# 在之前的文章中 LyShark 一直都在教大家如何让驱动程序与应用层进行 正向通信 ，而在某些时候我们不仅仅只需要正向通信，也需要反向通信，例如杀毒软件如果驱动程序拦截到恶意操作则必须将这个请求 动态的转发到应用层以此来通知用户，而这种通信方式的实现有多种，通常可以使用创建Socket套接字 的方式实现，亦或者使用本章所介绍的通过 事件同步 的方法实现反向通信。

基于事件同步方式实现的通信需要用的最重要函数 IoCreateNotificationEvent() 这是微软定为开发者提供的。

# IoCreateNotificationEvent 例程创建或打开一个命名通知事件，用于通知一个或多个执行线程已发生事件。

PKEVENT IoCreateNotificationEvent( [in] PUNICODE\_STRING EventName, [out] PHANDLE EventHandle

);

其中的第二个参数 EventHandle 指向返回事件对象的内核句柄的位置的指针。此处为了能更好的接收和管理指针与进程之间的关系我们最好定义一个 DEVICE\_EXTEN 结构体。



// 自定义设备扩展

typedef struct

{

HANDLE hProcess; // HANDLE PKEVENT pkProcessEvent; // 事件对象HANDLE hProcessId; // PID

HANDLE hpParProcessId; // PPID BOOLEAN bIsCreateMark; // Flag

}DEVICE\_EXTEN, PDEVICE\_EXTEN;

驱动入口处 PsSetCreateProcessNotifyRoutine 则用于创建一个进程回调，该回调函数被指定为

pMyCreateProcessThreadRoutine 当回调函数被创建后，一旦有新进程创建则会执行函数内部的具体流程。

如代码所示，在 pMyCreateProcessThreadRoutine 函数内首先通过 (PDEVICE\_EXTEN)GlobalDevObj-

>DeviceExtension 得到了自定义设备扩展指针，接着将当前进程的 PID,PPID,isCreate 等属性赋予到扩展指针中，当一切准备就绪后，通过调用 KeSetEvent 将当前进程事件设置为有信号状态，设置后重置为默认值。

// 自定义回调函数

VOID pMyCreateProcessThreadRoutine(IN HANDLE pParentId, HANDLE hProcessId, BOOLEAN bisCreate)

{

PDEVICE\_EXTEN pDeviceExten = (PDEVICE\_EXTEN)GlobalDevObj->DeviceExtension;

// 将进行信息依次复制到结构体中

pDeviceExten->hProcessId = hProcessId; pDeviceExten->hpParProcessId = pParentId; pDeviceExten->bIsCreateMark = bisCreate;

// 设置为有信号

KeSetEvent(pDeviceExten->pkProcessEvent, 0, FALSE);

// 重置状态信号

KeResetEvent(pDeviceExten->pkProcessEvent);

}

此时由于客户端中通过 OpenEventW(SYNCHRONIZE, FALSE, EVENT\_NAME) 打开了内核对象，并通过WaitForSingleObject(hProcessEvent, INFINITE) 一直在等待事件，一旦内核驱动KeSetEvent(pDeviceExten->pkProcessEvent, 0, FALSE) 设置为有信号状态，则应用层会通过

DeviceIoControl 向内核层发送 IOCTL 控制信号并等待接收数据。

此时主派遣函数 DisPatchIoControl 被触发执行，通过 (PPROCESS\_PTR)pUserOutPutBuffer 获取到应用层缓冲区设备指针，并依次通过 pBuffer-> 的方式设置参数，最后返回状态码，此时应用层

PROCESS\_PTR 中也就得到了进程创建的相关信息。

pBuffer = (PPROCESS\_PTR)pUserOutPutBuffer;

uIoControl = pIrpStack->Parameters.DeviceIoControl.IoControlCode; uReadLen = pIrpStack->Parameters.DeviceIoControl.InputBufferLength; uWriteLen = pIrpStack->Parameters.DeviceIoControl.OutputBufferLength;

switch (uIoControl)

{

case IOCTL:

pDeviceExten = (PDEVICE\_EXTEN)GlobalDevObj->DeviceExtension; pBuffer->hProcessId = pDeviceExten->hProcessId;

pBuffer->hpParProcessId = pDeviceExten->hpParProcessId; pBuffer->bIsCreateMark = pDeviceExten->bIsCreateMark; break;

default:

ntStatus = STATUS\_INVALID\_PARAMETER;

uWriteLen = 0; break;

}

# 如上就是内核层与应用层的部分代码功能分析，接下来我将完整代码分享出来，大家可以自行测试效 果。

驱动程序WinDDK.sys完整代码；

// 署名权

// right to sign one's name on a piece of work

// PowerBy: LyShark

// Email: [me@lyshark.com](mailto:me@lyshark.com)

