在笔者上一篇文章`《驱动开发：内核监视LoadImage映像回调》`中`LyShark`简单介绍了如何通过`PsSetLoadImageNotifyRoutine`函数注册回调来`监视驱动`模块的加载，注意我这里用的是`监视`而不是`监控`之所以是监视而不是监控那是因为`PsSetLoadImageNotifyRoutine`无法实现参数控制，而如果我们想要控制特定驱动的加载则需要自己做一些事情来实现，如下`LyShark`将解密如何实现屏蔽特定驱动的加载。

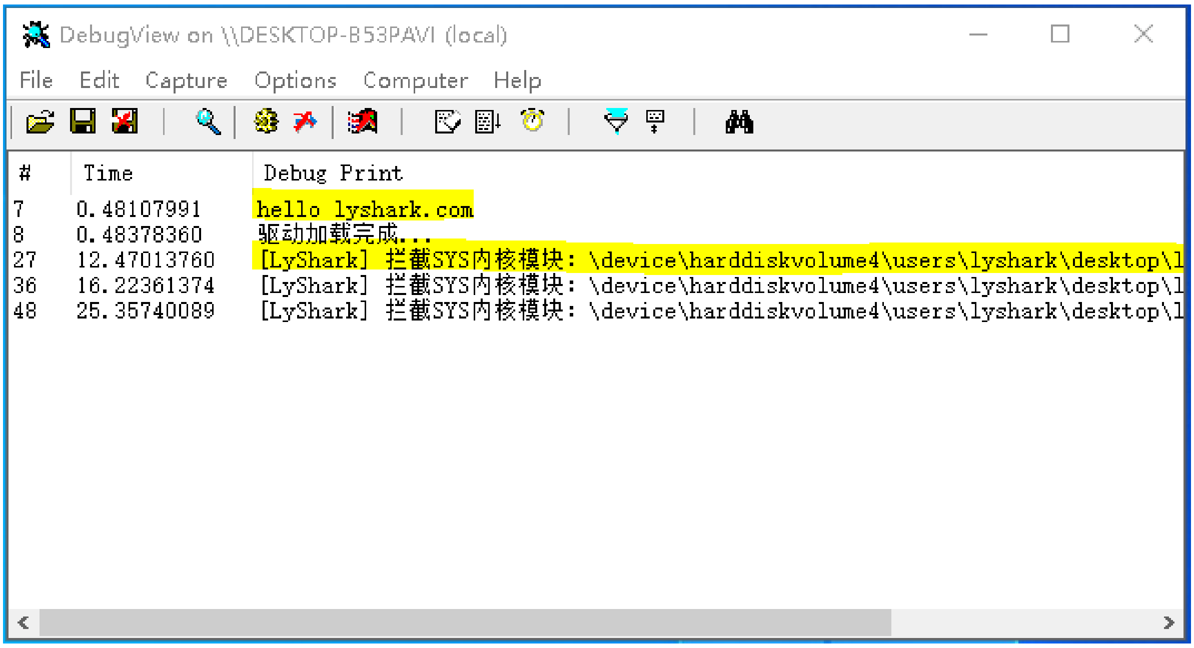
要想实现`驱动屏蔽`其原理很简单，通过`ImageInfo->ImageBase`得到镜像基地址，然后调用`GetDriverEntryByImageBase`函数来得到程序的入口地址，找NT头的`OptionalHeader`节点，该节点里面就是被加载驱动入口，通过汇编在驱动头部写入`ret`返回指令，即可实现屏蔽加载特定驱动文件。

原理其实很容易理解，如果我们需要实现则只需要在`《驱动开发：内核监视LoadImage映像回调》`这篇文章的代码上稍加改进即可，当检测到`lyshark.sys`驱动加载时，直接跳转到入口处快速写入一个`Ret`让驱动返回即可，至于如何写出指令的问题如果不懂建议回头看看`《驱动开发：内核CR3切换读写内存》`文章中是如何读写内存的，这段代码实现如下所示。

