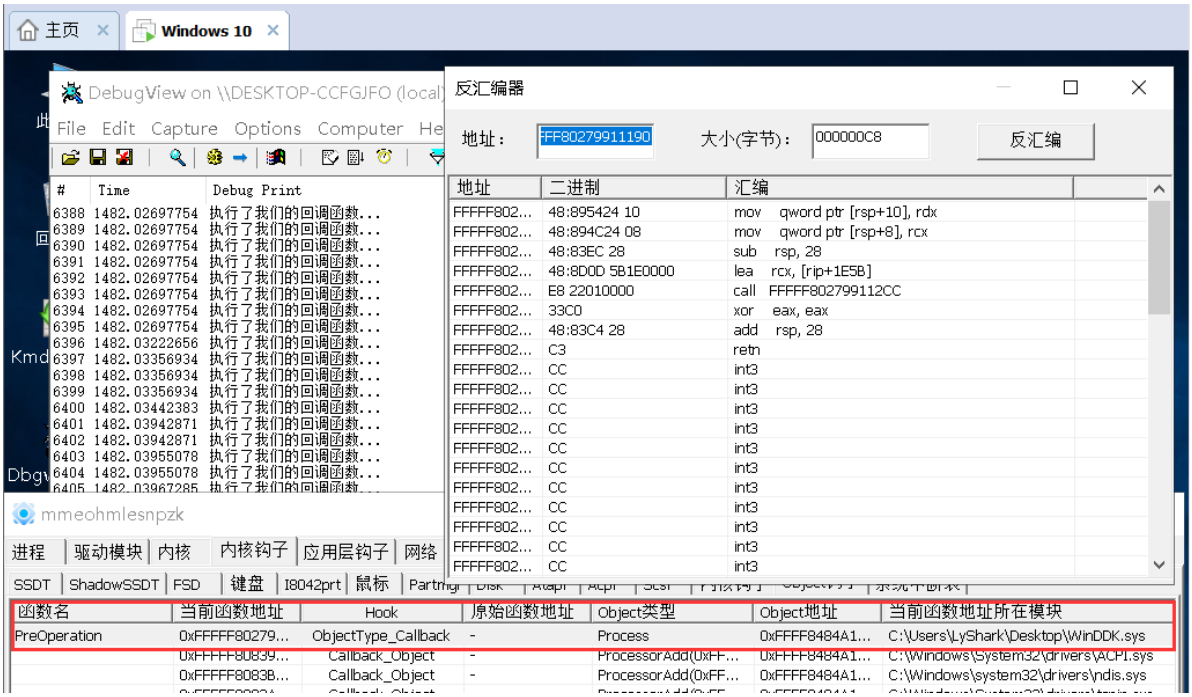
监控进程对象和线程对象操作，可以使用`ObRegisterCallbacks`这个内核回调函数，通过回调我们可以实现保护calc.exe进程不被关闭，具体操作从`OperationInformation->Object`获得进程或线程的对象，然后再回调中判断是否是计算器，如果是就直接去掉`TERMINATE_PROCESS`或`TERMINATE_THREAD`权限即可。