#include <ntifs.h> #include <ntstrsafe.h>

#define IOCTL CTL\_CODE(FILE\_DEVICE\_UNKNOWN,0x801,METHOD\_BUFFERED,FILE\_ANY\_ACCESS)

UNICODE\_STRING GlobalSym = { 0 };

PDEVICE\_OBJECT GlobalDevObj;

// 自定义设备扩展

typedef struct

{

HANDLE hProcess; // HANDLE PKEVENT pkProcessEvent; // 事件对象

HANDLE hProcessId; // PID HANDLE hpParProcessId; // PPID BOOLEAN bIsCreateMark; // Flag



}DEVICE\_EXTEN, PDEVICE\_EXTEN;

// 自定义结构体(临时) typedef struct

{

HANDLE hProcessId; HANDLE hpParProcessId; BOOLEAN bIsCreateMark;

}PROCESS\_PTR, PPROCESS\_PTR;

// 自定义回调函数

VOID pMyCreateProcessThreadRoutine(IN HANDLE pParentId, HANDLE hProcessId, BOOLEAN bisCreate)

{

PDEVICE\_EXTEN pDeviceExten = (PDEVICE\_EXTEN)GlobalDevObj->DeviceExtension;

// 将进行信息依次复制到结构体中

pDeviceExten->hProcessId = hProcessId; pDeviceExten->hpParProcessId = pParentId; pDeviceExten->bIsCreateMark = bisCreate;

// 设置为有信号

KeSetEvent(pDeviceExten->pkProcessEvent, 0, FALSE);

// 重置状态信号

KeResetEvent(pDeviceExten->pkProcessEvent);

}

// 默认派遣函数

NTSTATUS DisPatchComd(PDEVICE\_OBJECT pDeviceObject, PIRP pIrp)

{

pIrp->IoStatus.Information = 0;

pIrp->IoStatus.Status = STATUS\_SUCCESS; IoCompleteRequest(pIrp, IO\_NO\_INCREMENT);

return pIrp->IoStatus.Status;

}

// 主派遣函数

NTSTATUS DisPatchIoControl(PDEVICE\_OBJECT pDeviceObject, PIRP pIrp)

{

NTSTATUS ntStatus; PIO\_STACK\_LOCATION pIrpStack; PVOID pUserOutPutBuffer; PPROCESS\_PTR pBuffer;

ULONG uIoControl = 0; ULONG uReadLen;

ULONG uWriteLen; PDEVICE\_EXTEN pDeviceExten;

pIrpStack = IoGetCurrentIrpStackLocation(pIrp); pUserOutPutBuffer = pIrp->AssociatedIrp.SystemBuffer;

pBuffer = (PPROCESS\_PTR)pUserOutPutBuffer;

uIoControl = pIrpStack->Parameters.DeviceIoControl.IoControlCode; uReadLen = pIrpStack->Parameters.DeviceIoControl.InputBufferLength; uWriteLen = pIrpStack->Parameters.DeviceIoControl.OutputBufferLength;

switch (uIoControl)

{

case IOCTL:

pDeviceExten = (PDEVICE\_EXTEN)GlobalDevObj->DeviceExtension; pBuffer->hProcessId = pDeviceExten->hProcessId;

pBuffer->hpParProcessId = pDeviceExten->hpParProcessId; pBuffer->bIsCreateMark = pDeviceExten->bIsCreateMark; break;

default:

ntStatus = STATUS\_INVALID\_PARAMETER;

uWriteLen = 0; break;

}

pIrp->IoStatus.Information = uWriteLen; pIrp->IoStatus.Status = ntStatus; IoCompleteRequest(pIrp, IO\_NO\_INCREMENT); return ntStatus;

}

// 卸载驱动

VOID DriverUnLoad(PDRIVER\_OBJECT pDriverObject)

{

NTSTATUS ntStatus;

UNICODE\_STRING SymboLicLinkStr = { 0 };

// 删除符号链接

ntStatus = RtlUnicodeStringInit(&SymboLicLinkStr, L"\\DosDevices\\Symboliclnk");

ntStatus = IoDeleteSymbolicLink(&SymboLicLinkStr);

// 注销进程消息回调PsSetCreateProcessNotifyRoutine(pMyCreateProcessThreadRoutine, TRUE); IoDeleteDevice(pDriverObject->DeviceObject);

}

// 入口函数

NTSTATUS DriverEntry(PDRIVER\_OBJECT pDriver, PUNICODE\_STRING RegistryPath)

{

NTSTATUS ntStatus = NULL; PDEVICE\_OBJECT pDeviceObject = NULL; UNICODE\_STRING uDeviceName = { 0 }; UNICODE\_STRING uEventName = { 0 };

// 初始化字符串

ntStatus = RtlUnicodeStringInit(&uDeviceName, L"\\Device\\ProcLook"); ntStatus = RtlUnicodeStringInit(&GlobalSym, L"\\DosDevices\\Symboliclnk"); ntStatus = RtlUnicodeStringInit(&uEventName,