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com
#include <ntddk.h>
#include <intrin.h>
#include <ntimage.h>

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 UnicodeToChar(PUNICODE_STRING dst, char *src)
{
    ANSI_STRING string;
    RtlUnicodeStringToAnsiString(&string, dst, TRUE);
    strcpy(src, string.Buffer);
    RtlFreeAnsiString(&string);
}

// 使用开关写保护需要在[C/C++]->[优化]->启用内部函数
// 关闭写保护
KIRQL  WPOFFx64()
{
    KIRQL  irql = KeRaiseIrqlToDpcLevel();
    UINT64  cr0 = __readcr0();
    cr0 &= 0xfffffffffffeffff;
    _disable();
    __writecr0(cr0);
    return  irql;
}

// 开启写保护
void  WPONx64(KIRQL  irql)
{
    UINT64  cr0 = __readcr0();
    cr0 |= 0x10000;
    _enable();
    __writecr0(cr0);
    KeLowerIrql(irql);
}

BOOLEAN DenyLoadDriver(PVOID DriverEntry)
{
    UCHAR fuck[] = "\xB8\x22\x00\x00\xC0\xC3";
    KIRQL kirql;
    /* 在模块开头写入以下汇编指令
    Mov eax,c0000022h
    ret
    */
    if (DriverEntry == NULL) return FALSE;
    kirql = WPOFFx64();
    memcpy(DriverEntry, fuck, sizeof(fuck) / sizeof(fuck[0]));
    WPONx64(kirql);
    return TRUE;
}

VOID MyLySharkComLoadImageNotifyRoutine(PUNICODE_STRING FullImageName, HANDLE 
ModuleStyle, PIMAGE_INFO ImageInfo)
{
    PVOID pDrvEntry;
    char szFullImageName[256] = { 0 };

    // MmIsAddress 验证地址可用性
    if (FullImageName != NULL && MmIsAddressValid(FullImageName))
    {
        // ModuleStyle为零表示加载sys
        if (ModuleStyle == 0)
        {
            pDrvEntry = GetDriverEntryByImageBase(ImageInfo->ImageBase);
            UnicodeToChar(FullImageName, szFullImageName);
            if (strstr(_strlwr(szFullImageName), "lyshark.sys"))
            {
                DbgPrint("[LyShark] 拦截SYS内核模块：%s", szFullImageName);
                DenyLoadDriver(pDrvEntry);
            }
        }
    }
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
```

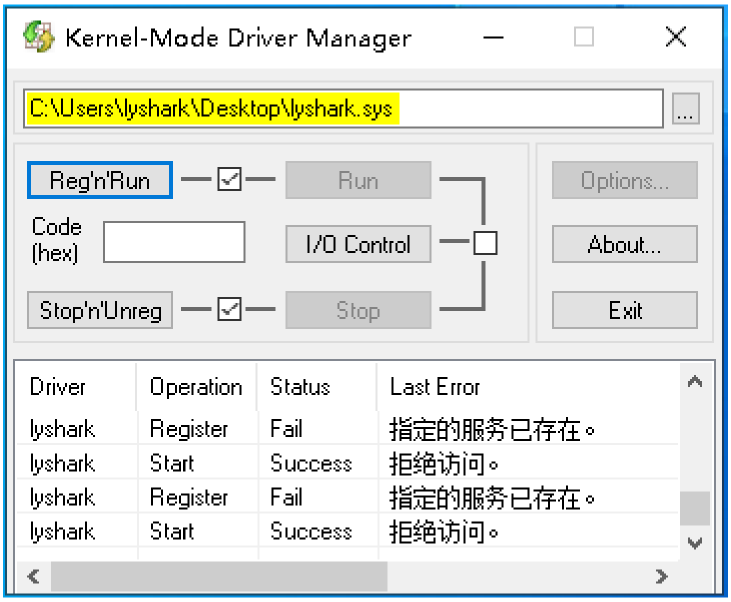
```
PsRemoveLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)MyLySharkComLoadImageNotifyRoutine);
    DbgPrint("驱动卸载完成...");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");


PsSetLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)MyLySharkComLoadImageNotifyRoutine);
    DbgPrint("驱动加载完成...");
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

首先运行我们的驱动，然后我们接着加载 `lyshark.sys` 则你会发现驱动被拦截了。



```
#   Time          Debug Print
7   0.48107991    hello lyshark.com
8   0.48378360    驱动加载完成...
27  12.47013760   [LyShark] 拦截SYS内核模块: \device\harddiskvolume4\users\lyshark\desktop\l
36  16.22361374   [LyShark] 拦截SYS内核模块: \device\harddiskvolume4\users\lyshark\desktop\l
48  25.35740089   [LyShark] 拦截SYS内核模块: \device\harddiskvolume4\users\lyshark\desktop\l
```

我们看下驱动加载器，提示的信息是拒绝访问，因为这个驱动其实是加载了的，只是入口处被填充了返回而已。



C:\Users\lyshark\Desktop\lyshark.sys

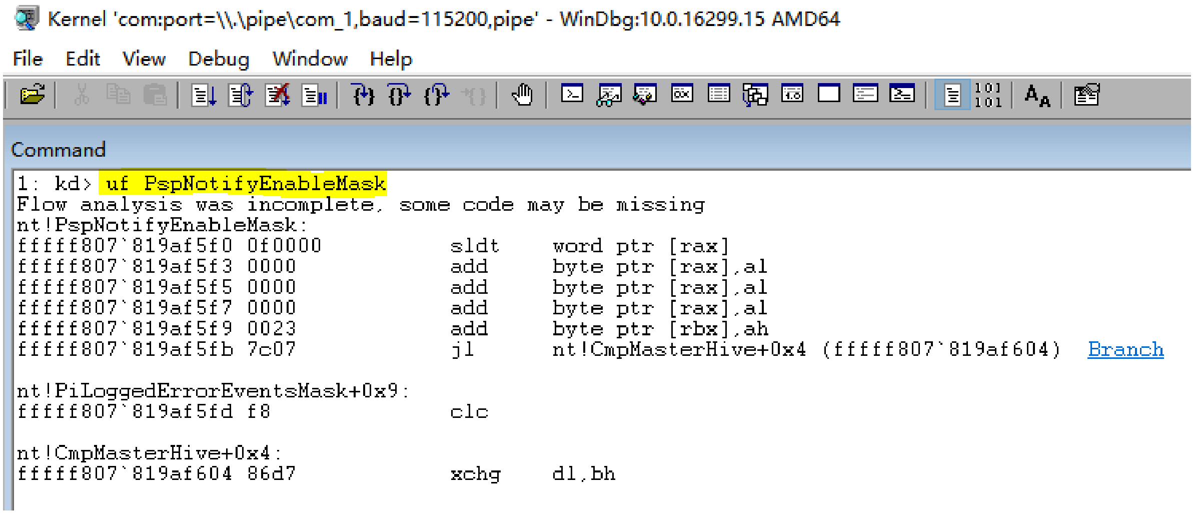
Driver	Operation	Status	Last Error
lyshark	Register	Fail	指定的服务已存在。
lyshark	Start	Success	拒绝访问。
lyshark	Register	Fail	指定的服务已存在。
lyshark	Start	Success	拒绝访问。

除了使用`Ret`强制返回的方法意外，屏蔽驱动加载还可以使用另一种方式实现禁用模块加载，例如当驱动被加载首先回调函数内可以接收到，当接收到以后直接调用`MmUnmapViewOfSection`函数强制卸载掉即可，如果使用这种方法实现则这段代码需要改进成如下样子。

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com
#include <ntifs.h>
#include <ntimage.h>
#include <intrin.h>