监控进程对象

附上进程监控回调的写法：

```
#include <ntddk.h>
#include <ntstrsafe.h>

PVOID Globle_Object_Handle;

OB_PREOP_CALLBACK_STATUS MyObjectCallBack(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION OperationInformation)
{
    DbgPrint("执行了我们的回调函数...");
    return STATUS_SUCCESS;
}
VOID UnDriver(PDRIVER_OBJECT driver)
{
    ObUnRegisterCallbacks(Globle_Object_Handle);
    DbgPrint("回调卸载完成...");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    OB_OPERATION_REGISTRATION Base;                          // 回调函数结构体(你所
填的结构都在这里)
    OB_CALLBACK_REGISTRATION CallbackReg;

    CallbackReg.RegistrationContext = NULL;                  // 注册上下文(你回调函
数返回参数)
    CallbackReg.Version = OB_FLT_REGISTRATION_VERSION;       // 注册回调版本
    CallbackReg.OperationRegistration = &Base;
    CallbackReg.OperationRegistrationCount = 1;               // 操作计数(下钩数量)
    RtlUnicodeStringInit(&CallbackReg.Altitude, L"600000");   // 长度
    Base.ObjectType = PsProcessType;                         // 进程操作类型.此处为
进程操作
    Base.Operations = OB_OPERATION_HANDLE_CREATE;             // 操作句柄创建
    Base.PreOperation = MyObjectCallBack;                    // 你自己的回调函数
    Base.PostOperation = NULL;

    if (ObRegisterCallbacks(&CallbackReg, &Globle_Object_Handle)) // 注册回调
        DbgPrint("回调注册成功...");
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

上方代码运行后，我们可以打开Xuetr扫描一下内核Object钩子，可以看到已经成功挂钩了。

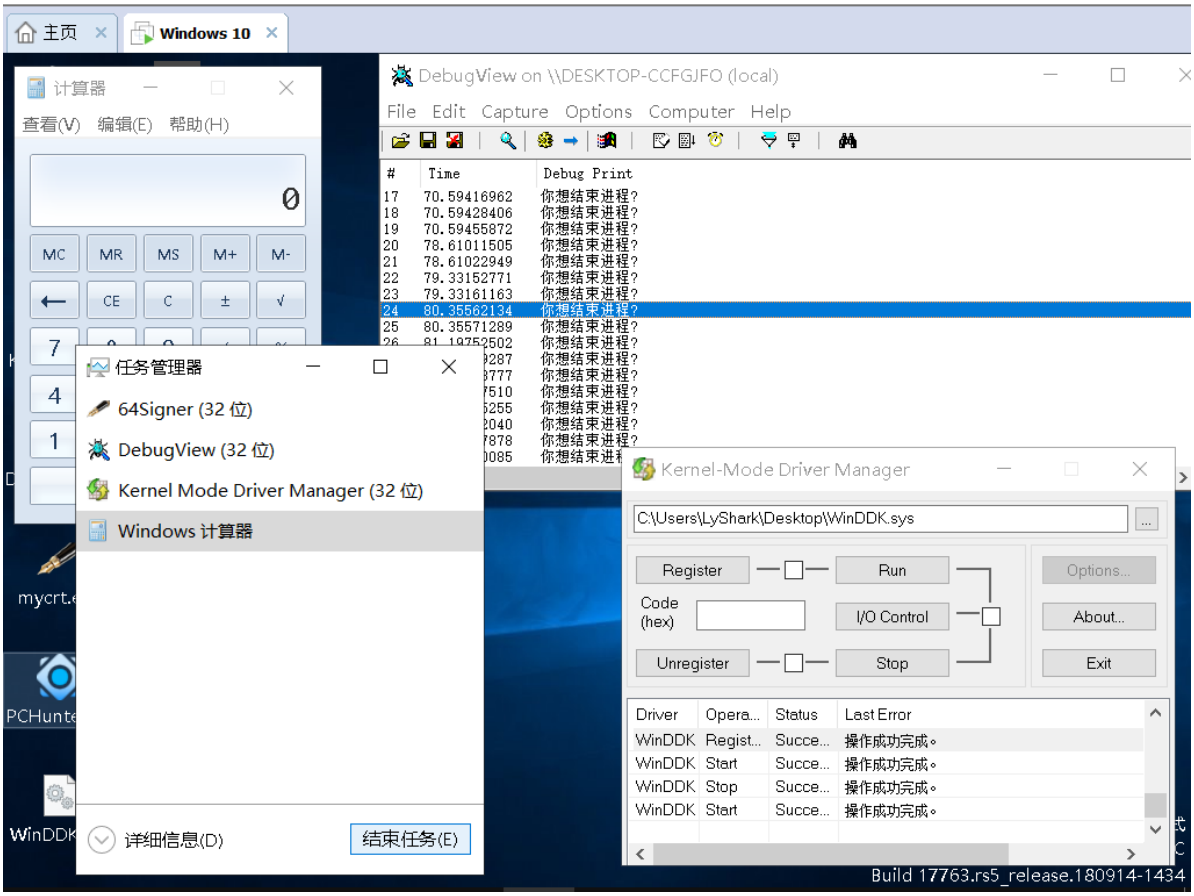


检测计算器进程的关闭状态，代码如下：

