



## ❖ Dump内存

- 抓取内存映像，也称“转存”（Dump），是指把内存指定地址的映像文件读出，用文件等形式将其保存下来的过程
- 当外壳来到OEP处时进行Dump是正确的。如果等到程序运行起来，由于一些变量已经初始化了，不适合进行Dump
- 在外壳处理过程中，外壳要把压缩后的全部代码数据释放到内存中，并初始化一些项目，也可以选择合适的位置进行Dump



# Dump内存



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ Dump内存

- 内存随时改变，选择在目标程序的原始入口点OEP处进行转存操作
- 根据确定的程序原始入口点位置，再查找进程环境
- 控制块中的相应字段来获取内存镜像的大小
- 然后将内存数据转存入磁盘，最后再将虚拟地址和文件地址对齐



# Dump内存



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ Dump原理

- 常用的Dump软件有LordPE等
- 一般利用Module32Next来获取欲Dump进程的基本信息

---

```
BOOL Module32Next (HANDLE hSnapshot , LPMODULEENTRY32 lpme)
```

---

```
typedef struct tagMODULEENTRY32 {  
    DWORD      dwSize;                //此结构的大小  
    DWORD      th32ModuleID;  
    DWORD      th32ProcessID;         //进程标识符  
    DWORD      GlblcntUsage;  
    DWORD      ProccntUsage;  
    BYTE*      modBaseAddr;           //进程映像基址  
    DWORD      modBaseSize;           //进程映像大小  
    HMODULE     hModule;               //进程句柄  
    TCHAR      szModule[MAX_MODULE_NAME32 + 1];  
    TCHAR      szExePath[MAX_PATH];    //进程的完整路径  
} MODULEENTRY32;  
typedef MODULEENTRY32 *PMODULEENTRY32; Members
```

---



# Dump内存



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ Dump原理

- 根据结构中的modBaseSize和modBaseAddr字段得到进程的映像大小和基址
- 调用ReadProcessMemory来读取进程内的数据
- 检测IMAGE\_DOS\_SIGNATURE和IMAGE\_NT\_SIGNATURE是否完整
- 如果不完整，会根据szExePath字段打开进程的原始文件，读取其文件头以取代进程的文件头

