳类软件。例如BJFNT、PELOCKNT等，它们的出现使暴露了3年多的Windows下PE格式的EXE文件得到了很好的保护
- 压缩壳 (Packers)。UPX、ASPack、PECompact等是其中的佼佼者
- 加密壳（Protectors)。加密壳使用各种反跟踪技术来保护程序不被调试、脱壳等，加壳后软件的体积不是其考虑的主要因素，代表软件有ASProtect、Armadillo、EXECryptor等
- 随着加壳技术的发展，压缩壳和加密壳之间的界线越来越模糊，很多加壳软件不仅具有较强的压缩性能，也具有较强的保护性能

壳的发展历史

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 虚拟机技术应用到壳的领域。代表软件有VMProtect、Themida
- 设计了一套虚拟机引擎，将原始的汇编代码转译成虚拟机指令，要理解原始的汇编指令，就必须对其虚拟机引擎进行研究，极大地增加了破解和逆向的难度及时间成本

壳的发展历史

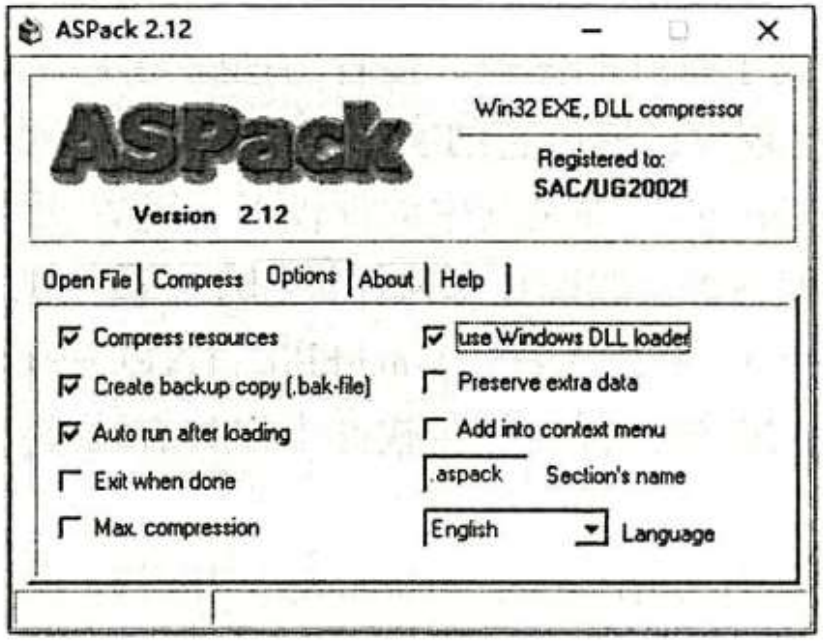
哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 壳在对软件提供良好保护的同时，也带来了兼容性方面的问题，选择一款壳保护软件后，需要在不同的硬件和系统上进行测试
- 壳能保护自身代码，许多木马和病毒都喜欢用壳来保护及隐藏自己
- 对一些流行的壳，杀毒引擎能先对目标软件进行脱壳，再进行病毒检查。对大多数私人壳，杀毒软件不会专门开发解压引擎，而是直接把壳当成木马或病毒来处理
- 越来越多的商业软件出于对兼容性的考虑，已经很少使用加壳保护，转而从其他方面提高软件保护强度

常见压缩壳

压缩壳

- **ASPack**：一款Win32可执行文件压缩软件，可压缩Win32可执行文件EXE、DLL、OCX，具有很高的兼容性和稳定性
- **UPX**：一个以命令行方式操作的可执行文件压缩程序，免费开源，兼容性和稳定性很好，UPX有DOS、Linux和Windows等版本
- **PECompact**



常见加密壳

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

❖ 加密壳

- **ASProtect**：一款非常强大的Win32保护工具，它的出现开创了壳的新时代，拥有压缩、加密、反跟踪代码、CRC校验和花指令等保护措施，使用Blowfish、Twofish、TEA等强大的加密算法，以RSA1024为注册密钥生成器，通过API钩子与加壳的程序通信。同时，为软件开发人员提供了SDK，从而实现了加密程序的内外结合。SDK支持VC、VB、Delphi等
- Armadillo
- EXECryptor
- Themida

