2021/06/02_Windows64位内核_第3课_强制结束进程

笔记本：	Windows64位内核
创建时间：	2021/6/2 星期三 15:11
作者：	ileemi

课前会议

Windbg高版本调试操作系统时不显示寄存器值解决办法：

- 将补丁文件 `wingdbg.dll` 拷贝到 wingdbg 对应的版本文件夹下
- 执行命令 `!wingdbg.regfix` 后重新打开寄存器窗口即可

驱动框架

磁盘驱动无法分辨对应的磁盘号（驱动名以HarddiskVolume1~4为例）。

文件过滤驱动：

- sfilter驱动：**IoRegisterFsRegistrationChange -- 注册文件系统筛选器驱动程序的通知例程，以便在文件系统将自身注册或注销为活动文件系统时调用该例程。**
  调用API注册回调，当有新设备接入，会通过回调进行通知。
- 文件驱动框架：Mini-filter（VS框架工程）

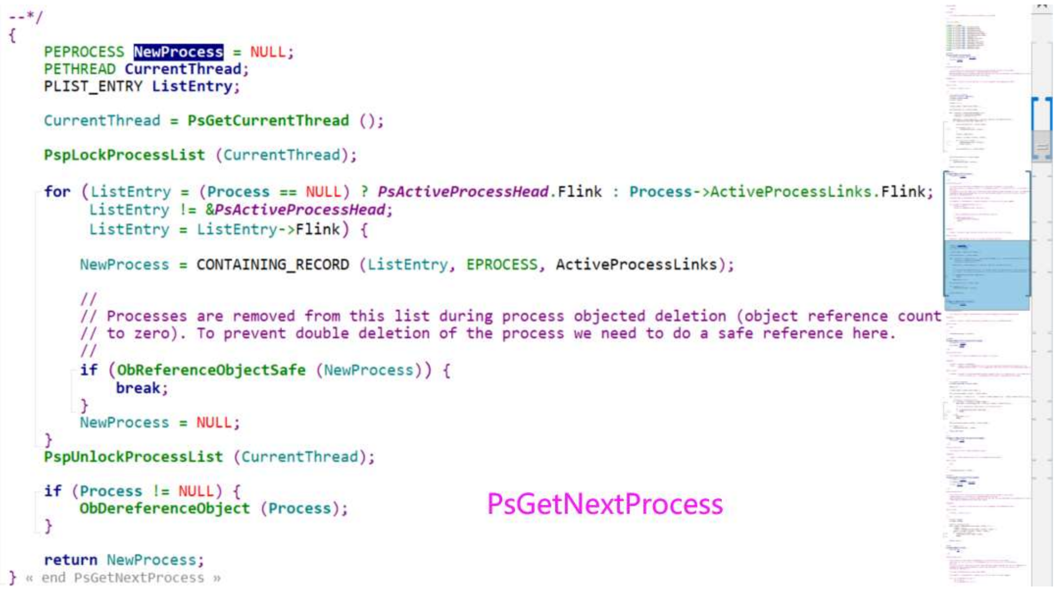
网络过滤驱动（网络防火墙 HIPS）：

- TDI层 传输（可以判断进程）：\device\tcp、\device\udp、\device\rawip
- NDIS层（无法判断进程）：底层，微软有提供对应的驱动框架，可参考WinDDK中的网络部分源码（src\network\ndis），微软官方示例：https://github.com/Microsoft/Windows-driver-samples
- 网卡驱动
- WFP：简单的网络监控框架，微软官方示例：https://github.com/Microsoft/Windows-driver-samples/tree/master/network/trans