NTSTATUS MmUnmapViewOfSection(PEPROCESS Process, PVOID BaseAddress);
NTSTATUS SetNotifyRoutine();
NTSTATUS RemoveNotifyRoutine();

VOID LoadImageNotifyRoutine(PUNICODE_STRING FullImageName, HANDLE ProcessId,
PIMAGE_INFO ImageInfo);
NTSTATUS U2C(PUNICODE_STRING pustrSrc, PCHAR pszDest, ULONG ulDestLength);
VOID ThreadProc(_In_ PVOID StartContext);

// 拒绝加载驱动
NTSTATUS DenyLoadDriver(PVOID pImageBase);

// 拒绝加载DLL模块
NTSTATUS DenyLoadDll(HANDLE ProcessId, PVOID pImageBase);

typedef struct _MY_DATA
{
    HANDLE ProcessId;
    PVOID pImageBase;
}MY_DATA, *PMY_DATA;

// 设置消息回调
NTSTATUS SetNotifyRoutine()
{
    NTSTATUS status = STATUS_SUCCESS;
    status = PsSetLoadImageNotifyRoutine(LoadImageNotifyRoutine);
    return status;
}

// 关闭消息回调
NTSTATUS RemoveNotifyRoutine()
{
    NTSTATUS status = STATUS_SUCCESS;
    status = PsRemoveLoadImageNotifyRoutine(LoadImageNotifyRoutine);
    return status;
}

VOID LoadImageNotifyRoutine(PUNICODE_STRING FullImageName, HANDLE ProcessId, PIMAGE_INFO ImageInfo)
{
    DbgPrint("PID: %d --> 完整路径: %wZ --> 大小: %d --> 基地址: 0x%p \n", ProcessId, FullImageName, ImageInfo->ImageSize, ImageInfo->ImageBase);

    HANDLE hThread = NULL;
    CHAR szTemp[1024] = { 0 };
    U2C(FullImageName, szTemp, 1024);
    if (NULL != strstr(szTemp, "lyshark.sys"))
    {
        // EXE或者DLL
        if (0 != ProcessId)
        {
            // 创建多线程 延时1秒钟后再卸载模块
            PMY_DATA pMyData = ExAllocatePool(NonPagedPool, sizeof(MY_DATA));
            pMyData->ProcessId = ProcessId;
            pMyData->pImageBase = ImageInfo->ImageBase;
            PsCreateSystemThread(&hThread, 0, NULL, NtCurrentProcess(), NULL, ThreadProc, pMyData);
            DbgPrint("[LyShark] 禁止加载DLL文件 \n");
        }
        // 驱动
        else
        {
            DenyLoadDriver(ImageInfo->ImageBase);
            DbgPrint("[LyShark] 禁止加载SYS驱动文件 \n");
        }
    }
}

// 拒绝加载驱动
NTSTATUS DenyLoadDriver(PVOID pImageBase)
{
    NTSTATUS status = STATUS_SUCCESS;
    PMDL pMdl = NULL;
    PVOID pVoid = NULL;
    ULONG ulShellcodeLength = 16;
    UCHAR pShellcode[16] = { 0xB8, 0x22, 0x00, 0x00, 0xC0, 0xC3, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90 };
    PIMAGE_DOS_HEADER pDosHeader = pImageBase;
    PIMAGE_NT_HEADERS pNtHeaders = (PIMAGE_NT_HEADERS)((PUCHAR)pDosHeader + pDosHeader->e_lfanew);
    PVOID pDriverEntry = (PVOID)((PUCHAR)pDosHeader + pNtHeaders-
>OptionalHeader.AddressOfEntryPoint);

    pMdl = MmCreateMdl(NULL, pDriverEntry, ulShellcodeLength);
    MmBuildMdlForNonPagedPool(pMdl);
    pVoid = MmMapLockedPages(pMdl, KernelMode);
    RtlCopyMemory(pVoid, pShellcode, ulShellcodeLength);
    MmUnmapLockedPages(pVoid, pMdl);
    IoFreeMdl(pMdl);

