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;

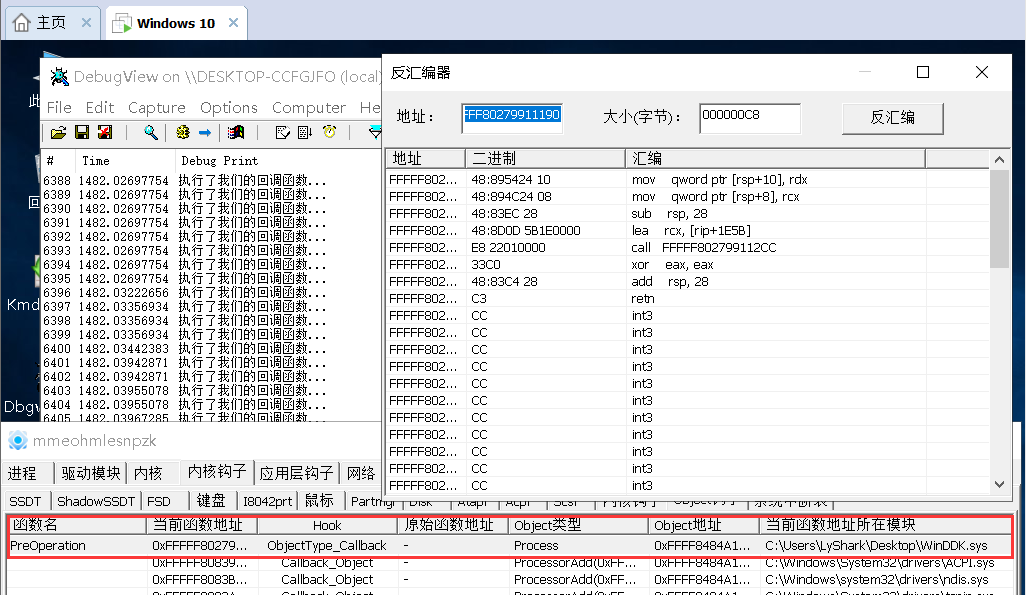
if (ObRegisterCallbacks(&CallbackReg, &Globle\_Object\_Handle)) // 注册回调

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);

前调用函数的PID

strcpy(ProcName, GetProcessImageNameByProcessID((ULONG)pid)); 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);

// 取出当

// 通过

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;

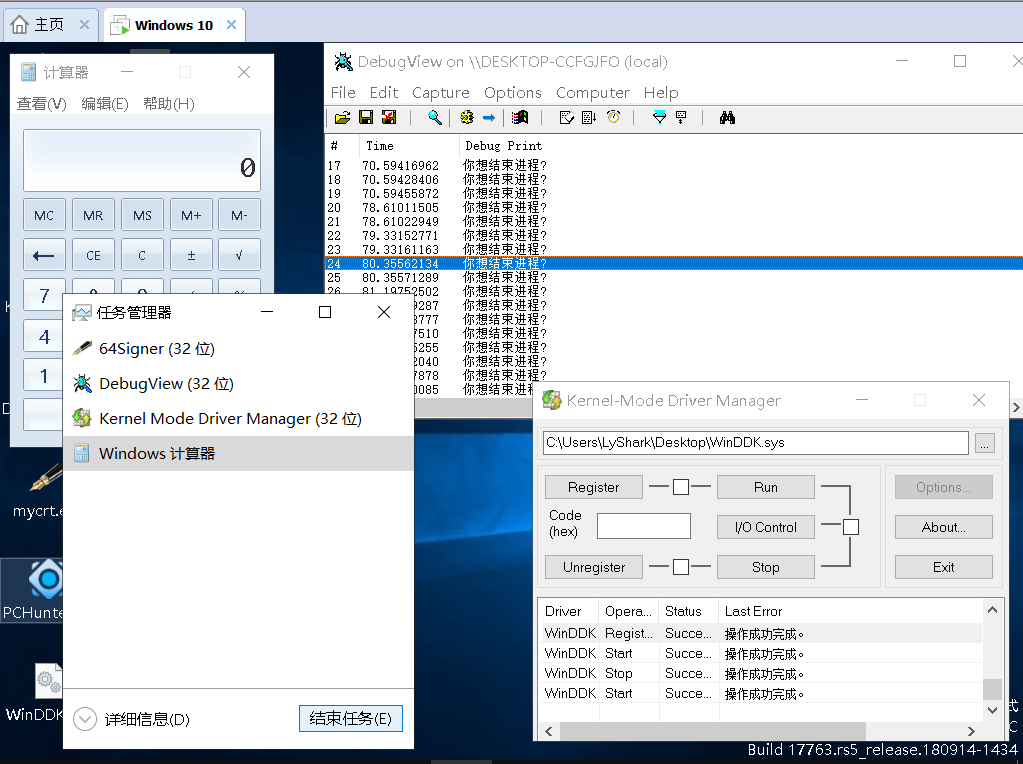
opReg.PreOperation = (POB\_PRE\_OPERATION\_CALLBACK)&MyObjectCallBack; obst = ObRegisterCallbacks(&obReg, &Globle\_Object\_Handle);