进程隐藏

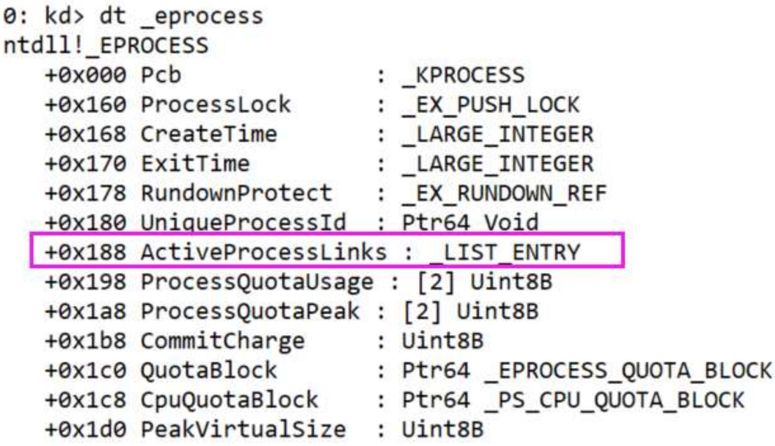
方法：EPROCESS脱链

Process32Next
NtQuerySystemInformation
ExpGetProcessInformation
PsGetNextProcess -- 通过 EPROCESS 的 ActiveProcessLinks（类型 LIST_ENTRY） 成员进行遍历。



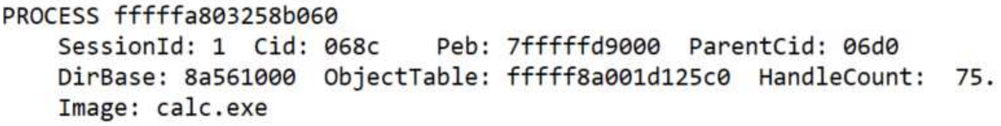
```
--*/
{
    PEPROCESS NewProcess = NULL;
    PETHREAD CurrentThread;
    PLIST_ENTRY ListEntry;

    CurrentThread = PsGetCurrentThread ();

    PspLockProcessList (CurrentThread);

    for (ListEntry = (Process == NULL) ? PsActiveProcessHead.Flink : Process->ActiveProcessLinks.Flink;
         ListEntry != &PsActiveProcessHead;
         ListEntry = ListEntry->Flink) {

        NewProcess = CONTAINING_RECORD (ListEntry, EPROCESS, ActiveProcessLinks);

        //
        // Processes are removed from this list during process objected deletion (object reference count
        // to zero). To prevent double deletion of the process we need to do a safe reference here.
        //
        if (ObReferenceObjectSafe (NewProcess)) {
            break;
        }
        NewProcess = NULL;
    }
    PspUnlockProcessList (CurrentThread);

    if (Process != NULL) {
        ObDereferenceObject (Process);
    }

    return NewProcess;
} « end PsGetNextProcess »
```

PsGetNextProcess

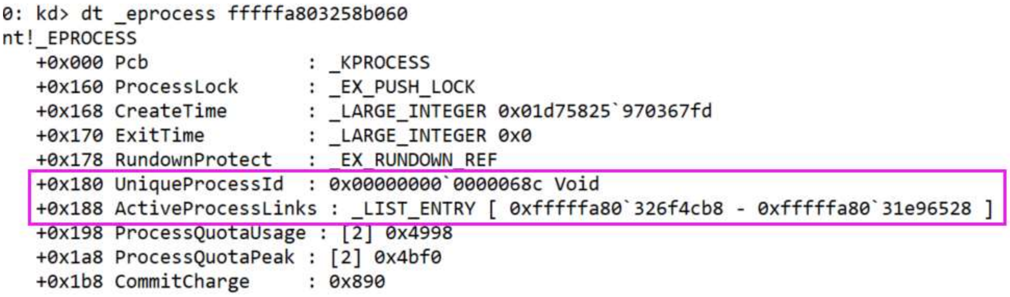


```
0: kd> dt _eprocess
ntdll!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x160 ProcessLock      : _EX_PUSH_LOCK
   +0x168 CreateTime       : _LARGE_INTEGER
   +0x170 ExitTime         : _LARGE_INTEGER
   +0x178 RundownProtect   : _EX_RUNDOWN_REF
   +0x180 UniqueProcessId  : Ptr64 Void
   +0x188 ActiveProcessLinks : _LIST_ENTRY
   +0x198 ProcessQuotaUsage : [2] Uint8B
   +0x1a8 ProcessQuotaPeak : [2] Uint8B
   +0x1b8 CommitCharge     : Uint8B
   +0x1c0 QuotaBlock       : Ptr64 _EPROCESS_QUOTA_BLOCK
   +0x1c8 CpuQuotaBlock    : Ptr64 _PS_CPU_QUOTA_BLOCK
   +0x1d0 PeakVirtualSize  : Uint8B
```