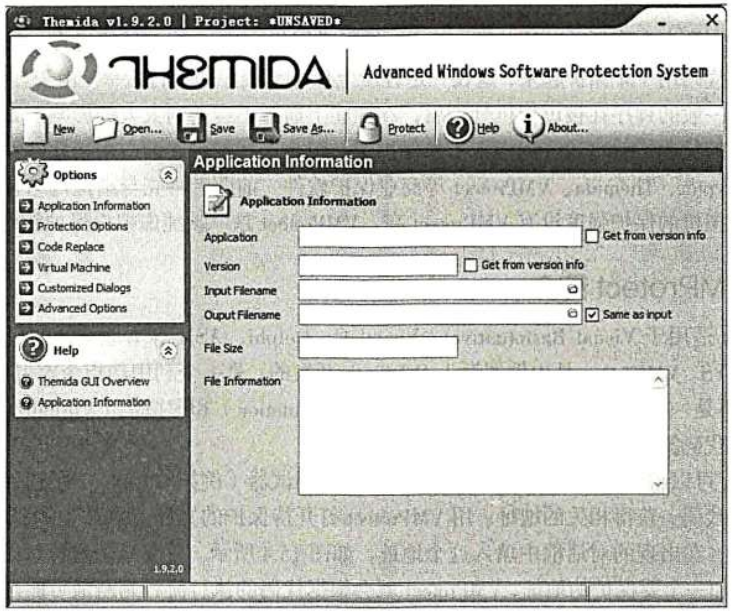
常见加密壳

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- Armadillo可以运用多种手段来保护软件，也可以为软件加上多种限制，包括时间、次数、启动画面等
- EXECryptor是一款商业保护软件，可以为目标软件添加注册机制、时间限制、使用次数等附加功能。其特点是其Anti-Debug比较强大，也做得比较隐蔽，并采用了虚拟机来保护一些关键代码。要发挥这款壳强大的保护能力，必须合理使用SDK，用虚拟机将关键的功能代码保护起来

常见加密壳

- Themida是Oreans的一款商业保护软件，Themida最大的特点就是其虚拟机保护技术，因此在程序中要善用SDK，将关键的代码交给Themida用虚拟机进行保护
- Themida最大的缺点就是生成的软件体积有些大



虚拟机保护软件

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 一个虚拟机引擎主要由编译器、解释器和虚拟CPU环境（VPU Context)组成，还会搭配一个或多个指令系统
- 虚拟机在运行时，先根据自定义的指令系统把已知的x86指令解释成字节码并放在PE文件中，然后将原始代码删除，进入虚拟机执行循环
- 调试者跟踪并进入虚拟机后很难理解原指令
- 跟踪虚拟机内代码执行的工作非常繁重。要想理解程序的流程，就必须对虚拟机引擎进行深入的分析
- 虚拟机技术是以效率换安全的

虚拟机保护软件

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- VMProtect适用于Visual Basic(native)、Visual C、Delphi、ASM等本地编译的目标程序，支持EXE、DLL、SYS
- VMProtect并不是一款壳，它将指定的代码进行变形(Mutation)和虚拟化(Virtualization)处理后，能很好地隐藏代码算法，防止算法被逆向