L"\\BaseNamedObjects\\ProcLook"); DbgPrint("hello lyshark.com"); if (!NT\_SUCCESS(ntStatus))

{

return ntStatus;

}

// 创建设备

ntStatus = IoCreateDevice(pDriver,sizeof(DEVICE\_EXTEN),&uDeviceName,FILE\_DEVICE\_UNKNOWN,FILE

\_DEVICE\_SECURE\_OPEN,FALSE,&pDeviceObject);

DbgPrint("[LyShark] 创建设备对象");

if (!NT\_SUCCESS(ntStatus))

{

IoDeleteDevice(pDeviceObject); return ntStatus;

}

pDriver->Flags |= DO\_BUFFERED\_IO; GlobalDevObj = pDeviceObject;

// 获取设备扩展指针,并IoCreateNotificationEvent创建一个R3到R0的事件 PDEVICE\_EXTEN pDeviceExten = (PDEVICE\_EXTEN)pDeviceObject->DeviceExtension; pDeviceExten->pkProcessEvent = IoCreateNotificationEvent(&uEventName,

&pDeviceExten->hProcess); KeClearEvent(pDeviceExten->pkProcessEvent);

DbgPrint("[LyShark] 创建事件回调");

// 创建符号链接

ntStatus = IoCreateSymbolicLink(&GlobalSym, &uDeviceName); if (!NT\_SUCCESS(ntStatus))

{

IoDeleteDevice(pDeviceObject); return ntStatus;

}

// 注册进程创建回调

ntStatus = PsSetCreateProcessNotifyRoutine(pMyCreateProcessThreadRoutine, FALSE);

DbgPrint("[LyShark] 创建进程回调");

if (!NT\_SUCCESS(ntStatus))

{

IoDeleteDevice(pDeviceObject); return ntStatus;

}

// 初始化派遣函数

for (ULONG i = 0; i < IRP\_MJ\_MAXIMUM\_FUNCTION; i++)

{

pDriver->MajorFunction[i] = DisPatchComd;

}

pDriver->MajorFunction[IRP\_MJ\_DEVICE\_CONTROL] = DisPatchIoControl;

// 关闭驱动

pDriver->DriverUnload = DriverUnLoad; return ntStatus;

}

应用层客户端程序 lyshark.exe 完整代码；



// 署名权

// right to sign one's name on a piece of work

// PowerBy: LyShark

// Email: [me@lyshark.com](mailto:me@lyshark.com)

#include <iostream> #include <Windows.h>

#define EVENT\_NAME L"Global\\ProcLook"

#define IOCTL CTL\_CODE(FILE\_DEVICE\_UNKNOWN,0x801,METHOD\_BUFFERED,FILE\_ANY\_ACCESS)

typedef struct

{

HANDLE hProcessId; HANDLE hpParProcessId; BOOLEAN bIsCreateMark;

}PROCESS\_PTR, PPROCESS\_PTR;

int main(int argc, char argv[])

{

PROCESS\_PTR Master = { 0 }; PROCESS\_PTR Slave = { 0 };

DWORD dwRet = 0; BOOL bRet = FALSE;

// 打开驱动设备对象

HANDLE hDriver = CreateFile(L"\\\\.\\Symboliclnk",GENERIC\_READ | GENERIC\_WRITE,NULL,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);

if (hDriver == INVALID\_HANDLE\_VALUE)

{

return 0;

}

// 打开内核事件对象

HANDLE hProcessEvent = OpenEventW(SYNCHRONIZE, FALSE, EVENT\_NAME);

if (NULL == hProcessEvent)

{

return 0;

}

while (TRUE)

{

// 等待事件

WaitForSingleObject(hProcessEvent, INFINITE);

// 发送控制信号

bRet = DeviceIoControl(hDriver,IOCTL,NULL,0,&Master,sizeof(Master),&dwRet,NULL);

if (!bRet)

{

return 0;

}

if (bRet)

{

if (Master.hpParProcessId != Slave.hpParProcessId || Master.hProcessId != Slave.hProcessId || Master.bIsCreateMark != Slave.bIsCreateMark)

{

if (Master.bIsCreateMark)

{

printf("[LyShark] 父进程: %d | 进程PID: %d | 状态: 创建进程 \n", Master.hpParProcessId, Master.hProcessId);

}

else

{

printf("[LyShark] 父进程: %d | 进程PID: %d | 状态: 退出进程 \n", Master.hpParProcessId, Master.hProcessId);

}

Slave = Master;

}

}

else

{

break;

}

}

CloseHandle(hDriver); return 0;

}

# 手动编译这两个程序，将驱动签名后以管理员身份运行 lyshark.exe 客户端，此时屏幕中即可看到滚动输出效果，如此一来就实现了循环传递参数的目的。

