2021/06/01_Windows64位内核_第2课_64位IDT表、Hook驱动的方法

笔记本： Windows64位内核
创建时间： 2021/6/1 星期二 15:37
作者： ileemi

vt模式

hook

SSDT、GDT表不能hook，IDT表可以hook。

hook键盘驱动的缺点：

64位IDT表

> 中断门和陷阱门的长度为16个字节，以便为指令指针(RIP)提供64位的偏移量。由中断门描述符引用的64位RIP允许中断服务例程位于线性地址空间中的任何位置。

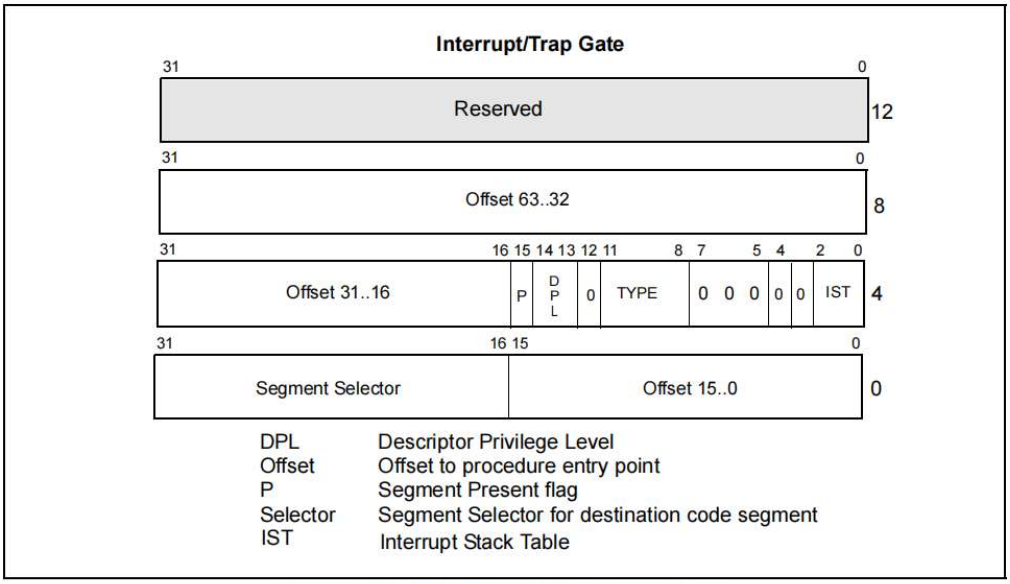
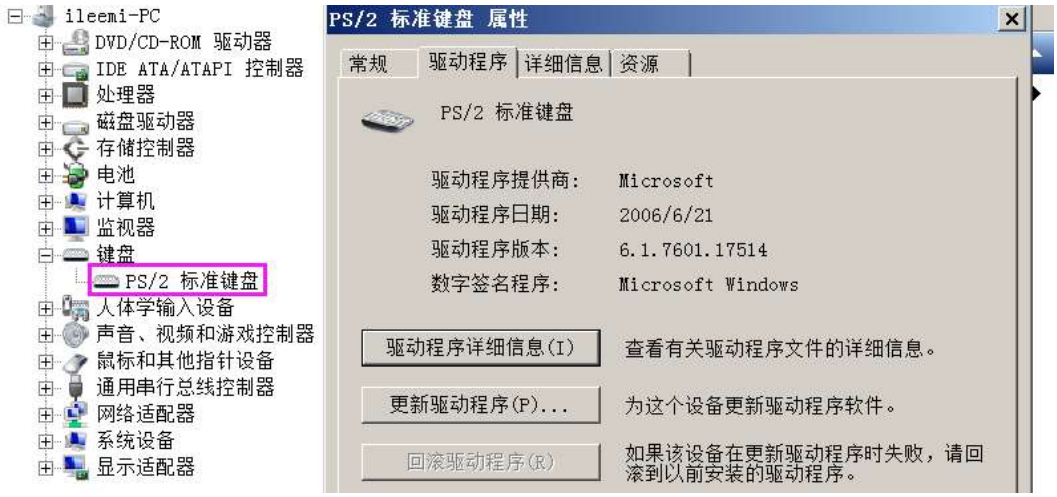


Figure 6-8. 64-Bit IDT Gate Descriptors

idtr：物理地址48位，高16位为符号扩展。

```
0: kd> r idtr
idtr=fffff80000b95080
0: kd> dq fffff80000b95080
fffff800`00b95080  040e8e00`001033c0 00000000`fffff800
fffff800`00b95090  040e8e00`001034c0 00000000`fffff800
fffff800`00b950a0  040e8e03`00103680 00000000`fffff800
fffff800`00b950b0  040eee00`00103a00 00000000`fffff800
fffff800`00b950c0  040eee00`00103b00 00000000`fffff800
fffff800`00b950d0  040e8e00`00103c00 00000000`fffff800
fffff800`00b950e0  040e8e00`00103d00 00000000`fffff800
fffff800`00b950f0  040e8e00`00103f40 00000000`fffff800
0: kd> u fffff800040e33c0
nt!KeSynchronizeExecution+0x1530:
fffff800`040e33c0 4883ec08        sub     rsp,8
fffff800`040e33c4 55              push    rbp
fffff800`040e33c5 4881ec58010000  sub     rsp,158h
fffff800`040e33cc 488dac2480000000 lea     rbp,[rsp+80h]
fffff800`040e33d4 c645ab01        mov     byte ptr [rbp-55h],1
fffff800`040e33d8 488945b0        mov     qword ptr [rbp-50h],rax
fffff800`040e33dc 48894db8        mov     qword ptr [rbp-48h],rcx
fffff800`040e33e0 488955c0        mov     qword ptr [rbp-40h],rdx
```