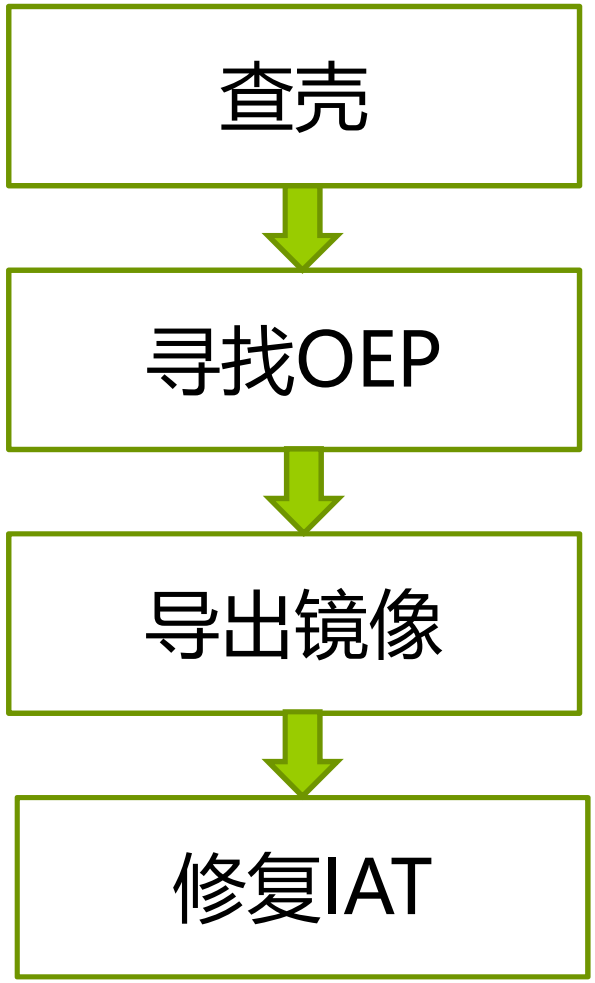
壳的加载过程

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

1. **保存入口参数**
   - 加壳程序在初始化时会保存各寄存器的值，待外壳执行完毕，再恢复各寄存器的内容，最后跳到原程序执行。通常用pushad/popad、pushfd/popfd指令对来保存与恢复现场环境
2. **获取壳自己所需要使用的API地址**
   - 外壳的输入表中只有GetProcAddress、GetModuleHandle和LoadLibrary等3个API函数，甚至只有Kernel32.dll及GetProcAddress
3. **解密原程序的各个区块的数据**
   - 壳一般会加密原程序文件的各个区块。在程序执行时，外壳将解密这些区块数据，从而使程序能够正常运行

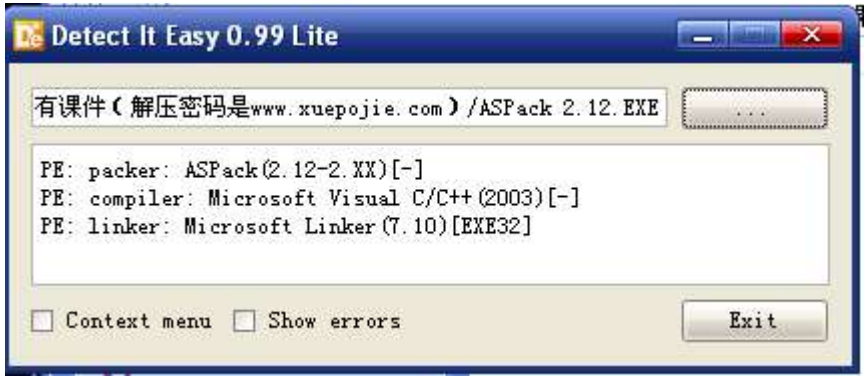
4. **重载输入表**
   - IAT的填写本来应该由PE装载器实现，但由于在加壳时构造了一个自建输入表，并让PE文件头数据目录表中的输入表指针指向自建的输入表，PE装载器会对自建的输入表进行填写。程序的原始输入表被外壳变形后存储，IAT的填写会由外壳程序实现
5. **跳转到程序原入口点（OEP）**

