**ReverseCore\_NoteBook**



目录

[13 PE文件结构 2](#_Toc486409985)

[13.1 PE头结构 4](#_Toc486409986)

[13.1.1 DOS头 4](#_Toc486409987)

[13.1.2 虚拟地址和文件偏移地址的换算（RVARAW） 4](#_Toc486409988)

[13.2 IAT（导入地址表） 5](#_Toc486409989)

[13.3 notepad.exe 练习 5](#_Toc486409990)

[14 运行时压缩（加壳） 6](#_Toc486409991)

[15 UPX调试脱壳 6](#_Toc486409992)

[15.1 跟踪代码和OEP定律 6](#_Toc486409993)

[15.2 修复IAT 7](#_Toc486409994)

[16 基地址重定位 7](#_Toc486409995)

[17 删除.reloc节 8](#_Toc486409996)

[18 Upack PE文件头详细分析 8](#_Toc486409997)

[19 Upack 调试 查找OEP 9](#_Toc486409998)

[20 内嵌补丁 9](#_Toc486409999)

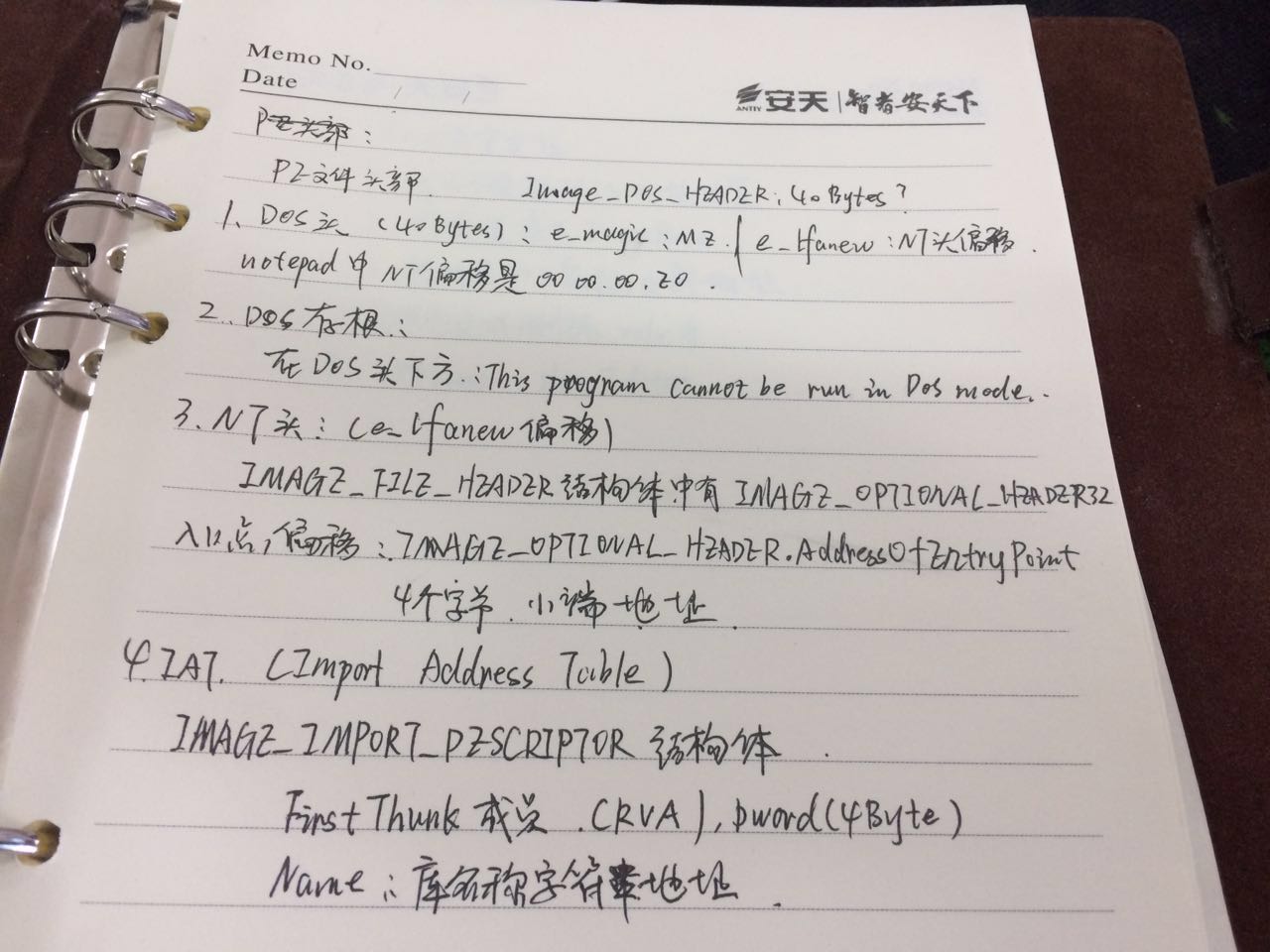
[21 Hook 9](#_Toc486410000)