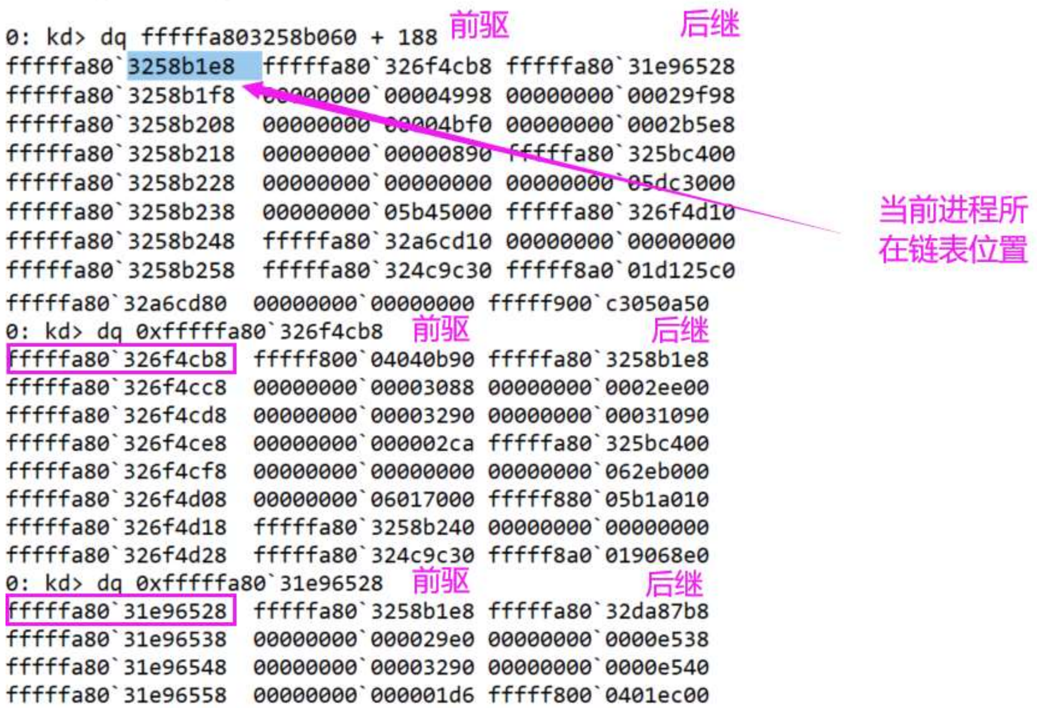
定位目标进程的EPROCESS：



```
PROCESS fffffa803258b060
    SessionId: 1  Cid: 068c    Peb: 7fffffd9000  ParentCid: 06d0
    DirBase: 8a561000  ObjectTable: fffff8a001d125c0  HandleCount:  75.
    Image: calc.exe
```



```
0: kd> dt _eprocess fffffa803258b060
nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x160 ProcessLock      : _EX_PUSH_LOCK
   +0x168 CreateTime       : _LARGE_INTEGER 0x01d75825`970367fd
   +0x170 ExitTime         : _LARGE_INTEGER 0x0
   +0x178 RundownProtect   : _EX_RUNDOWN_REF
   +0x180 UniqueProcessId  : 0x00000000`0000068c Void
   +0x188 ActiveProcessLinks : _LIST_ENTRY [ 0xfffffa80`326f4cb8 - 0xfffffa80`31e96528 ]
   +0x198 ProcessQuotaUsage : [2] 0x4998
   +0x1a8 ProcessQuotaPeak : [2] 0x4bf0
   +0x1b8 CommitCharge     : 0x890
```