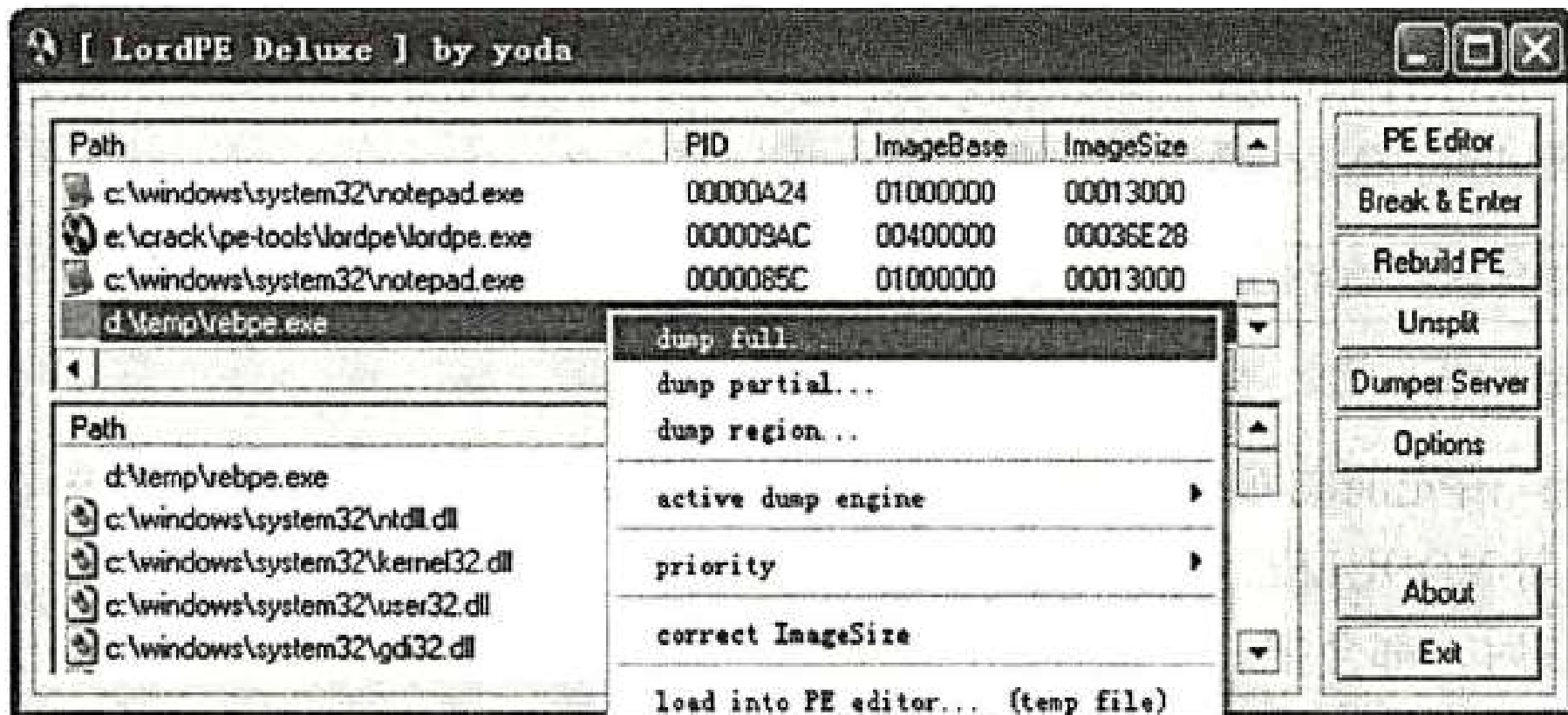


# Dump内存



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ Dump内存





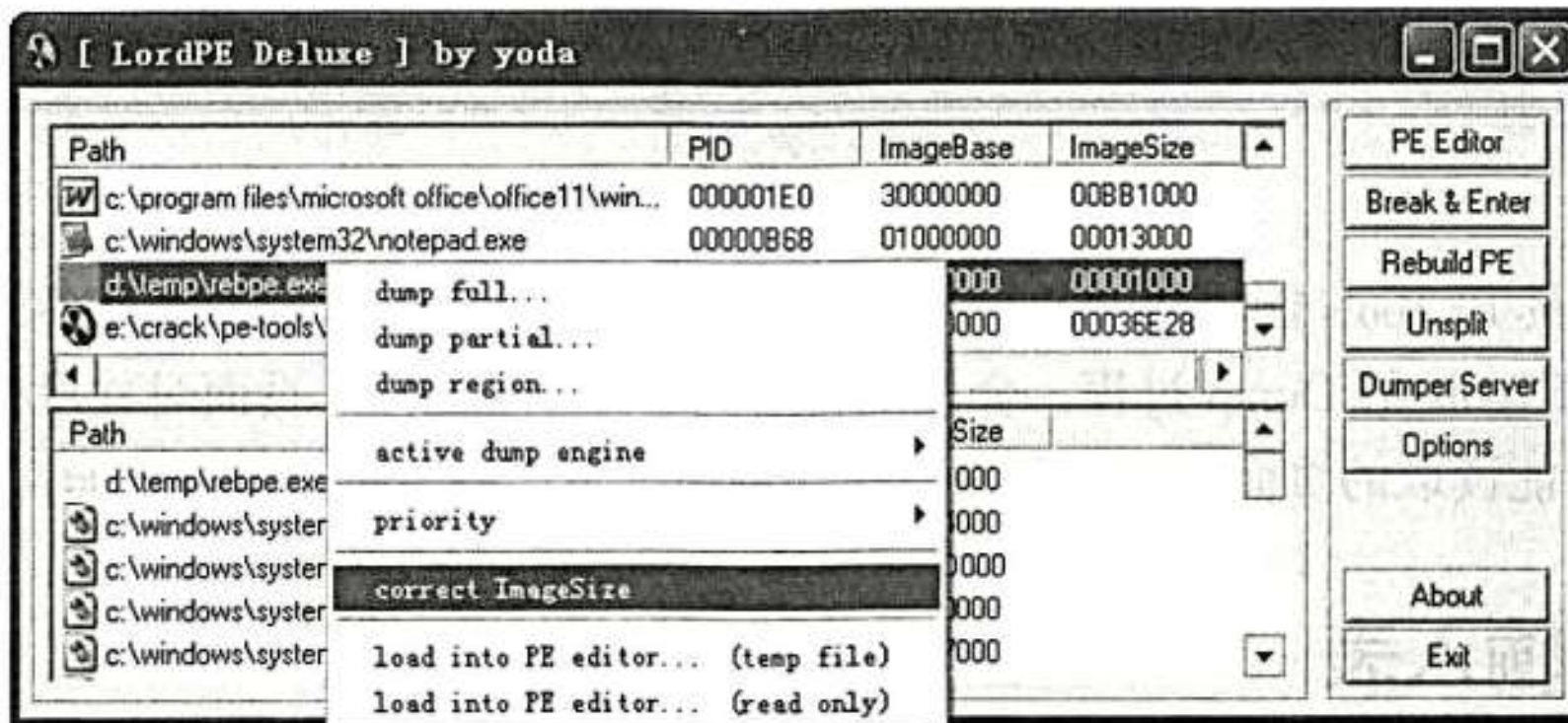
# Dump内存



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 反Dump技术

- **纠正SizeOfImage**: 在Dump文件时, 一些关键参数是通过MODULEENTRY32结构的快照获得的, 可以在modBaseSize和modBaseAddr字段中填入错误的值, 让Dump软件无法正确读取进程中的数据







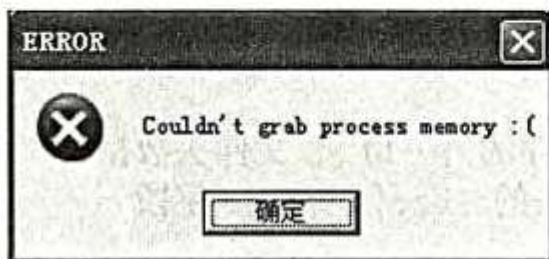
# Dump内存



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 反Dump技术

- **修改内存属性:** 当PE文件被加载到内存中时, 其所有段的属性都是可读的。这样, 在用Dump工具打开进程时, 就可以读取内存数据并将其转存到磁盘文件中了。如果进程的某个地址不可读, 那么使用某些Dump工具可能无法正确读取相关数据



[ Dump Region ]				
Address	Size	Protect	State	Type
00400000	00001000	NOACCESS	COMMIT	IMAGE
00401000	00007000	XR	COMMIT	IMAGE
00408000	00001000	R	COMMIT	IMAGE
00409000	00001000	RW	COMMIT	IMAGE
0040A000	00001000	WC	COMMIT	IMAGE
0040B000	00002000	RW	COMMIT	IMAGE
0040D000	7C3F3000	NOACCESS	FREE	
7C800000	00001000	XRW	COMMIT	IMAGE
7C801000	00001000	XR	COMMIT	IMAGE
7C802000	00001000	XRW	COMMIT	IMAGE
7C803000	00008000	XR	COMMIT	IMAGE
7C883000	00001000	XRW	COMMIT	IMAGE
7C884000	00003000	RW	COMMIT	IMAGE