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)

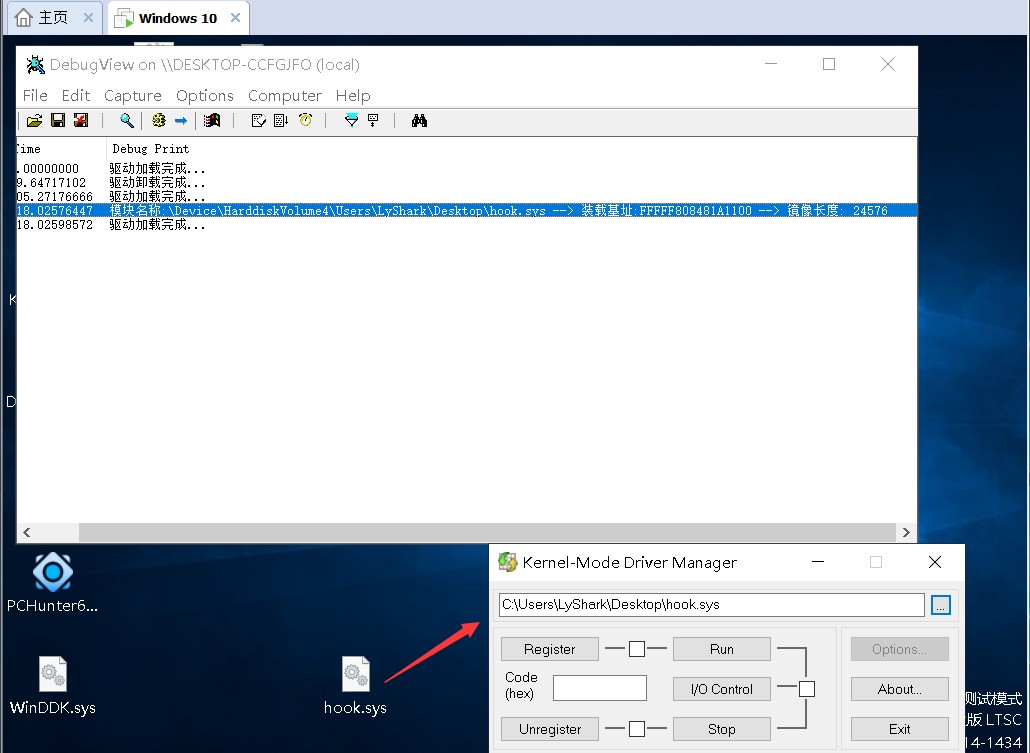
{

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"))

{

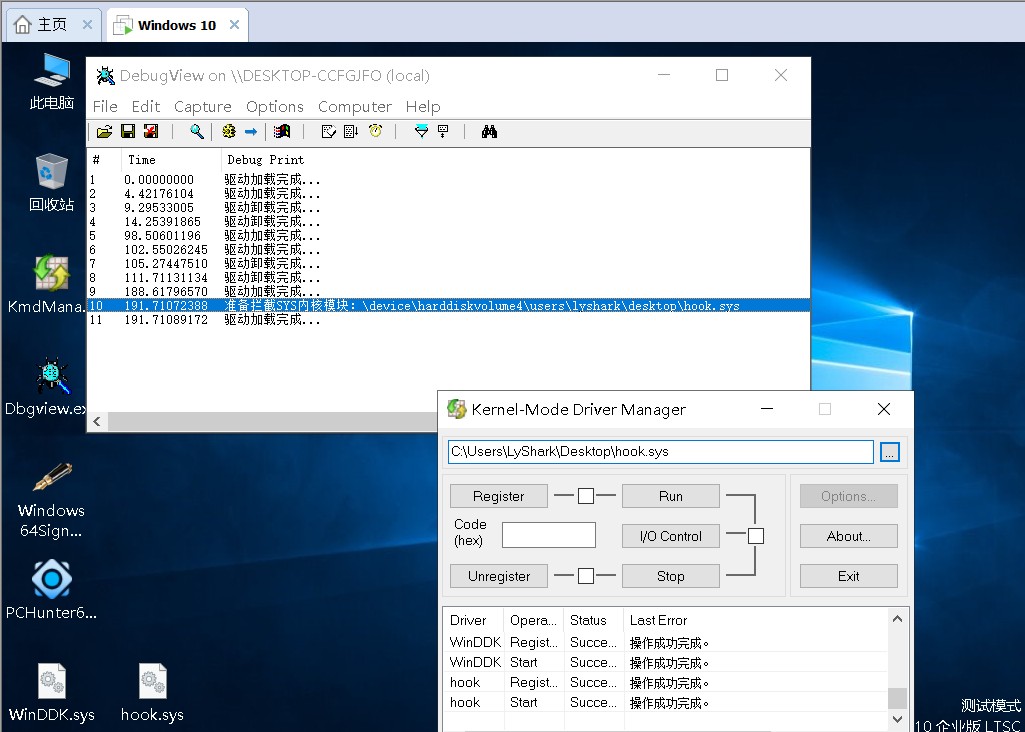
DbgPrint("准备拦截SYS内核模块：%s", \_strlwr(szFullImageName));