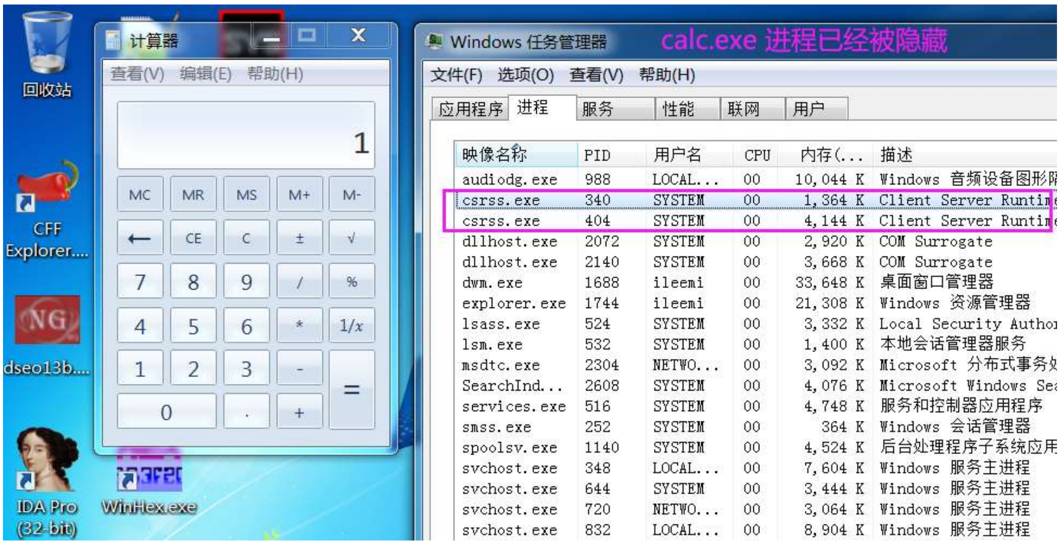


```
0: kd> dq fffffa803258b060 + 188
fffffa80`3258b1e8  fffffa80`326f4cb8 fffffa80`31e96528
fffffa80`3258b1f8  00000000`00004998 00000000`00029f98
fffffa80`3258b208  00000000`00004bf0 00000000`0002b5e8
fffffa80`3258b218  00000000`00000890 fffffa80`325bc400
fffffa80`3258b228  00000000`00000000 00000000`05dc3000
fffffa80`3258b238  00000000`05b45000 fffffa80`326f4d10
fffffa80`3258b248  fffffa80`32a6cd10 00000000`00000000
fffffa80`3258b258  fffffa80`324c9c30 fffff8a0`01d125c0
fffffa80`32a6cd80  00000000`00000000 fffff900`c3050a50
0: kd> dq 0xfffffa80`326f4cb8
fffffa80`326f4cb8  fffff800`04040b90 fffffa80`3258b1e8
fffffa80`326f4cc8  00000000`00003088 00000000`0002ee00
fffffa80`326f4cd8  00000000`00003290 00000000`00031090
fffffa80`326f4ce8  00000000`000002ca fffffa80`325bc400
fffffa80`326f4cf8  00000000`00000000 00000000`062eb000
fffffa80`326f4d08  00000000`06017000 fffff880`05b1a010
fffffa80`326f4d18  fffffa80`3258b240 00000000`00000000
fffffa80`326f4d28  fffffa80`324c9c30 fffff8a0`019068e0
0: kd> dq 0xfffffa80`31e96528
fffffa80`31e96528  fffffa80`3258b1e8 fffffa80`32da87b8
fffffa80`31e96538  00000000`000029e0 00000000`0000e538
fffffa80`31e96548  00000000`00003290 00000000`0000e540
fffffa80`31e96558  00000000`000001d6 fffff800`0401ec00
```

```
prev
0: kd> dq 0xfffffa80`326f4cb8
fffffa80`326f4cb8  fffff800`04040b90 fffffa80`3258b1e8

current
0: kd> dq fffffa803258b060 + 188
fffffa80`3258b1e8  fffffa80`326f4cb8 fffffa80`31e96528

next
0: kd> dq 0xfffffa80`31e96528
fffffa80`31e96528  fffffa80`3258b1e8 fffffa80`32da87b8
```

> 隐藏当前进程：

```
prev
0: kd> dq 0xfffffa80`326f4cb8
fffffa80`326f4cb8  fffff800`04040b90 fffffa80`31e96528

next
0: kd> dq 0xfffffa80`31e96528
fffffa80`31e96528  fffffa80`326f4cb8 fffffa80`32da87b8

// Windbg 执行以下命令
eq 0xfffffa80`326f4cc0 fffffa80`31e96528
eq 0xfffffa80`31e96528 fffffa80`326f4cb8
g
```