Dump Information

Address: 00400000 Size: 00001000

Dump



# Dump内存



哈尔滨工业大学  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 反Dump技术

Address	Size	Owner	Section	Contains	Type	Access	Initial	Happed as
00400000	00001000	Modify_t		PE header	Image	RWE	00000000	RWE
00401000	00007000	Modify_t	.text	Actualize	code			
00408000	00001000	Modify_t	.rdata	Dump in CPU	image			
00409000	00004000	Modify_t	.data	Dump	data			
7C800000	00001000	kernel32		Search	PE		Ctrl+B	
7C801000	00003000	kernel32	.text		code			
7C884000	00005000	kernel32	.data	Set break-on-access	data		F2	
7C889000	0000E000	kernel32	.rsrc		resource			
7C917000	00006000	kernel32	.reloc	Set memory breakpoint on access	resource			
7C920000	00001000	ntdll		Set memory breakpoint on write	PE			
7C921000	00007000	ntdll	.text	Set access	code			
7C99C000	00005000	ntdll	.data		data			
7C9A1000	00010000	ntdll	.rsrc	Copy to clipboard	resource			
7C9B1000	00003000	ntdll	.reloc	Sort by	resource			
7F6F0000	00000000			Appearance				

Command

No access

Read only

Read/write

Execute

Execute/read

Full access





# 重建输入表



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 重建IAT表

- 壳代码的执行过程中，不但会完成对原程序代码的还原操作，还会完成IAT导入表的重定向
- 壳代码大多会将原程序的导入函数名称字符串、DLL 名称字符串进行加密或者清除操作
- 当成功还原原程序代码后，由于原始的IAT导入表已经被加密或者破坏，原程序无法找到正确的执行地址，程序依然不能正常运行
- 需要修复IAT导入表，还原原程序的IAT地址范围



# 重建输入表



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

- ❖ 从原始入口点OEP开始进行反汇编分析，根据特征查找一个IAT表项作为搜索的起始点
- ❖ 鉴于IAT表项都是连续存放的，以找到的IAT表项作为起始点，向前向后进行扩展搜索，最终计算出完整的IAT范围
- ❖ 通过系统提供的API遍历目标程序加载的DLL模块，并将每个DLL模块的导出函数名称和地址等信息提取出来组成API列表
- ❖ 将搜索到的每一个IAT表项存放的地址与API列表进行匹配，查找对应的函数名称字符串，完成对IAT的修复工作