```
#include <ntddk.h>
#include <wdm.h>
#include <ntstrsafe.h>
#define PROCESS_TERMINATE 1

PVOID Globle_Object_Handle;
NTKERNELAPI UCHAR * PsGetProcessImageFileName(__in PEPROCESS Process);

char* GetProcessImageNameByProcessID(ULONG ulProcessID)
{
    NTSTATUS  Status;
    PEPROCESS  EProcess = NULL;
    Status = PsLookupProcessByProcessId((HANDLE)ulProcessID, &EProcess);
    if (!NT_SUCCESS(Status))
        return FALSE;
    ObDereferenceObject(EProcess);
    return (char*)PsGetProcessImageFileName(EProcess);
}
OB_PREOP_CALLBACK_STATUS MyObjectCallBack(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION Operation)
{
    char ProcName[256] = { 0 };
    HANDLE pid = PsGetProcessId((PEPROCESS)Operation->Object);            // 取出当
前调用函数的PID
    strcpy(ProcName, GetProcessImageNameByProcessID((ULONG)pid));        // 通过
PID取出进程名,然后直接拷贝内存
    //DbgPrint("当前进程的名字是：%s", ProcName);

    if (strstr(ProcName, "win32calc.exe"))
    {
        if (Operation->Operation == OB_OPERATION_HANDLE_CREATE)
        {
```

```
            if ((Operation->Parameters-
>CreateHandleInformation.OriginalDesiredAccess & PROCESS_TERMINATE) ==
PROCESS_TERMINATE)
            {
                DbgPrint("你想结束进程?");
                // 如果是计算器，则去掉它的结束权限，在Win10上无效
                Operation->Parameters->CreateHandleInformation.DesiredAccess =
~THREAD_TERMINATE;
                return STATUS_UNSUCCESSFUL;
            }
        }
    }
    return STATUS_SUCCESS;
}
VOID UnDriver(PDRIVER_OBJECT driver)
{
    ObUnRegisterCallbacks(Globle_Object_Handle);
    DbgPrint("回调卸载完成...");
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    NTSTATUS obst = 0;
    OB_CALLBACK_REGISTRATION obReg;
    OB_OPERATION_REGISTRATION opReg;

    memset(&obReg, 0, sizeof(obReg));
    obReg.Version = ObGetFilterVersion();
    obReg.OperationRegistrationCount = 1;
    obReg.RegistrationContext = NULL;
    RtlInitUnicodeString(&obReg.Altitude, L"321125");
    obReg.OperationRegistration = &opReg;
    memset(&opReg, 0, sizeof(opReg));
    opReg.ObjectType = PsProcessType;
    opReg.Operations = OB_OPERATION_HANDLE_CREATE |
OB_OPERATION_HANDLE_DUPLICATE;
    opReg.PreOperation = (POB_PRE_OPERATION_CALLBACK)&MyObjectCallBack;
    obst = ObRegisterCallbacks(&obReg, &Globle_Object_Handle);
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