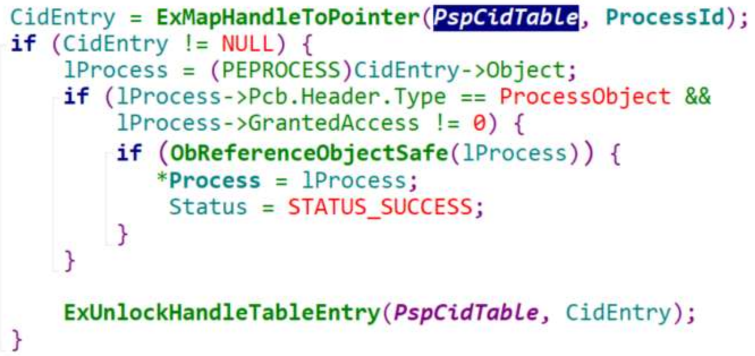


calc.exe 进程已经被隐藏

映像名称	PID	用户名	CPU	内存(...	描述
audiodg.exe	988	LOCAL...	00	10,044 K	Windows 音频设备图形
csrss.exe	340	SYSTEM	00	1,364 K	Client Server Runtim
csrss.exe	404	SYSTEM	00	4,144 K	Client Server Runtim
dllhost.exe	2072	SYSTEM	00	2,920 K	COM Surrogate
dllhost.exe	2140	SYSTEM	00	3,668 K	COM Surrogate
dwm.exe	1688	ileemi	00	33,648 K	桌面窗口管理器
explorer.exe	1744	ileemi	00	21,308 K	Windows 资源管理器
lsass.exe	524	SYSTEM	00	3,332 K	Local Security Autho
lsm.exe	532	SYSTEM	00	1,400 K	本地会话管理器服务
msdtc.exe	2304	NETWO...	00	3,092 K	Microsoft 分布式事务
SearchInd...	2608	SYSTEM	00	4,076 K	Microsoft Windows Se
services.exe	516	SYSTEM	00	4,748 K	服务和控制器应用程序
smss.exe	252	SYSTEM	00	364 K	Windows 会话管理器
spoolsv.exe	1140	SYSTEM	00	4,524 K	后台处理程序子系统应用
svchost.exe	348	LOCAL...	00	7,604 K	Windows 服务主进程
svchost.exe	644	SYSTEM	00	3,444 K	Windows 服务主进程
svchost.exe	720	NETWO...	00	3,064 K	Windows 服务主进程
svchost.exe	832	LOCAL...	00	8,904 K	Windows 服务主进程

目标进程虽然被隐藏了，但是进程相关功能依然可以正常使用，也就是进程对象依然存在，进程相关线程还在运行，线程中保存了进程对象信息，通过遍历线程依然可以遍历出被隐藏的进程。

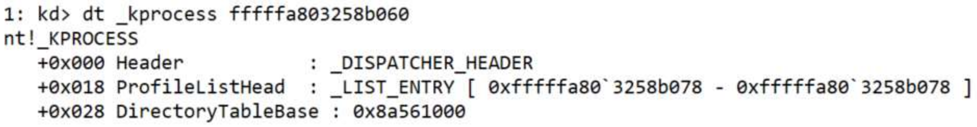
NtOpenProcess 函数内部 通过ID号遍历线程（PsLookupProcessThreadByCid）、通过id号遍历进程（PsLookupProcessByProcessId）。
PsLookupProcessByProcessId 函数内部通过PID查询进程对象。



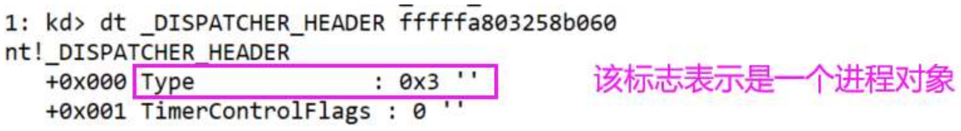
```
CidEntry = ExMapHandleToPointer(PspCidTable, ProcessId);
if (CidEntry != NULL) {
    lProcess = (PEPROCESS)CidEntry->Object;
    if (lProcess->Pcb.Header.Type == ProcessObject &&
        lProcess->GrantedAccess != 0) {
        if (ObReferenceObjectSafe(lProcess)) {
           *Process = lProcess;
            Status = STATUS_SUCCESS;
        }
    }

    ExUnlockHandleTableEntry(PspCidTable, CidEntry);
}
```

```
PHANDLE_TABLE PspCidTable; // nonpaged
```

