1.V.E.H利用

从windows xp开始，在全面兼容以前S.E.H异常处理机制的情况下，又加入了一种新的异常处理：V.E.H（Vectored Exception Handler；向量化异常处理）

(1)V.E.H和进程异常处理类似，都是基于进程，而且需要使用API注册回调函数

PVOID AddVectoredExceptionHandler(

ULONG FirstHandler,

PVECTORED\_EXCEPTION\_HANDLER VectoredHandler

);

(2)MSDN上对V.E.H结构的描述

Struct \_VECTORED\_EXCEPTION\_NODE

{

DWORD m\_pNextNode;

DWORD m\_pPreviousNode;

DWORD m\_pfnVectoredHandler;

}

(3)可以注册多个V.E.H，V.E.H结构体之间串成双向链表，可以看到结构体里面有两个指针，比S.E.H多了一个前向指针

(4)调试器处理优先级高于V.E.H，V.E.H优先级高于S.E.H，也就是说呢，KiUserExceptionDispatcher()先检查是不是被调试了，然后检查V.E.H，最后检查S.E.H链表

(5)V.E.H注册位置可以指定，S.E.H注册位置不能指定

(6)V.E.H保存在堆中

(7)最后呢，unwind操作只会对S.E.H产生作用，不会影响V.E.H

如果能在堆溢出的时候使用DWORD SHOOT来修改V.E.H头节点的指针，在处理异常的时候就能够执行shellcode

2.攻击T.E.B中的S.E.H头节点

前面有说，异常处理机制会遍历S.E.H链表 ，而第一个S.E.H指针在TEB第一个DWORD位置，而且永远指向离栈顶最近的S.E.H

那么如果能用DWORD SHOOT来修改这个指针，使其指向shellcode的起始位置，就可以执行shellcode

一个进程可能有多个线程，每个线程有一个线程环境块T.E.B，第一个T.E.B开始于地址0x7FFDE000，然后接下来每一个线程的T.E.B以间隔0x1000字节的规则向内存低址增长，如果一个线程退出，那么对应的T.E.B也会被销毁，空间会被以后的线程重复使用

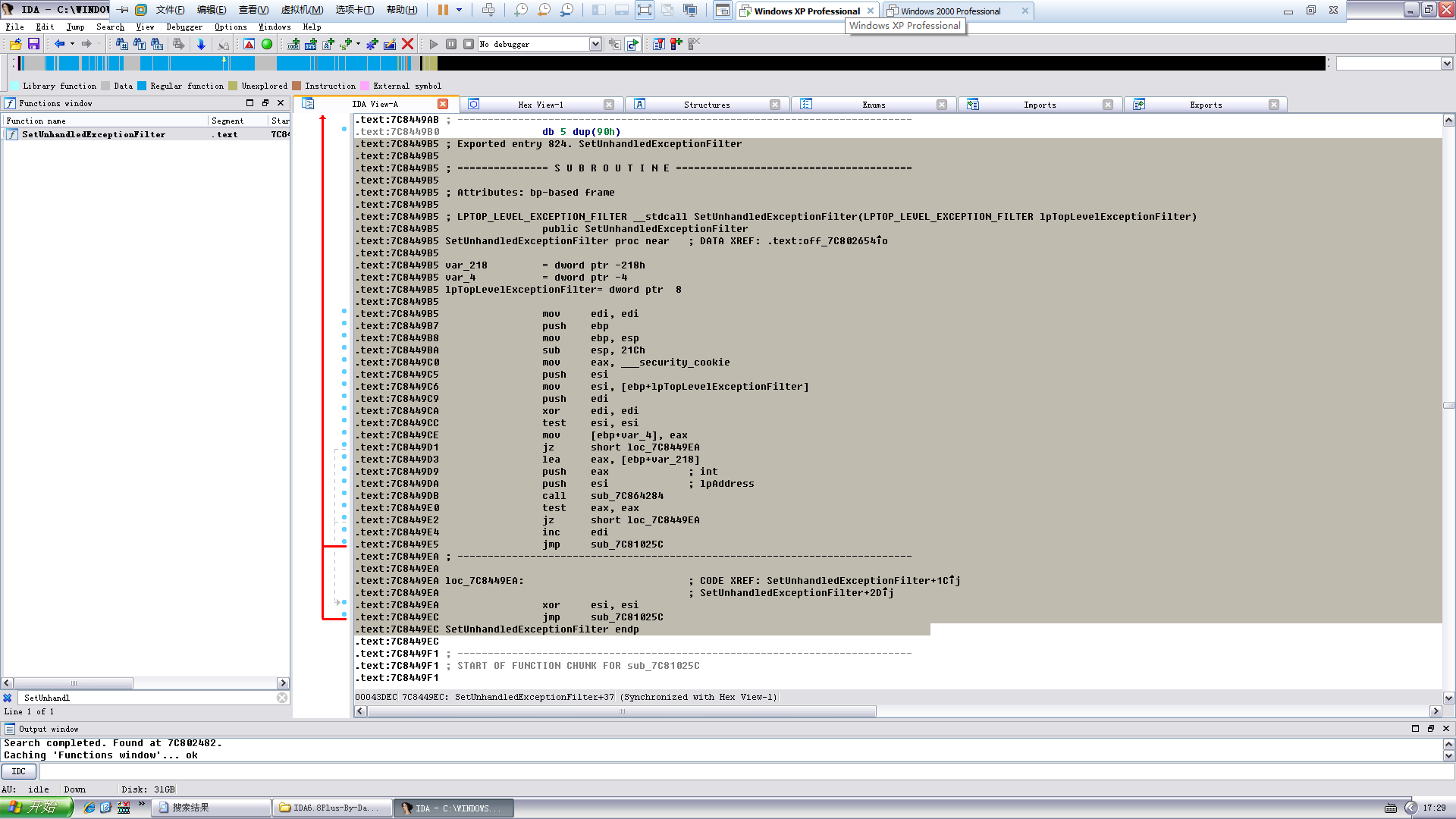
当遇到多线程的程序时，那么对应的T.E.B在什么位置就不好说了，这也是一个局限

3.攻击U.E.F

U.E.F(UnhandledExceptionFilter())，系统默认的异常处理函数，是系统处理异常的最后一个环节，当我们构造的异常所有异常处理都处理不了，那么系统就会调用这个异常处理，我们只需要利用DWORD SHOOT来修改这个异常的调用句柄，就可以执行shellcode

但是不同的系统这个句柄也不愿意，所以我们来反汇编一下kernel32.dll

小书包里掏出IDA，搜SetUnhandledExceptionFilter()



4.攻击P.E.B中的函数指针

前面有一章我们使用ExitProcess()来结束进程，ExitProcess()在结束进程时会调用临界区的RtlEnterCriticalSection()和RtlLeaveCriticalSection()

而ExitProcess()调用这两个函数的时候是通过调用P.E.B里的两个指针，所以呢，我们可以使用DWORD SHOOT来修改这对指针，改为shellcode的入口就可以了，而且这种办法比T.E.B可靠的多

参考

[VEH,VCH,UEF Windows向量化异常处理机制详解](http://blog.csdn.net/mycsersoft/article/details/33307655)

http://blog.csdn.net/mycsersoft/article/details/33307655