首先运行计算器，然后启动驱动保护，此时我们在任务管理器中就无法结束计算器进程了。

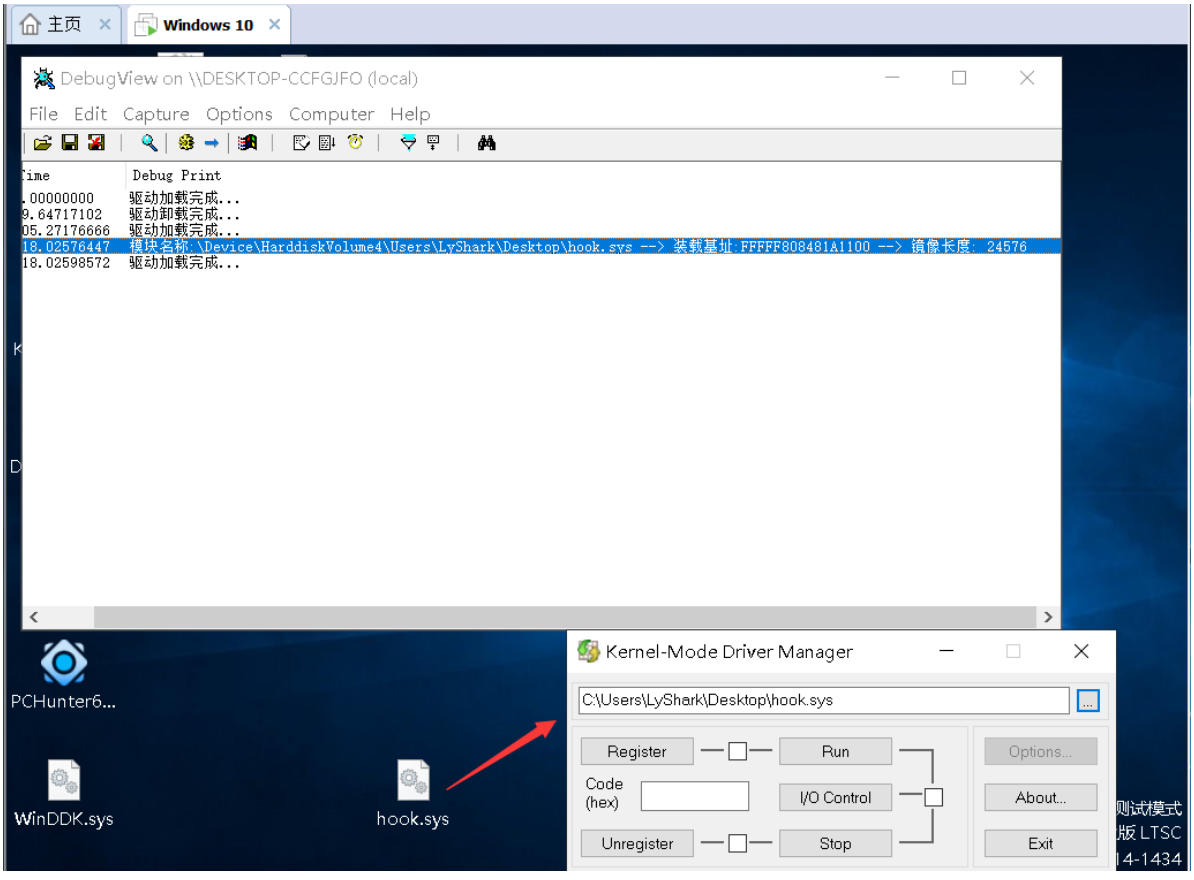


监控进程中模块加载

系统中的模块加载包括用户层模块DLL和内核模块SYS的加载，在 Windows X64 环境下我们可以调用 `PsSetLoadImageNotifyRoutine` 内核函数来设置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从而执行我们自己的回调例程。

```
#include <ntddk.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 MyLoadImageNotifyRoutine(PUNICODE_STRING FullImageName,HANDLE
ProcessId,PIMAGE_INFO ImageInfo)
{
    PVOID pDrvEntry;
    if (FullImageName != NULL && MmIsAddressValid(FullImageName)) // MmIsAddress
验证地址可用性
    {
```