    return status;
}

// 调用 MmUnmapViewOfSection 函数来卸载已经加载的 DLL 模块
NTSTATUS DenyLoadDll(HANDLE ProcessId, PVOID pImageBase)
{
    NTSTATUS status = STATUS_SUCCESS;
    PEPROCESS pEProcess = NULL;

    status = PsLookupProcessByProcessId(ProcessId, &pEProcess);
    if (!NT_SUCCESS(status))
    {
        return status;
    }

    // 卸载模块
    status = MmUnmapViewOfSection(pEProcess, pImageBase);
    if (!NT_SUCCESS(status))
    {
        return status;
    }
    return status;
}

VOID ThreadProc(_In_ PVOID StartContext)
{
    PMY_DATA pMyData = (PMY_DATA)StartContext;
    LARGE_INTEGER liTime = { 0 };

    // 延时 1 秒 负值表示相对时间
    liTime.QuadPart = -10 * 1000 * 1000;
    KeDelayExecutionThread(KernelMode, FALSE, &liTime);

    // 卸载
    DenyLoadDll(pMyData->ProcessId, pMyData->pImageBase);

    ExFreePool(pMyData);
}

NTSTATUS U2C(PUNICODE_STRING pustrSrc, PCHAR pszDest, ULONG ulDestLength)
{
    NTSTATUS status = STATUS_SUCCESS;
    ANSI_STRING strTemp;

    RtlZeroMemory(pszDest, ulDestLength);
    RtlUnicodeStringToAnsiString(&strTemp, pustrSrc, TRUE);
    if (ulDestLength > strTemp.Length)
    {
        RtlCopyMemory(pszDest, strTemp.Buffer, strTemp.Length);
    }
    RtlFreeAnsiString(&strTemp);

    return status;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{

PsRemoveLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)RemoveNotifyRoutine);
    DbgPrint("驱动卸载完成...");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.ocm \n");

    PsSetLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)SetNotifyRoutine);
    DbgPrint("驱动加载完成...");
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

加载这段驱动程序，当有DLL文件被加载后，则会强制弹出，从而实现屏蔽模块加载的作用。

当然用 `LoadImage` 回调做监控并不靠谱，因为它很容易被绕过，其实系统里存在一个开关，叫做 `PspNotifyEnableMask` 如果它的值被设置为 `0`，那么所有的相关操作都不会经过回调，所有回调都会失效。


Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe' - WinDbg:10.0.16299.15 AMD64
File Edit View Debug Window Help
Command

```
1: kd> uf PspNotifyEnableMask
Flow analysis was incomplete, some code may be missing
nt!PspNotifyEnableMask:
fffff807`819af5f0 0f0000          sldt    word ptr [rax]
fffff807`819af5f3 0000            add     byte ptr [rax],al
fffff807`819af5f5 0000            add     byte ptr [rax],al
fffff807`819af5f7 0000            add     byte ptr [rax],al
fffff807`819af5f9 0023            add     byte ptr [rbx],ah
fffff807`819af5fb 7c07            jl      nt!CmpMasterHive+0x4 (fffff807`819af604)  Branch

nt!PiLoggedErrorEventsMask+0x9:
fffff807`819af5fd f8              clc

nt!CmpMasterHive+0x4:
fffff807`819af604 86d7            xchg    dl,bh
```

> 作者： 王瑞 (LyShark)
>
> 作者邮箱： me@lyshark.com

> 版权声明：本博客文章与代码均为学习时整理的笔记，文章 [均为原创] 作品，转载文章请遵守《中华人民共和国著作权法》相关法律规定或遵守《署名CC BY-ND 4.0国际》规范，合理合规携带原创出处转载，如果不携带文章出处，并恶意转载多篇原创文章被本人发现，本人保留起诉权！