- 暴力搜索内存：搜索整个内核内存，搜索KPROCESS、EPROCESS结构中固定的特征。



```
1: kd> dt _kprocess fffffa803258b060
nt!_KPROCESS
   +0x000 Header           : _DISPATCHER_HEADER
   +0x018 ProfileListHead  : _LIST_ENTRY [ 0xfffffa80`3258b078 - 0xfffffa80`3258b078 ]
   +0x028 DirectoryTableBase : 0x8a561000
```

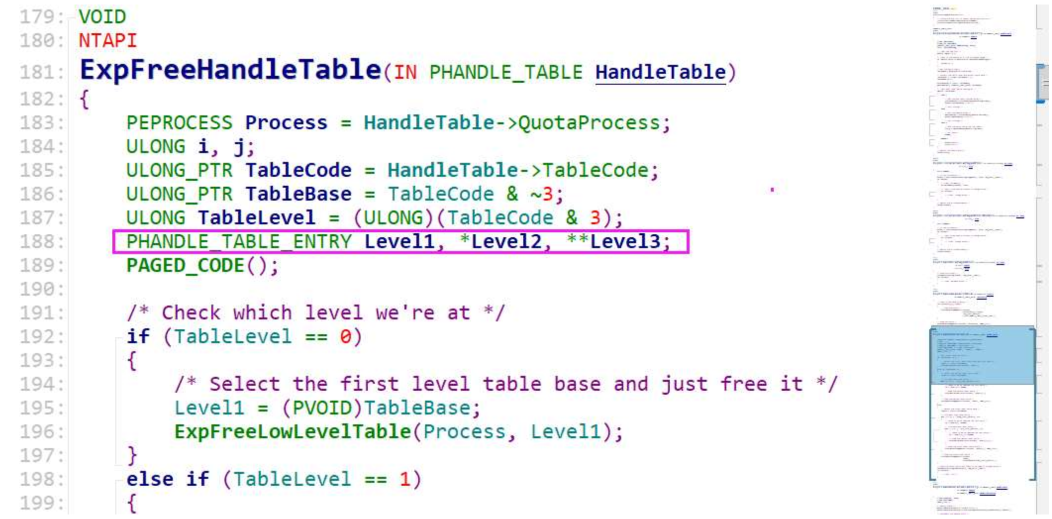


```
1: kd> dt _DISPATCHER_HEADER fffffa803258b060
nt!_DISPATCHER_HEADER
   +0x000 Type             : 0x3 ''
   +0x001 TimerControlFlags : 0 ''
```

该标志表示是一个进程对象

- 暴力结束隐藏的进程。

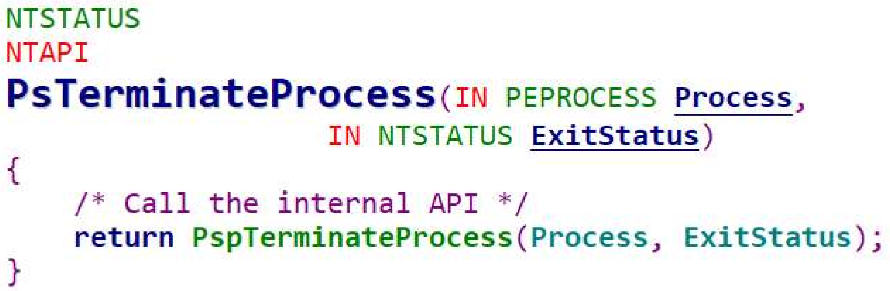
ExpFreeHandleTable



```
VOID
NTAPI
ExpFreeHandleTable(IN PHANDLE_TABLE HandleTable)
{
    PEPROCESS Process = HandleTable->QuotaProcess;
    ULONG i, j;
    ULONG_PTR TableCode = HandleTable->TableCode;
    ULONG_PTR TableBase = TableCode & ~3;
    ULONG TableLevel = (ULONG)(TableCode & 3);
    PHANDLE_TABLE_ENTRY Level1, *Level2, **Level3;
    PAGED_CODE();

    /* Check which level we're at */
    if (TableLevel == 0)
    {
        /* Select the first level table base and just free it */
        Level1 = (PVOID)TableBase;
        ExpFreeLowLevelTable(Process, Level1);
    }
    else if (TableLevel == 1)
    {
```