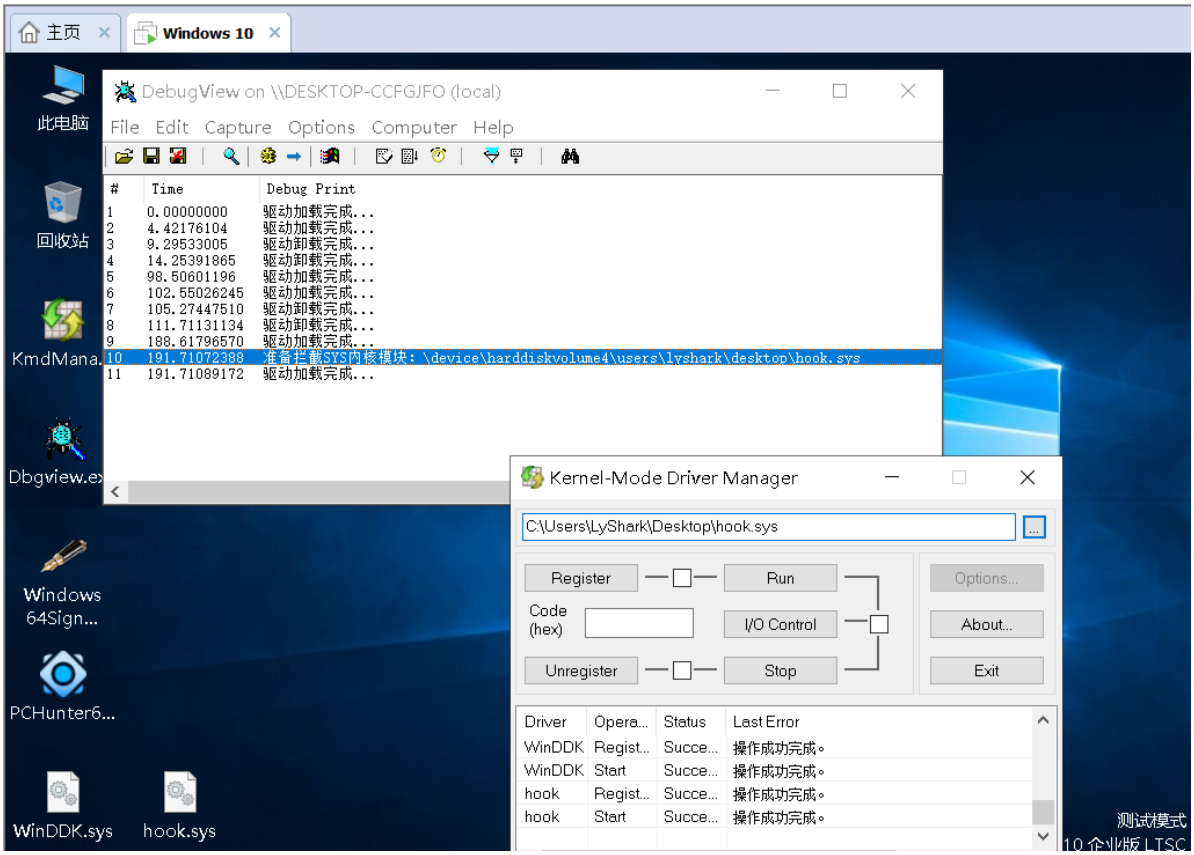
```
        if (ProcessId == 0)
        {
            pDrvEntry = GetDriverEntryByImageBase(ImageInfo->ImageBase);
            DbgPrint("模块名称:%wZ --> 装载基址:%p --> 镜像长度: %d", FullImageName,
pDrvEntry,ImageInfo->ImageSize);
        }
    }
}

VOID UnDriver(PDRIVER_OBJECT driver)
{

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

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{

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



接着我们给上方的代码加上判断功能，只需在上方代码的基础上小改一下即可，需要注意回调函数中的第二个参数，如果返回值为零则表示加载SYS，如果返回非零则表示加载DLL

```
VOID UnicodeToChar(PUNICODE_STRING dst, char *src)
```

```
{
    ANSI_STRING string;
    RtlUnicodeStringToAnsiString(&string, dst, TRUE);
    strcpy(src, string.Buffer);
    RtlFreeAnsiString(&string);
}

VOID MyLoadImageNotifyRoutine(PUNICODE_STRING FullImageName,HANDLE
ModuleStyle,PIMAGE_INFO ImageInfo)
{
    PVOID pDrvEntry;
    char szFullImageName[256] = { 0 };
    if (FullImageName != NULL && MmIsAddressValid(FullImageName)) // MmIsAddress
验证地址可用性
    {
        if (ModuleStyle == 0)  // ModuleStyle为零表示加载sys非零表示加载DLL
        {
            pDrvEntry = GetDriverEntryByImageBase(ImageInfo->ImageBase);
            UnicodeToChar(FullImageName, szFullImageName);
            if (strstr(_strlwr(szFullImageName), "hook.sys"))
            {
                DbgPrint("准备拦截SYS内核模块：%s", _strlwr(szFullImageName));
            }
        }
    }
}
```



上方代码就可以判断加载的模块并作出处理动作了，但是我们仍然无法判断到底是那个进程加载的 `hook.sys` 驱动，因为回调函数很底层，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。

判断了是驱动后，接着我们就要实现屏蔽驱动，通过`ImageInfo->ImageBase`来获取被加载驱动程序`hook.sys`的映像基址，然后找到NT头的OptionalHeader节点，该节点里面就是被加载驱动入口的地址，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。