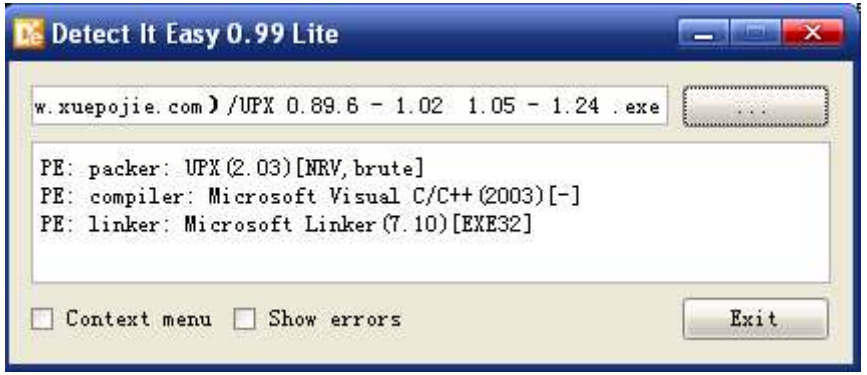
脱壳的步骤



查壳工具

DiE64
diel
DiE99
EnigmaInfo
ExeinfoPe
FFI中文版
PEID
查找OEP





脱壳工具

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

壳名称	定向脱壳工具
ASPack	ASPack unpacker
	AspackDie
	waspack
AsProtect	CASPr
	AsprDgr
	ASProtect unpacker
E_code	EUnpacker
FSG	UnFSG2.0
NsPack	wnspack
PECompact	UnPecompact
UPX	UPXshell
	Upxfix
(Win)Upack	WinUpack Stripper v0.3x

手动脱壳

重要概念：

- PUSHAD(压栈)代表程序的入口点
- POPAD(出栈)代表程序的出口点，与PUSHAD对应，找到这个OEP就在附近
- OEP：程序的入口点，软件加壳就是隐藏了OEP，只要找到程序真正的OEP，就可以立刻脱壳

主流的判定依据主要有以下四类特征：

- 编译器入口点指令特征
- 跨区段跳转特征
- 用内存访问断点寻找OEP
- ESP堆栈平衡

入口点指令特征

VC++ 6.0

```
0040A41E > 55                  push ebp
0040A41F   8BEC                mov ebp,esp
0040A421   6A FF               push -1
0040A423   68 C8CB4000         push xxxxxx
0040A428   68 A4A54000         push <jmp.&MSVCRT._except_handler3>
0040A42D   64:A1 00000000      mov eax,dword ptr fs:[0]
0040A433   50                  push eax
0040A434   64:8925 0000000>    mov dword ptr fs:[0],esp
0040A43B   83EC 68             sub esp,68
0040A43E   53                  push ebx
0040A43F   56                  push esi
0040A440   57                  push edi
```

BC++

```
00401678 > /EB 10              jmp short xxxxxx
0040167A   |66:623A            bound di,dword ptr ds:[edx]
0040167D   |43                 inc ebx
0040167E   |2B2B               sub ebp,dword ptr ds:[ebx]
00401680   |48                 dec eax
00401681   |4F                 dec edi
00401682   |4F                 dec edi
00401683   |4B                 dec ebx
00401684   |90                 nop
00401685  -|E9 98005400        jmp xxxxxx
0040168A   \A1 8B005400        mov eax,dword ptr ds:[xxxxxx]
0040168F   C1E0 02             shl eax,2
00401692   A3 8F005400         mov dword ptr ds:[xxxxxx],eax
00401697   52                  push edx
00401698   6A 00               push 0
0040169A   E8 99D01300         call <jmp.&KERNEL32.GetModuleHandleA>
0040169F   8BD0                mov edx,eax
```

VB

```
004012D4 > 68 54474000         push xxxxxx
004012D9   E8 F0FFFFFF         call <jmp.&MSVBVM60.#100>
004012DE   0000                add byte ptr ds:[eax],al
004012E0   0000                add byte ptr ds:[eax],al
004012E2   0000                add byte ptr ds:[eax],al
004012E4   3000                xor byte ptr ds:[eax],al
004012E6   0000                add byte ptr ds:[eax],al
004012E8   48                  dec eax
```

MASM

```
004035C9 > 6A 00               push 0
004035CB   E8 A20A0000         call <jmp.&kernel32.GetModuleHandleA>
004035D0   A3 5B704000         mov dword ptr ds:[xxxxxx],eax
```

Delphi

```
004A5C54 > 55                  push ebp
004A5C55   8BEC                mov ebp,esp
004A5C57   83C4 F0             add esp,-10
004A5C5A   B8 EC594A00         mov eax,xxxxxx
```