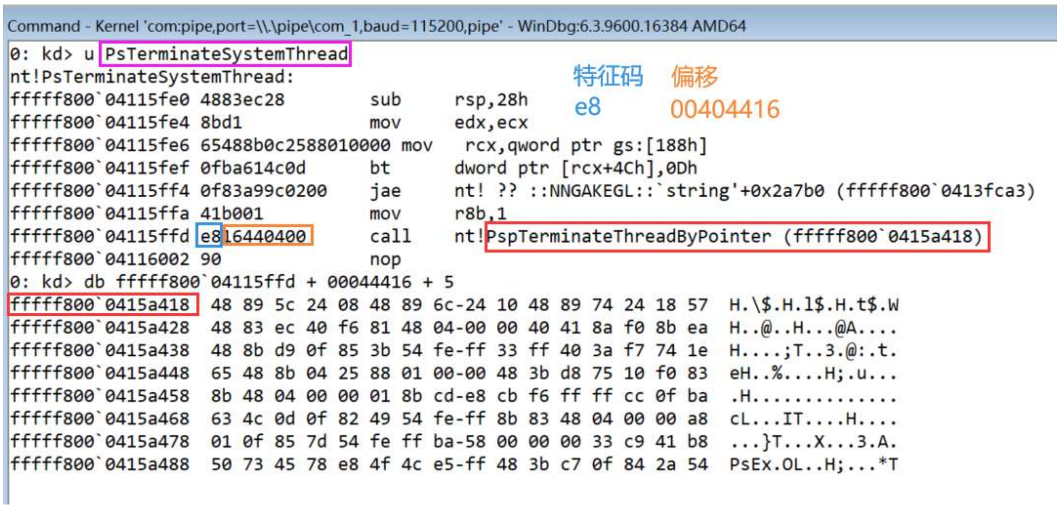
强制结束进程

将目标进程的Ring3层内存置0（手动制造异常，时程序崩溃），这种做法一般不推荐，存在隐患。
定位内核中结束进程的API：PsTerminateProcess --> PspTerminateProcess



```
NTSTATUS
NTAPI
PsTerminateProcess(IN PEPROCESS Process,
                   IN NTSTATUS ExitStatus)
{
    /* Call the internal API */
    return PspTerminateProcess(Process, ExitStatus);
}
```

PsTerminateSystemThread --> PspTerminateThreadByPointer（较为底层API，用来结束主线程）。通过调用更底层的函数，搜索特征码：



Command - Kernel 'com:pipe,port=\\.\pipe\com_1,baud=115200,pipe' - WinDbg:6.3.9600.16384 AMD64

```
0: kd> u PsTerminateSystemThread
nt!PsTerminateSystemThread:
fffff800`04115fe0 4883ec28         sub     rsp,28h
fffff800`04115fe4 8bd1             mov     edx,ecx
fffff800`04115fe6 65488b0c2588010000 mov    rcx,qword ptr gs:[188h]
fffff800`04115fef 0fba614c0d       bt      dword ptr [rcx+4Ch],0Dh
fffff800`04115ff4 0f83a99c0200     jae     nt! ?? ::NNGAKEGL::`string'+0x2a7b0 (fffff800`0413fca3)
fffff800`04115ffa 41b001           mov     r8b,1
fffff800`04115ffd e816440400       call    nt!PspTerminateThreadByPointer (fffff800`0415a418)
fffff800`04116002 90               nop
0: kd> db fffff800`04115ffd + 00044416 + 5
fffff800`0415a418  48 89 5c 24 08 48 89 6c-24 10 48 89 74 24 18 57  H.\$.H.l$.H.t$.W
fffff800`0415a428  48 83 ec 40 f6 81 48 04-00 00 40 41 8a f0 8b ea  H..@..H...@A....
fffff800`0415a438  48 8b d9 0f 85 3b 54 fe-ff 33 ff 40 3a f7 74 1e  H....;T..3.@:.t.
fffff800`0415a448  65 48 8b 04 25 88 01 00-00 48 3b d8 75 10 f0 83  eH..%....H;.u...
fffff800`0415a458  8b 48 04 00 00 01 8b cd-e8 cb f6 ff ff cc 0f ba  .H..............
fffff800`0415a468  63 4c 0d 0f 82 49 54 fe-ff 8b 83 48 04 00 00 a8  cL...IT....H....
fffff800`0415a478  01 0f 85 7d 54 fe ff ba-58 00 00 00 33 c9 41 b8  ...}T...X...3.A.
fffff800`0415a488  50 73 45 78 e8 4f 4c e5-ff 48 3b c7 0f 84 2a 54  PsEx.OL..H;...*T
```