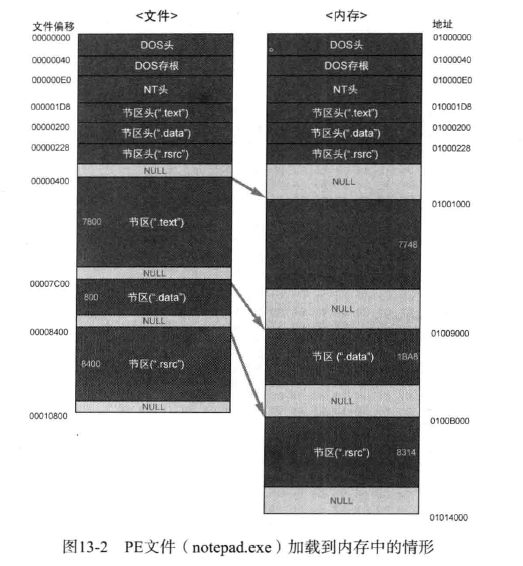
[22 击键记录器 9](#_Toc486410001)

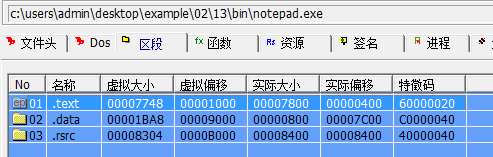
[23 Dll注入 9](#_Toc486410002)

# PE文件结构

两图为notepad.exe的文件结构







名词解释：

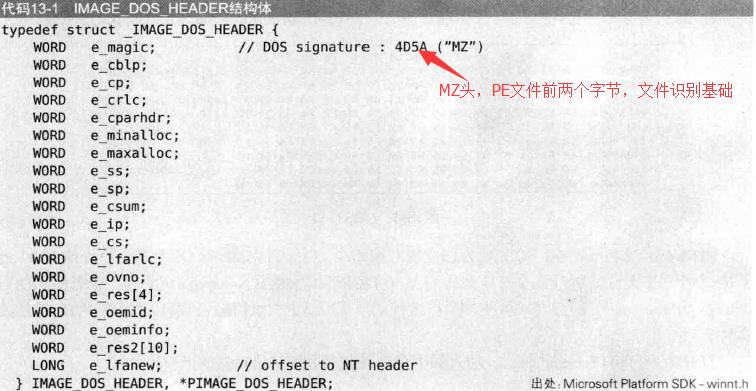
VA：进程虚拟内存的绝对地址

RVA：相对虚拟地址，从基准位置ImageBase开始的相对地址

VA = RVA + ImageBase

## PE头结构

### DOS头



DOS占用40个字节

### 虚拟地址和文件偏移地址的换算（RVARAW）



1. 查找RVA所在的区段

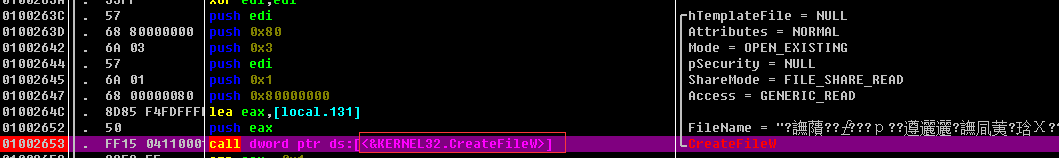
2. 利用公式计算RAW（文件偏移）

RAW – PointerToRawData = RVA – VirtualAddress（虚拟地址相对于虚拟偏移的位置等于文件偏移相当于当前节的文件偏移的位置）。

示例：RVA = 5000，求文件偏移

解： RVA = 5000，在1000和 9000之间，在text节，RAW = 5000 – 1000 + 400 = 4400.

