模块加载包括用户层模块（.DLL）和内核模块（.SYS）的加载。传统方法要监控这两者加在必须 HOOK 好几个函数，比如 NtCreateSection 和 NtLoadDriver 等，而且这些方法还不能监控未知的驱动加载方法。其实为了监控模块加载而 HOOK API 是非常傻的，因为微软已经提供了一对标准的 API 实现此功能。它们分别是 PsSetLoadImageNotifyRoutine 和 PsRemoveLoadImageNotifyRoutine，可以设置/取消一个“映像加载通告例程”，当有驱动或者 DLL 被加载时，回调函数就会被调用。有人可能认为这个标准方法的监控非常表层，其实恰恰相反，这个方法非常底层，大部分隐秘的加载驱动的方法都可以绕过 NtLoadDriver，但是无法绕过“映像加载通告例程”。所以用此方法监控驱动加载是最合适的了。

首先看看此函数的原型：

```
//添加：
PsSetLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)LoadImageNotifyRoutine);
```

```
//删除：
PsRemoveLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)LoadImageNotifyRoutine);
```

其中 NotifyRoutine 是一个函数指针，此回调函数的原型是：

```
VOID (*PLOAD_IMAGE_NOTIFY_ROUTINE)
(
    __in_opt PUNICODE_STRING  FullImageName,
    __in HANDLE  ProcessId,
    __in PIMAGE_INFO  ImageInfo
);
```

回调函数的前两个参数都显而易见，分别是『映像的路径』和『加载此映像的进程 ID』，第三个参数包含了更加详细的信息：

```
typedef struct  _IMAGE_INFO {
    union {
        ULONG  Properties;
        struct {
            ULONG ImageAddressingMode  : 8; //code addressing mode
            ULONG SystemModeImage      : 1; //system mode image
            ULONG ImageMappedToAllPids : 1; //mapped in all processes
            ULONG Reserved             : 22;
        };
    };
    PVOID  ImageBase;
    ULONG  ImageSelector;
    ULONG  ImageSize;
    ULONG  ImageSectionNumber;
} IMAGE_INFO, *PIMAGE_INFO;
```

不过此结构体到了 VISTA 之后发生了一点变化：

```
typedef struct _IMAGE_INFO {
```

```
    union {
        ULONG Properties;
        struct {
            ULONG ImageAddressingMode  : 8;  // Code addressing mode
            ULONG SystemModeImage      : 1;  // System mode image
            ULONG ImageMappedToAllPids : 1;  // Image mapped into all processes
            ULONG ExtendedInfoPresent  : 1;  // IMAGE_INFO_EX available
            ULONG Reserved             : 21;
        };
    };
    PVOID       ImageBase;
    ULONG       ImageSelector;
    SIZE_T      ImageSize;
    ULONG       ImageSectionNumber;
} IMAGE_INFO, *PIMAGE_INFO;
```

当 ExtendedInfoPresent 标志非零时，IMAGE_INFO 结构体被包含在了另外一个更大的结构体里：

```
typedef struct _IMAGE_INFO_EX {
    SIZE_T              Size;
    IMAGE_INFO          ImageInfo;
    struct _FILE_OBJECT *FileObject;
} IMAGE_INFO_EX, *PIMAGE_INFO_EX;
```

不过这个变动跟实现监控驱动加载的关系不大，我们只需要 IMAGE_INFO 的信息即可实现监控驱动加载。接下来讲如何获得加载驱动的信息。之前说过，这个通告例程不仅仅管加载驱动，连进程加载 DLL 也管，那我们怎么判断到底是加载驱动还是加载 DLL 呢？如果说根据后缀名判断则很明显是一个挫方法。我的方法是，**根据回调函数 LoadImageNotifyRoutine 的第二个参数判断，如果 PID 是 0，则表示加载驱动，如果 PID 位非零，则表示加载 DLL。**原因很简单，我之前说过这个函数很底层，到了一定的深度之后就无法判断到底是谁主动引发的行为了，一切都是系统的行为。当然，你也可以认为这是通过回调来监控驱动加载的缺点。判断了是驱动后，就通过 ImageInfo->ImageBase 来获取驱动的映像基址。**如果不想让这个驱动加载，就通过 ImageBase 来获得 DriverEntry 的地址**（ImageBase 就是 DOS 头，根据 DOS 头找到 NT 头，然后在 NT 头的 OptionalHeader 里就能找到入口点了。入口点的数据就是 DriverEntry 的地址），**并写入“拒绝访问”的机器码即可**。

Mov eax,c0000022h	B8 22 00 00 C0
Ret	C3

实现的代码如下：

```
PVOID GetDriverEntryByImageBase(PVOID ImageBase)
{
    PIMAGE_DOS_HEADER pDOSHeader;
```