PsLookupProcessByProcessId：通常进程ID，获取对应的 EPROCES 结构。
PsLookupThreadByThreadId：通过线程id返回到线程的 EPROCESS 结构。
NtQueryInformationProcess：查询进程的各种信息。
IoThreadToProcess：通过线程信息获取主进程信息。
PspCidTable

代码示例

```
extern "C" {

#include <ntddk.h>
#define DEVICE_NAME L"\\Device\\MyKeyboard"
typedef NTSTATUS (*PSP_TERMINATE_THREAD_BY_POINTER)(
        IN PETHREAD Thread,
        IN NTSTATUS ExitStatus,
        IN BOOLEAN DirectTerminate);
NTSTATUS PsLookupProcessByProcessId(
        __in HANDLE ProcessId,
        __deref_out PEPROCESS* Process);
VOID Unload(__in struct _DRIVER_OBJECT* DriverObject) {
        UNREFERENCED_PARAMETER(DriverObject);
        DbgPrint("[51asm] Unload\n");
}
// 结束进程
NTSTATUS PsTerminateProcess(
        PEPROCESS Process,
        NTSTATUS Status
);
// 通过线程id返回到线程的 EPROCESS 结构
NTSTATUS PsLookupThreadByThreadId(
        __in HANDLE ThreadId,
        __deref_out PETHREAD* Thread
);
PEPROCESS IoThreadToProcess(IN PETHREAD Thread);
// 强制结束进程
void MyTerminateProcess(HANDLE ProcessId) {
        // 获取API PsTerminateSystemThread 地址
        PVOID pfnPsTerminateSystemThread = PsTerminateSystemThread;

        // 搜索特征码 PspTerminateThreadByPointer -- 0xE8
        unsigned char* pCode = (unsigned char*)pfnPsTerminateSystemThr

        // 保存 PspTerminateThreadByPointer API地址
        PSP_TERMINATE_THREAD_BY_POINTER PspTerminateThreadByPointer = NU

        while (TRUE) {
                if (*pCode == 0xE8) {
                        // 特征码匹配
                        // 获取 PspTerminateThreadByPointer 地址
                        PspTerminateThreadByPointer = (PSP_TERMINATE_THRE
(pCode + *(int*)(pCode + 1) + 5);
                        break;
                }
                pCode++;
        }
        if (PspTerminateThreadByPointer != NULL) {
                DbgPrint("PspTerminateThreadByPointer:%p\n",
                        PspTerminateThreadByPointer);
                // 获取EPROCESS
                PEPROCESS Process = NULL;
                NTSTATUS Status;
                Status = PsLookupProcessByProcessId(ProcessId, &Process)

                DbgPrint("Process:%p\n", Process);
                if (NT_SUCCESS(Status)) {
                        // 遍历该进程的所有线程并且结束
                        for (unsigned int i = 0; i < 0xffffff; i

                                PETHREAD Thread; // 保存线程对象
                                Status = PsLookupThreadByThreadId((HANDLE

                                if (NT_SUCCESS(Status)) {
                                        if (IoThreadToProcess(Thread) ==

                                                DbgPrint("Thread:%p\n", Th

                                                // 结束当前线程
                                                (*PspTerminateThreadByPoint
(Thread, 0, TRUE);

                                                DbgPrint("PspTerminateThrea

                                        }
                                }
                        }
                }
        }
        else {
                DbgPrint("PspTerminateThreadByPointer == NULL\n");
        }
}
NTSTATUS DriverEntry(
        __in PDRIVER_OBJECT DriverObject,
        __in PUNICODE_STRING RegistryPath)
{
        UNREFERENCED_PARAMETER(RegistryPath);
        // 注册卸载函数
        DriverObject->DriverUnload = Unload;
        // 强制结束进程
        MyTerminateProcess((HANDLE)3504);
        // 隐藏进程
        // NtQuerySystemInformation
        return  STATUS_SUCCESS;
}


}
```

0603 课后会议

OpenARK
WKE工具

蓝色药丸（blue pill）：病毒，利用VT技术