## IAT（导入地址表）

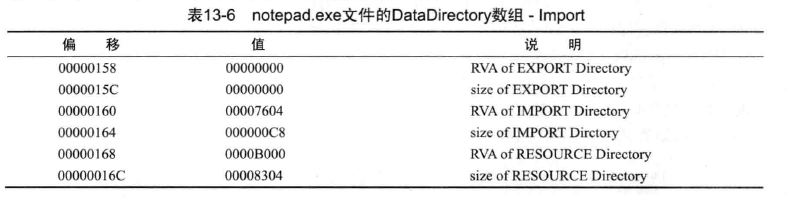


## notepad.exe 练习

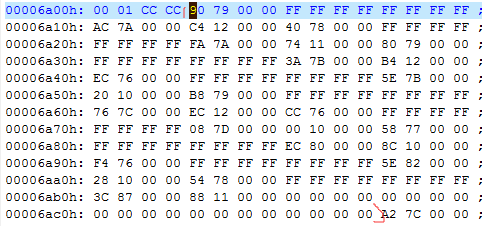
1. 从文件PE体中查找IMAGE\_OPTIONAL\_HEADER32.DataDirectory[1].

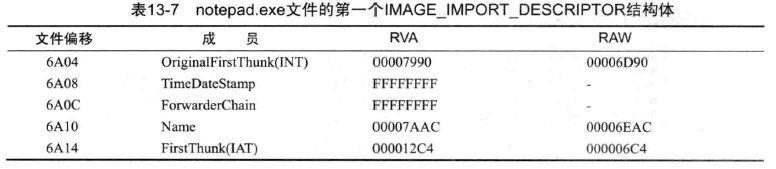
VirtualAddress（结构体数组的起始地址）。



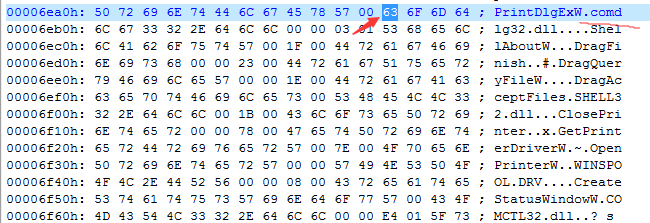


第一个字节是RVA，第二个字节是字节数。RVA = 00007604，size = 0xc8 。RAW = RVA – 1000 + 400（详见之前的公式） = 6A04