32位操作系统键盘中断在IDT表的93项（PS/2 键盘其中断一般都在IDT表的93项）。



驱动对象 hook

不需要考虑硬件问题，兼容性更好的hook方式：
硬件都有对应的驱动（一定存在），其驱动中都需要注册对应的回调函数（在驱动对象中），定位硬件对应的驱动对象进行替换。

查询硬件对应的驱动

> C:\Windows\System32\drivers

- 键盘驱动 kbdclass.sys
- 磁盘驱动 ntfs.sys fat32.sys exfat.sys
- 网络驱动 tcpip.sys tdi.sys(底层) ndis.sys(底层)

Hook键盘驱动程序编写较为麻烦且无法处理异步问题。网络发包不走SSDT，直接走驱动。三环hook网络连接时容易被检测，只有在Ring0层进行hook（进行网卡监控），其才能接收到网络上大部分数据包（wireshark -- hook ndis.sys）

获取驱动（设备）对象

> API：IoGetDeviceObjectPointer -- ntoskrnl.exe

通过设备名称获取设备对象，通过设备对象获取对应的驱动对象。

> 硬件对应设备名称的获取可通过逆向的方式获取:

- 驱动创建时会调用IoCreateDevice函数，会传递对应的设备名（查找引用）
- 使用WinObj工具查看硬件设备对应的设备名，以键盘为例，如下图所示：

Hook驱动的方法

1. 将驱动的回调替换掉 -- Hook驱动的IRP，缺点：如果是异步操作，那么Hook的时候很大概率是拿不到数据的。
2. 分层驱动：在原有的顶层驱动上再加一层（拦截顶层的发送请求），即过滤驱动。
    - IoAttachDevice 附加到设备上，不管同步还是异步都可以收到数据
    - 如果在本层驱动完成了请求，那么下层驱动就收不到了
    - 需要设置一样的通信方式、跳过当前IRP堆栈 IoSkipCurrentIrpStackLocation、调下层驱 动 IoCallDriver
    - 拿信息需要注册请求完成的回调 IoSetCompletionRoutine ，请求完成了会调用此回调，且 在回调用要有引用计数，在驱动卸载的时候注销回调（先取消Hook驱动、判断所有派遣都没 人调用之后，在返回）
    - 如果下层有的派遣函数，上层也必须实现派遣（实现全部派遣，派发给下层，感兴趣的单独 实现）

相关API

> IoAttachDevice：将调用方的设备对象附加到一个命名的目标设备对象，以便为目标设备绑定的I/O请求首先路由到调用方。

```
NTSTATUS IoAttachDevice(
  // 指向调用者创建的设备对象的指针
  PDEVICE_OBJECT SourceDevice,
  // 指向缓冲区的指针，该缓冲区包含要附加指定SourceDevice的设备对象的名称
```

```
  PUNICODE_STRING TargetDevice,
  // 指向指针的调用者分配存储的指针。返回时，如果附加成功，则包含指向目标设备对象的指针
  PDEVICE_OBJECT *AttachedDevice
);
```

> IoDetachDevice：在调用方的设备对象和下一个驱动程序的设备对象之间释放一个附加设备。

```
void IoDetachDevice(
  // 指向下层驱动程序的设备对象的指针。调用方先前成功调用了
  // IoAttachDevice或IoAttachDeviceToDeviceStack以获取此指针
  PDEVICE_OBJECT TargetDevice
)
```

> IoAttachDeviceToDeviceStack：将调用者的设备对象附加到链中最高的设备对象，并返回指向先前最高的设备对象的指针。

```
PDEVICE_OBJECT IoAttachDeviceToDeviceStack(
  PDEVICE_OBJECT SourceDevice,
  PDEVICE_OBJECT TargetDevice
);
```

> IoSetCompletionRoutine：注册一个IO完成例程，这个完成例程将会在调用此函数的驱动的下一层驱动完成IRP指定的操作请求时被调用。

```
// 注册完成请求的回调
void IoSetCompletionRoutine(
  // 指向驱动程序正在处理的IRP的指针
  _In_ PIRP Irp,
  // 指定驱动程序提供的IoCompletion例程的入口点，它在下一个驱动程序完成包时被调用
  _In_opt_ PIO_COMPLETION_ROUTINE CompletionRoutine,
  // 指向由驱动程序决定的上下文的指针，以传递到IoCompletion例程
  _In_opt_ __drv_aliasesMem PVOID Context,
  // 指定在IRP的IO_STATUS_BLOCK结构中使用成功状态值完成IRP时是否调用完成例程
  _In_ BOOLEAN InvokeOnSuccess,
  // 指定如果IRP在IRP的IO_STATUS_BLOCK结构中以失败状态值完成
  _In_ BOOLEAN InvokeOnError,
  // 指定当驱动程序或内核调用IoCancelIrp来取消IRP时是否调用完成例程
  _In_ BOOLEAN InvokeOnCancel
);
```