跨区段跳转特征

- 壳代码和被加壳代码分布于不同的区段中，不连续
- 当壳代码完成对原程序代码的解密、解压后，需要通过跨区段跳转方式将控制权转交给原程序代码

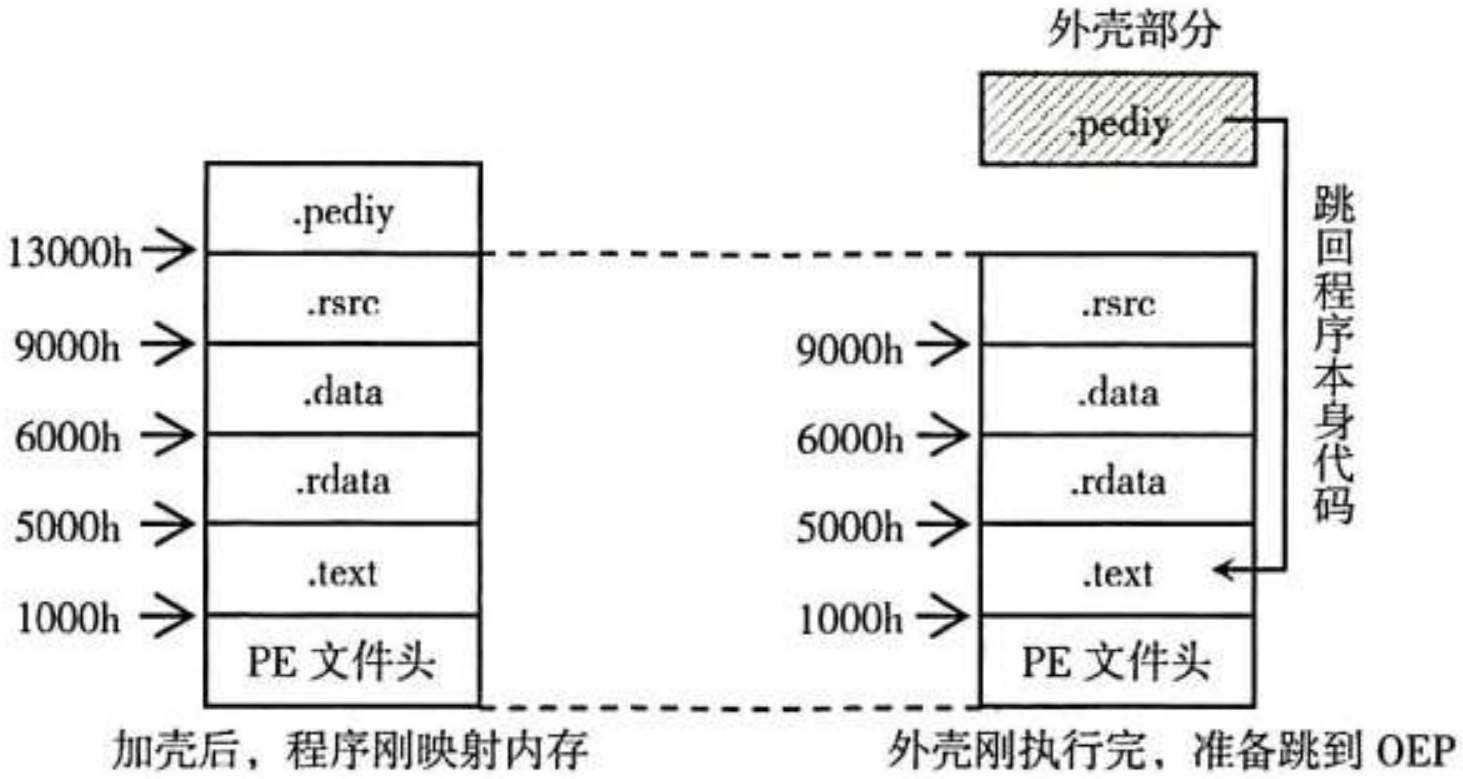
[Section Table]

Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	000036DE	00001000	00004000	60000020
.rdata	00005000	0000084E	00005000	00001000	40000040
.data	00006000	000029FC	00006000	00003000	C0000040
.rsrc	00009000	00009A70	00009000	0000A000	40000040

[Section Table]

Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	00004000	00000400	00002400	E0000020
.rdata	00005000	00001000	00002800	00000200	C0000040
.data	00006000	00003000	00002A00	00000200	C0000040
.rsrc	00009000	0000A000	00002C00	00001200	C0000040
.pediy	00013000	00007000	00003E00	00006A00	E0000040