}

}

}

}



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);

}

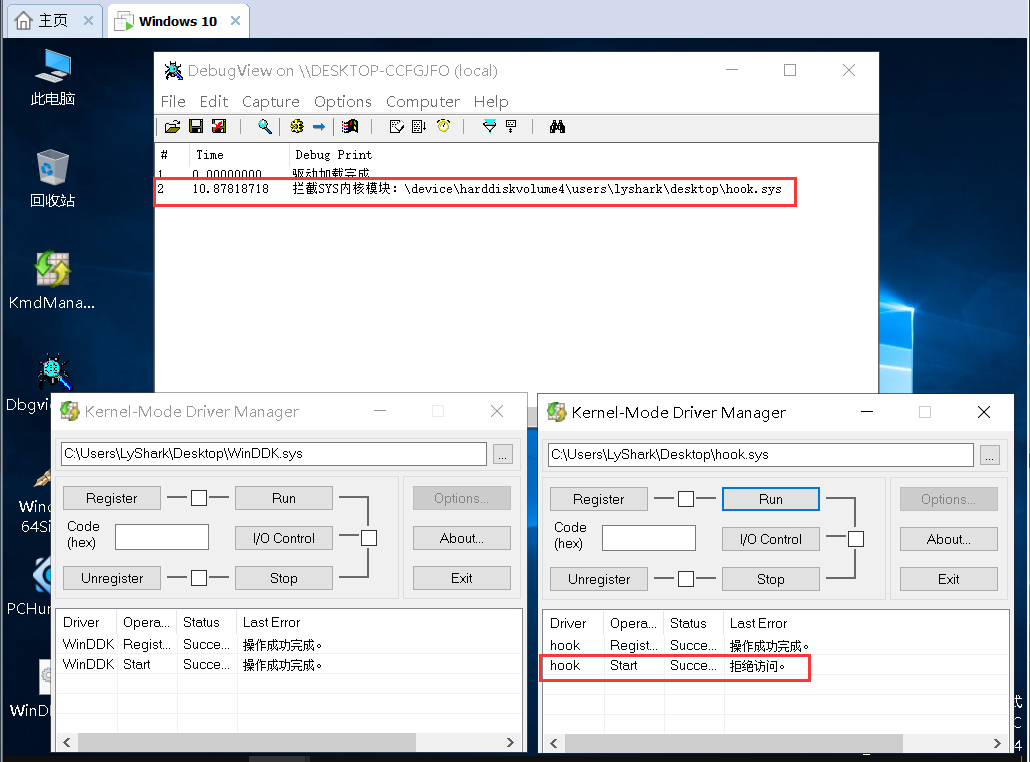
}

}

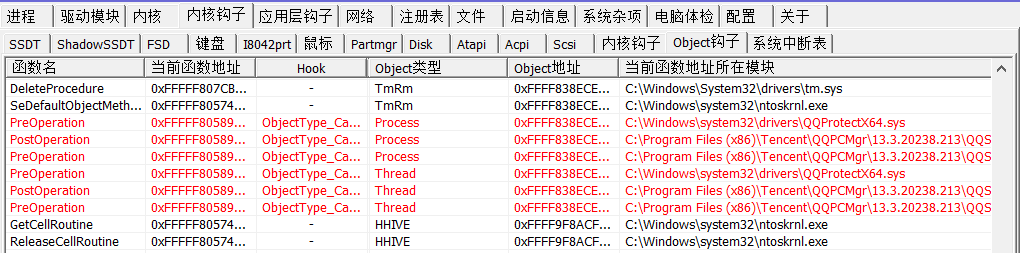
}

}

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： m [e@lyshark.com](mailto:e@lyshark.com)

作者博客： h ttps://lyshark.cnblogs.com

团队首页： w ww.lyshark.com