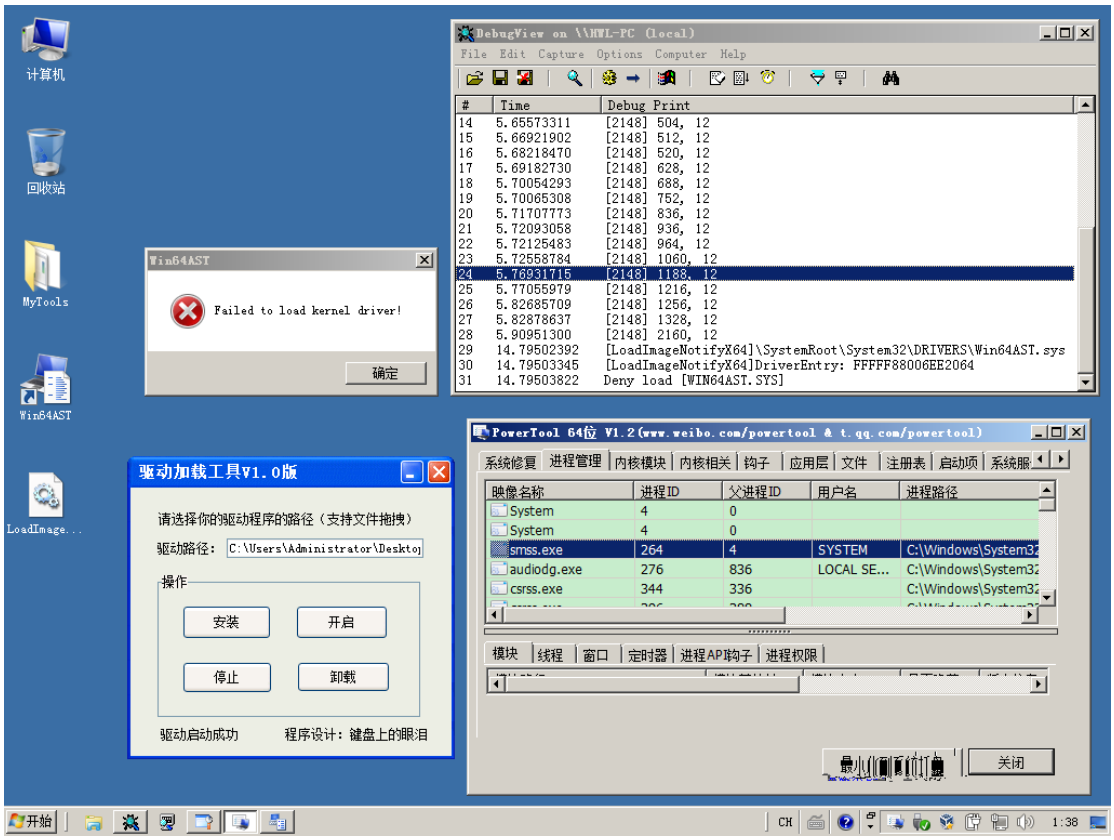
```
    PIMAGE_NT_HEADERS64 pNTHeader;
    PVOID pEntryPoint;
    pDOSHeader = (PIMAGE_DOS_HEADER)ImageBase;
    pNTHeader = (PIMAGE_NT_HEADERS64)((ULONG64)ImageBase + pDOSHeader->e_lfanew);
    pEntryPoint = (PVOID)((ULONG64)ImageBase + pNTHeader->OptionalHeader.AddressOfEntryPoint);
    return pEntryPoint;
}
```

```
void DenyLoadDriver(PVOID DriverEntry)
{
    UCHAR fuck[]="\xB8\x22\x00\x00\xC0\xC3";
    CopyMemory(DriverEntry,fuck,sizeof(fuck));
}
```

```
VOID LoadImageNotifyRoutine
(
    __in_opt PUNICODE_STRING  FullImageName,
    __in HANDLE  ProcessId,
    __in PIMAGE_INFO  ImageInfo
)
{
    PVOID pDrvEntry;
    char szFullImageName[260]={0};
    if(FullImageName!=NULL && MmIsAddressValid(FullImageName))
    {
        if(ProcessId==0)
        {
            DbgPrint("[LoadImageNotifyX64]%wZ\n",FullImageName);
            pDrvEntry=GetDriverEntryByImageBase(ImageInfo->ImageBase);
            DbgPrint("[LoadImageNotifyX64]DriverEntry: %p\n",pDrvEntry);
            UnicodeToChar(FullImageName,szFullImageName);
            if(strstr(_strlwr(szFullImageName),"win64ast.sys"))
            {
                DbgPrint("Deny load [WIN64AST.SYS]");
                //禁止加载 win64ast.sys
                DenyLoadDriver(pDrvEntry);
            }
        }
    }
}
```

有些人心中可能想问为什么拒绝加载驱动那里仍然是【mov eax,c000022h】而不【mov rax,c000022h】。这是因为 NTSTATUS 其实就是 long 的马甲，而 long 的长度在 WIN64 系统下依然是 4 字节而不是 8 字节，所以用【mov eax,立即数】足矣。最后实现的效果如下（监视所有的驱动加载并拒绝名为 win64ast.sys 的

驱动加载）：



课后作业：实现禁止指定名称的 DLL（例如 hook.dll）加载。