Name是导入函数的名称。 RVA ：7AAC =》RAW ：6EAC



# 运行时压缩（加壳）

UPX加壳特点：

1. PE头大小不变

2. 节名称有变化：.text => UPX0 | .data => UPX1

3. UPX0节的RawDataSize = 0

4. EP在第二个节区

5. .rsrc节大小几乎没有变化

# UPX调试脱壳

PUSHAD指令：

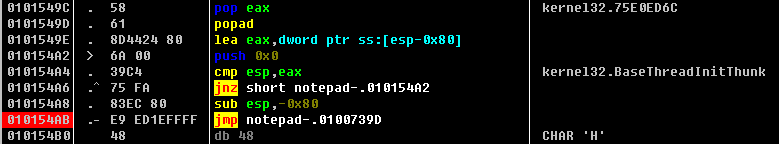
将EAX-EDI寄存器的值保存到栈中。

## 跟踪代码和OEP定律



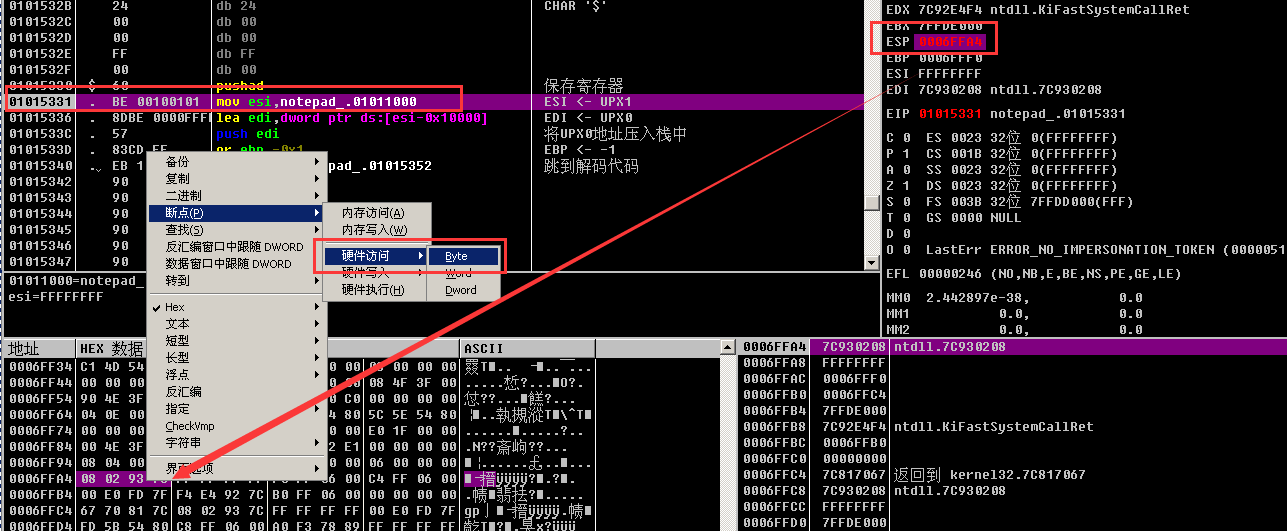
UPX快速脱壳方法：

1 popad 后面的jmp指令下断点



2 硬件断点（但是没有看明白）

在数据窗口找到执行完pushad后在esp对应地址，然后选择该地址开始的1字节，右键设置硬件断点（或者按书p133那样设置硬件断点均可）。



当程序执行popad时会首先从这个地址上读取数据到edi(pop edi)，此时会触发硬件断点，程序断下。其下方即是跳转到OEP的JMP指令。



不过和软件断点(INT 3)断点不同的是，触发断点的指令执行（popad）完成后程序才会停止运行，即程序会断在popad下一条指令。

OD的硬件断点可以在调试->硬件断点里看到。

## 修复IAT

ImportRCE工具和Ollydump

参考链接：<http://www.programlife.net/repair-import-table-through-importrec.html>

主要方法是，手动脱壳以后，用ollydump插件保存脱壳后的进程，然后记住入口点。Zzzzzz……还是看参考链接吧。

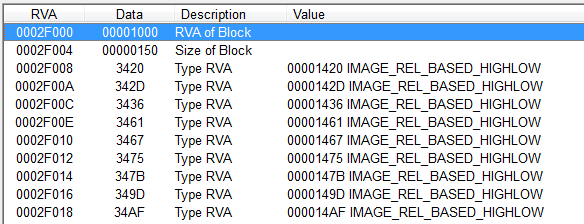
# 基地址重定位

Exe默认的ImageBase 是00400000，dll默认的ImageBase 是00100000 。

基址重定位表在PE头DataDirectory的第六个元素。



基址重定位表地址（RVA）为2f000。PEView 查看2f000地址。



TypeOffset的值是1000，块大小是0x150.