根据跨段指令寻找OEP



手动脱壳

哈爾濱工業大學 HARBIN INSTITUTE OF TECHNOLOGY

根据跨段指令寻找OEP

```
00370283  push    401130
00370288  retn
```

```
00401130      55      db      55          ; CHAR 'U'
00401131      8B      db      8B
00401132      EC      db      EC
00401133      6A      db      6A          ; CHAR 'j'
```

```
00401130   55              push    ebp
00401131   8BEC            mov     ebp, esp
```

跨段跳跃

Address	Size	Owner	Section	Contains	Type	Access	Initial
00370000	00001000				Priv 00021004	RW	RW
00380000	00004000				Priv 00021004	RW	RW
00390000	00003000				Map 00041002	R	R
003A0000	00008000				Priv 00021004	RW	RW
003B0000	00001000				Priv 00021004	RW	RW
003C0000	00001000				Priv 00021004	RW	RW
00400000	00001000	RebPE		PE header	Imag 01001002	R	RWE
00401000	00004000	RebPE	.text	code	Imag 01001002	R	RWE
00405000	00001000	RebPE	.rdata		Imag 01001002	R	RWE
00406000	00003000	RebPE	.data	data	Imag 01001002	R	RWE
00409000	0000A000	RebPE	.rsrc	resources	Imag 01001002	R	RWE
00413000	00007000	RebPE	.pediy	SFX,imports	Imag 01001002	R	RWE

用内存访问断点寻找OEP

Address	Size	Owner	Section	Contains	Type	Access	Initial
00400000	00001000	RebPE		PE header	Imag 01001002	R	RWE
[redacted]	00004000	RebPE	.text	code	Imag 01001002	R	RWE
00405000	00001000	RebPE	.rdata		Imag 01001002	R	RWE
00406000	00003000	RebPE	.data	data	Imag 01001002	R	RWE
00409000	0000A000	RebPE	.rsrc	resources	Imag 01001002	R	RWE
00413000	00007000	RebPE	.pediy	SFX,imports	Imag 01001002	R	RWE

```
00413145  movs    byte ptr es:[edi], byte ptr [esi]     ;将在此处中断
00413146  mov     bl, 2
00413148  call    004131BA
0041314D  jnb     short 00413145
......
004131D6  sub     edi, dword ptr [esp+28]
004131DA  mov     dword ptr [esp+1C], edi
004131DE  popad
004131DF  retn    8
```

ESP平衡原理

- 在被加壳程序的运行过程中，壳代码需要将原程序代码在内存中进行解压、解密操作。壳代码对于被加壳程序是透明的
- 为了确保原程序运行的稳定性，壳代码在运行过程中需要对当前进程的上下文环境进行保护，其中一项就是对堆栈的保护
- 根据堆栈平衡定律，在被加壳程序到达真正的原始入口点OEP时，必须保证堆栈的状态和加壳后程序的入口处状态相同
- 同时，二者的栈顶指针ESP状态也应该保持相同
- 监控堆栈信息变化，根据堆栈平衡原理快速筛选出候选的原始入口点OEP

手动脱壳


```
    PUSHAD          ;PUSHAD 相当于 push eax/ecx/edx/ebx/esp/ebp/esi/edi
    ……              ;外壳代码
    POPAD           ;POPAD 相当于 pop edi/esi/ebp/esp/ebx/edx/ecx/eax
    JMP OEP         ;准备跳到入口点
OEP: ……             ;解压后程序的源代码
```

Registers (FPU)

```
EAX 00000000
ECX 0012FFB0
EDX 7C92E514 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C930228 ntdll.7C930228
```

Address	Value	Comment
0012FFC4	7C817077	RETURN to kernel32.7C817077
0012FFC8	7C930228	ntdll.7C930228
0012FFCC	FFFFFFFF	

	Address	Value	Comment
edi →	0012FFA4	7C930738	ntdll.7C930738
esi →	0012FFA8	FFFFFFFF	
ebp →	0012FFAC	0012FFF0	
esp →	0012FFB0	0012FFC4	
ebx →	0012FFB4	7FFD3000	
edx →	0012FFB8	7C92EB94	ntdll.KiFastSystemCallRet
ecx →	0012FFBC	0012FFB0	
eax →	0012FFC0	00000000	