```
#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++ 优化中启用内部函数
KIRQL  WPOFFx64()          // 关闭写保护
{
    KIRQL  irql = KeRaiseIrqlToDpcLevel();
    UINT64  cr0 = __readcr0();
    cr0 &= 0xfffffffffffeffff;
    _disable();
    __writecr0(cr0);
    return  irql;
}
void  WPONx64(KIRQL  irql) // 开启写保护
{
    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;
```

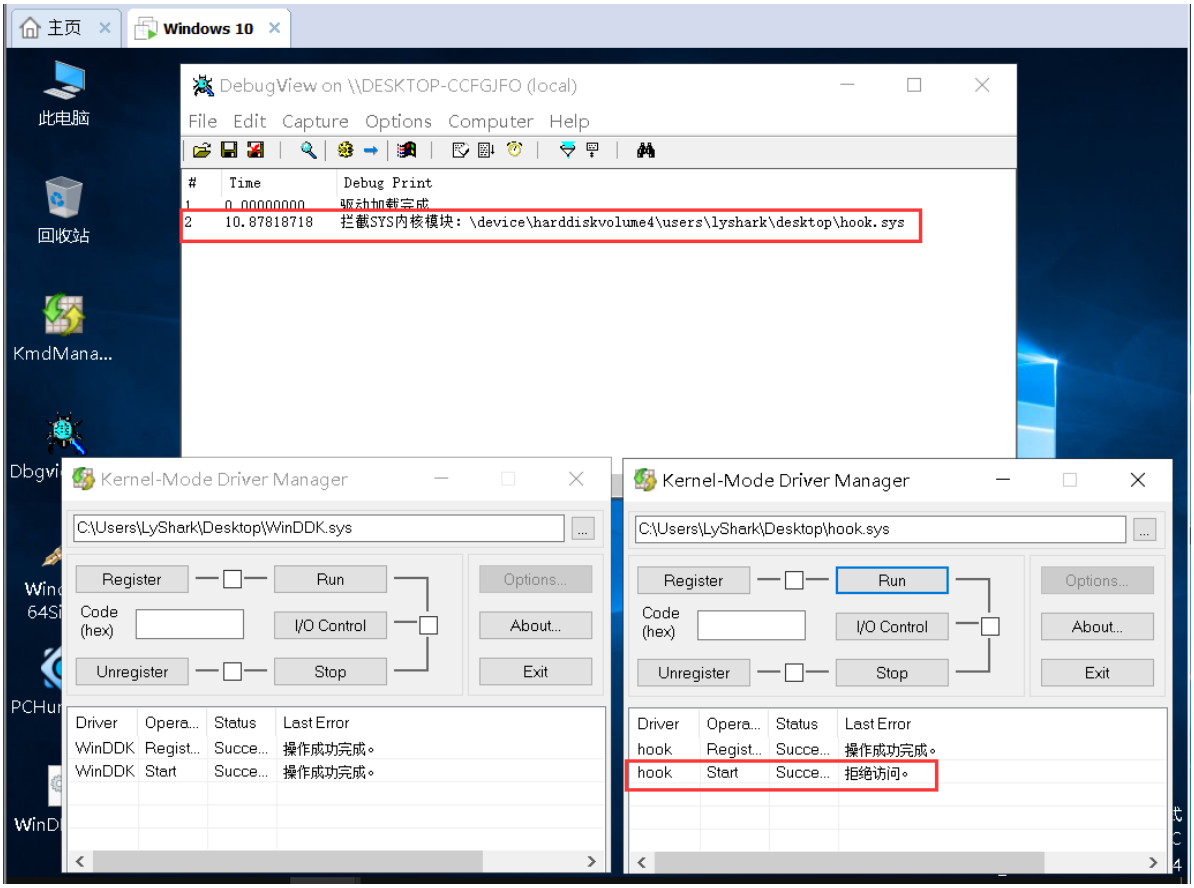
```
    kirql = WPOFFx64();
    memcpy(DriverEntry, fuck,sizeof(fuck) / sizeof(fuck[0]));
    WPONx64(kirql);
    return TRUE;
}