# 删除.reloc节

# Upack PE文件头详细分析

Upack 将MZ文件头和PE文件头进行重叠。

Stud\_PE查看文件：



PE头的地址是00000010，和DOS头重叠。



# Upack 调试 查找OEP

# 内嵌补丁

这章重新看。

# Windows消息钩取

Hookmain.cpp

#include *"stdio.h" //Hookmain.cpp*

#include *"conio.h"*

#include *"windows.h"*

#define DEF\_DLL\_NAME "KeyHook.dll"

#define DEF\_HOOKSTART "HookStart"

#define DEF\_HOOKSTOP "HookStop"

//typedef 关于函数指针的用法，typedef 返回值类型( \*新类型)(参数类型)

**typedef** void (\*PFN\_HOOKSTART)();

**typedef** void (\*PFN\_HOOKSTOP)();

void main()

{

HMODULE hDll = NULL;

PFN\_HOOKSTART HookStart = NULL;

PFN\_HOOKSTOP HookStop = NULL;

char ch = 0;

*// 加载KeyHook.dll*

hDll = LoadLibraryA(DEF\_DLL\_NAME);

**if**( hDll == NULL )

{

printf("LoadLibrary(%s) failed!!! [%d]", DEF\_DLL\_NAME, GetLastError());

**return**;

}

*// 获取导出两个导出函数地址，注意GetProcAddress前的函数指针和前面的typedef定义*

HookStart = (PFN\_HOOKSTART)GetProcAddress(hDll, DEF\_HOOKSTART);

HookStop = (PFN\_HOOKSTOP)GetProcAddress(hDll, DEF\_HOOKSTOP);

*// 开始钩取*

HookStart();

*// 等待用户输入q*

printf("press 'q' to quit!**\n**");

**while**( \_getch() != 'q' ) ;

*// 停止钩取*

HookStop();

FreeLibrary(hDll);

}

GetProcAddress()函数用法：第一个参数是LoadLibrary返回的dll句柄，第二个参数是字符串，要获取的函数名称。返回值与要获取的函数返回值同一类型，参见typedef定义。

# 击键记录器

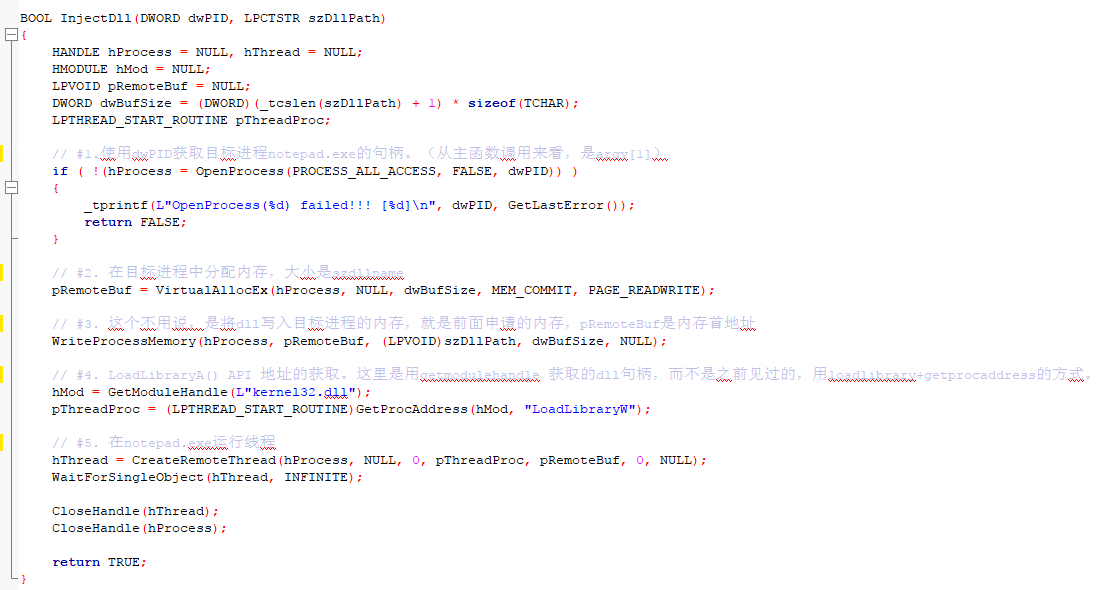
# Dll注入

统计dll注入的几种方法：

1. 创建远程线程（调用CreateRemoteThread API）

CreateRemoteThread API用来在目标进程中执行创建出的线程。

1. 利用注册表（AppInit\_DLLs的键值）
2. 利用消息钩子



一个进程注入的代码实例。