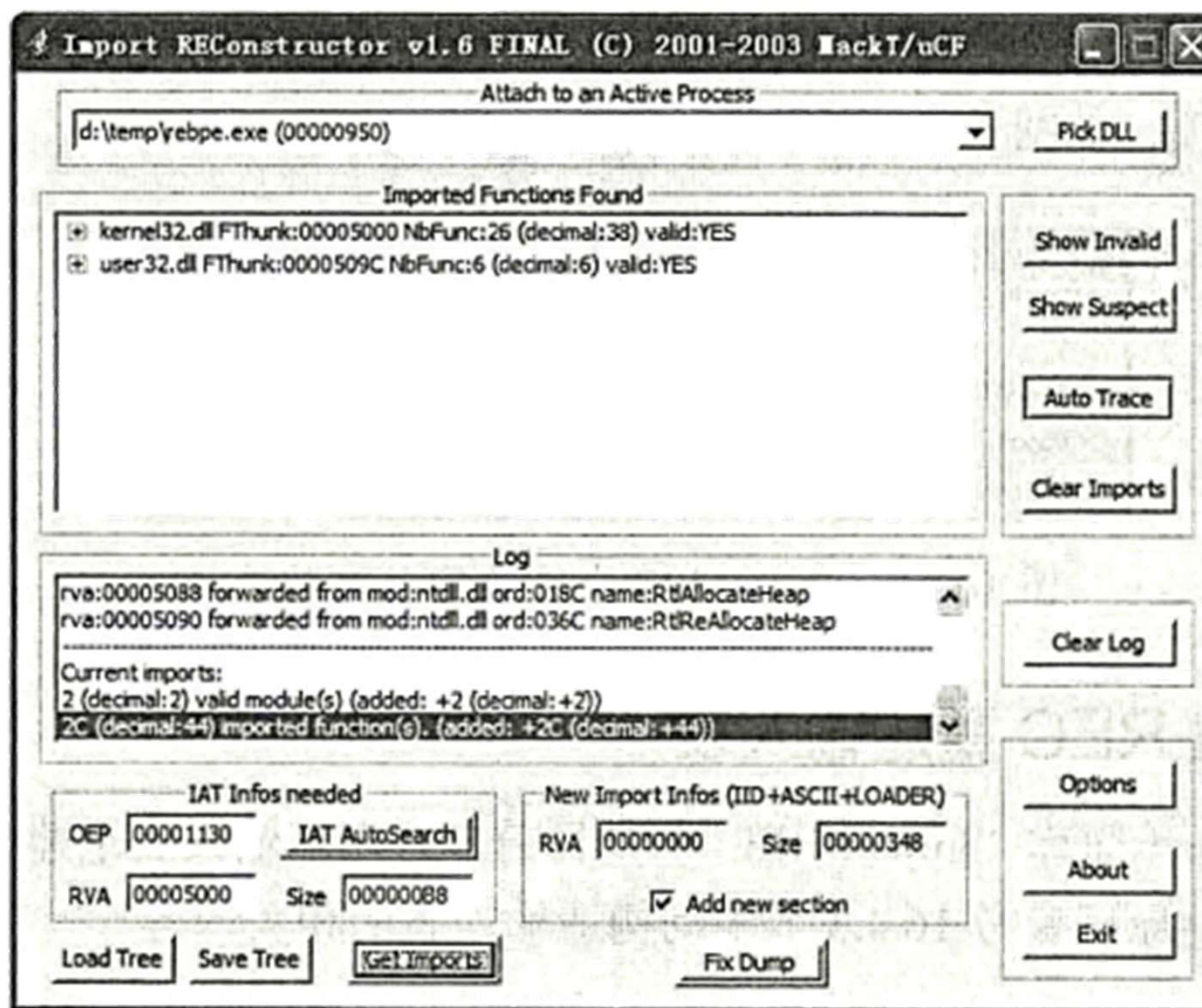


# 重建输入表



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 用Import REC重建输入表





# 重建输入表（加密）



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 完整地保留了原输入表，外壳加载时未对IAT加密

- 当外壳解压数据时，完整的输入表会在内存中出现
- 外壳用显式装载DLL的方式获得各函数的地址，并将该地址填充到IAT中
- 脱壳时，可以在内存映像文件刚生成时抓取输入表

## ❖ 完整地保留了原输入表，当外壳装载时对IAT进行加密处理

- 当外壳解压数据时，完整的输入表会在内存中出现
- 外壳用显式装载DLL的方式获得各函数地址，并对这些地址进行处理（即HookAPI）
- 将HookAPI的外壳代码的地址填充到IAT中



## 重建输入表（加密）



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

### ❖ 加壳时破坏了原输入表，外壳装载时未对IAT进行加密处理

- 外壳已经完全破坏原输入表，在外壳刚解压的映像文件中的是输入函数的字符串
- 外壳用显示装载DLL的方式获得这些函数的地址，直接将函数地址填充到IAT中
- 在脱壳时用Import REC根据IAT重建了一个输入表

### ❖ 加壳时破坏了原输入表，装载外壳时对IAT进行了加密处理

- 外壳已经完全破坏了原输入表，外壳将用显式装载DLL的方式获得各函数地址，并对该地址进行处理（即HookAPI）
- 将HookAPI的外壳代码的地址填充到IAT中
- 在脱壳时，不仅可以利用Import REC的一些插件来对付这些加密的IAT，也可以修改外壳处理输入函数地址的代码，使其生成的IAT不被加密，然后用Import REC重建输入表





# 常用脱壳技术总结



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

- ❖ **ESP定律**
- ❖ **两次断点法（内存镜像法）**
- ❖ **单步跟踪**
- ❖ **一步到达OEP（出口标志）**
- ❖ **SFX法**
- ❖ **最后一次异常法**