VOID MyLoadImageNotifyRoutine(PUNICODE_STRING FullImageName, HANDLE ModuleStyle,
PIMAGE_INFO ImageInfo)
{
    PVOID pDrvEntry;
    char szFullImageName[256] = { 0 };
    if (FullImageName != NULL && MmIsAddressValid(FullImageName)) // MmIsAddress
验证地址可用性
    {
        if (ModuleStyle == 0)  // ModuleStyle为零表示加载sys非零表示加载DLL
        {
            pDrvEntry = GetDriverEntryByImageBase(ImageInfo->ImageBase);
            UnicodeToChar(FullImageName, szFullImageName);
            if (strstr(_strlwr(szFullImageName), "hook.sys"))
            {
                DbgPrint("拦截SYS内核模块: %s", szFullImageName);
                DenyLoadDriver(pDrvEntry);
            }
        }
    }
}
VOID UnDriver(PDRIVER_OBJECT driver)
{

PsRemoveLoadImageNotifyRoutine((PLOAD_IMAGE_NOTIFY_ROUTINE)MyLoadImageNotifyRout
ine);
    DbgPrint("驱动卸载完成...");
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{

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

屏蔽DLL加载，只需要在上面的代码上稍微修改一下就好，这里提供到另一种写法。

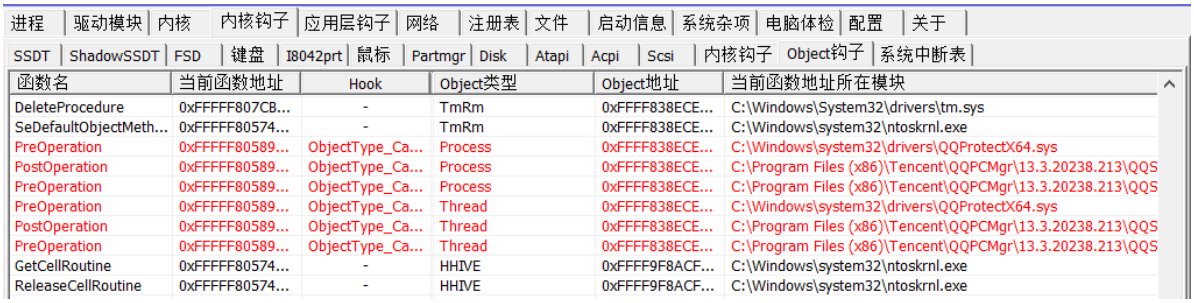
```
char *UnicodeToLongString(PUNICODE_STRING uString)
{
    ANSI_STRING asStr;
    char *Buffer = NULL;;
    RtlUnicodeStringToAnsiString(&asStr, uString, TRUE);
    Buffer = ExAllocatePoolWithTag(NonPagedPool, uString->MaximumLength *
sizeof(wchar_t), 0);
    if (Buffer == NULL)
        return NULL;
    RtlCopyMemory(Buffer, asStr.Buffer, asStr.Length);
```

```
    return Buffer;
}
VOID MyLoadImageNotifyRoutine(PUNICODE_STRING FullImageName, HANDLE ModuleStyle,
PIMAGE_INFO ImageInfo)
{
    PVOID pDrvEntry;
    char *PareString = NULL;

    if (MmIsAddressValid(FullImageName))
    {
        if (ModuleStyle != 0)  // 非零则监控DLL加载
        {
            PareString = UnicodeToLongString(FullImageName);
            if (PareString != NULL)
            {
                if (strstr(PareString, "hook.dll"))
                {
                    pDrvEntry = GetDriverEntryByImageBase(ImageInfo->ImageBase);
                    if (pDrvEntry != NULL)
                        DenyLoadDriver(pDrvEntry);
                }
            }
        }
    }
}
```

我们以屏蔽SYS内核模块为例，当驱动文件`WinDDK.sys`被加载后，尝试加载`hook.sys`会提示拒绝访问，说明我们的驱动保护生效了。



关键的内核进程骚操作已经分享完了，杀软的主动防御系统，游戏的保护系统等都会用到这些东西。



> 本书作者: 王瑞 (LyShark)
> 作者邮箱: me@lyshark.com
> 作者博客: https://lyshark.cnblogs.com
> 团队首页: www.lyshark.com