# 单步跟踪



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 单步跟踪目标软件的执行

## ❖ 识别OEP特征

- 跨段跳转
- OEP格式
- POPAD



# 单步跟踪



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 步过与步入

- CALL指令

## ❖ 跳过循环

## ❖ 识别大跳转



# 一步到达OEP（出口标志法）



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 原理：

- OEP之前往往是POPAD

## ❖ 步骤：

- Ctrl+F搜索POPAD，设置断点  
(只适合少数壳，包括UPX，ASPACK壳)
- 运行到断点
- 单步，到达OEP



# SFX法



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

- ❖ 将OD设置为忽略所有异常
- ❖ 在OD的“调试选项”对话框的“SFX”选项卡中选择“字节模式跟踪实际入口”选项并确定
- ❖ 将待脱壳程序载入OD，待程序载入完成后，会直接停在OEP处

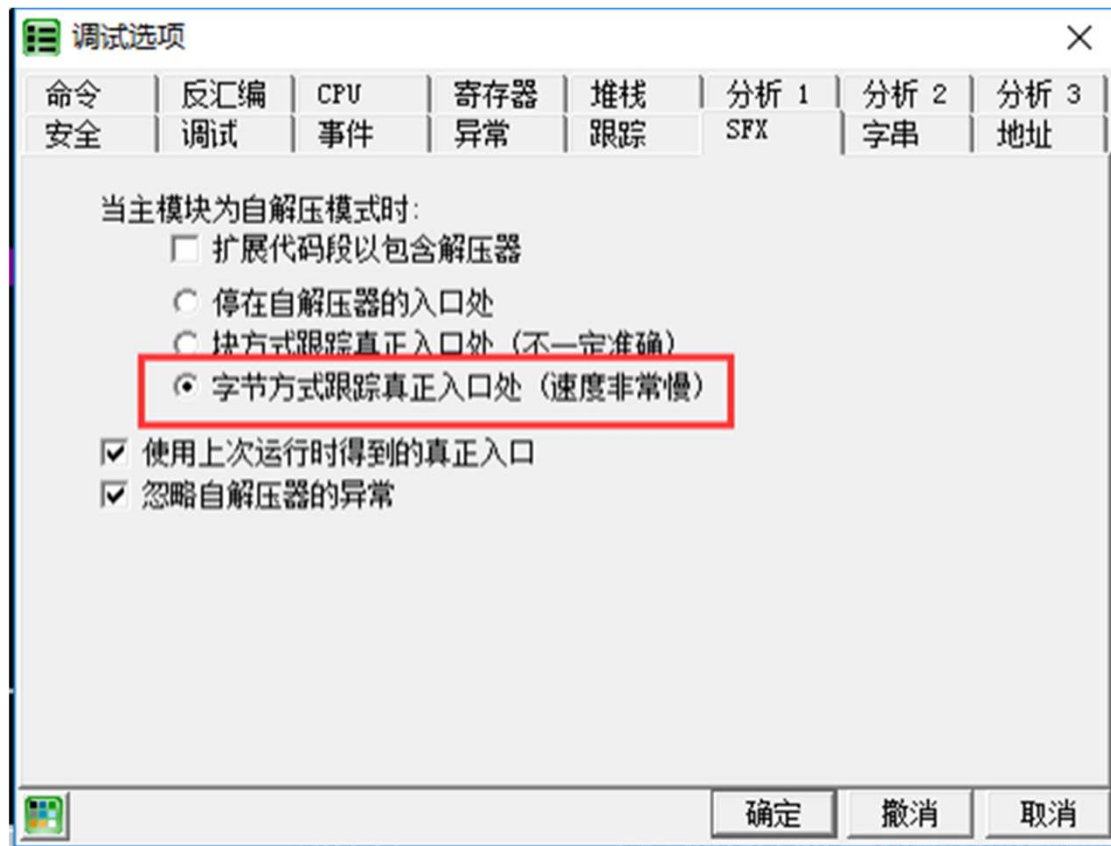




# SFX法



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY





# 最后一次异常法



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

## ❖ 原理：

- 壳运行时会产生大量异常
- 在OEP之前中断，寻找OEP

## ❖ 步骤：

- 在最后一次异常处设断点
- 单步跟踪异常处理程序
- 寻找OEP