本篇文章与上一篇文章《驱动开发：内核注册并监控对象回调》所使用的方式是一样的都是使用ObRegisterCallbacks注册回调事件，只不过上一篇博文中LyShark将回调结构体OB_OPERATION_REGISTRATION中的ObjectType填充为了PsProcessType和PsThreadType格式从而实现监控进程与线程，本章我们需要将该结构填充为IoFileObjectType以此来实现对文件的监控，文件过滤驱动不仅仅可以用来监控文件的打开，还可以用它实现对文件的保护，一旦驱动加载则文件是不可被删除和改动的。

与进程线程回调有少许的不同，文件回调需要开启驱动的TypeInfo.SupportsObjectCallbacks开关，并定义一些微软结构，如下是我们所需要的公开结构体，可在微软官方或WinDBG中获取到最新的，将其保存为lyshark.h方便后期引用。

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

typedef struct _CALLBACK_ENTRY
{
    LIST_ENTRY CallbackList;
    OB_OPERATION  Operations;
    ULONG Active;
    PVOID Handle;
    POBJECT_TYPE ObjectType;
    POB_PRE_OPERATION_CALLBACK  PreOperation;
    POB_POST_OPERATION_CALLBACK PostOperation;
    ULONG unknown;
} CALLBACK_ENTRY, *PCALLBACK_ENTRY;

typedef struct _LDR_DATA                            // 24 elements, 0xE0 bytes
(sizeof)
{
    /*0x000*/     struct _LIST_ENTRY InLoadOrderLinks;                    // 2
elements, 0x10 bytes (sizeof)
    /*0x010*/     struct _LIST_ENTRY InMemoryOrderLinks;                  // 2
elements, 0x10 bytes (sizeof)
    /*0x020*/     struct _LIST_ENTRY InInitializationOrderLinks;          // 2
elements, 0x10 bytes (sizeof)
    /*0x030*/     VOID*        DllBase;
    /*0x038*/     VOID*        EntryPoint;
    /*0x040*/     ULONG32      SizeOfImage;
    /*0x044*/     UINT8        _PADDING0_[0x4];
    /*0x048*/     struct _UNICODE_STRING FullDllName;                     // 3
elements, 0x10 bytes (sizeof)
    /*0x058*/     struct _UNICODE_STRING BaseDllName;                     // 3
elements, 0x10 bytes (sizeof)
    /*0x068*/     ULONG32      Flags;
    /*0x06C*/     UINT16       LoadCount;
    /*0x06E*/     UINT16       TlsIndex;
    union                                                   // 2 elements, 0x10
bytes (sizeof)
```

```
    {
        /*0x070*/         struct _LIST_ENTRY HashLinks;
 // 2 elements, 0x10 bytes (sizeof)
        struct                                                  // 2 elements, 0x10
bytes (sizeof)
        {
            /*0x070*/             VOID*        SectionPointer;
            /*0x078*/             ULONG32      CheckSum;
            /*0x07C*/             UINT8        _PADDING1_[0x4];
        };
    };
    union                                                       // 2 elements, 0x8
bytes (sizeof)
    {
        /*0x080*/         ULONG32      TimeDateStamp;
        /*0x080*/         VOID*        LoadedImports;
    };
    /*0x088*/     struct _ACTIVATION_CONTEXT* EntryPointActivationContext;
    /*0x090*/     VOID*        PatchInformation;
    /*0x098*/     struct _LIST_ENTRY ForwarderLinks;                        // 2
elements, 0x10 bytes (sizeof)
    /*0x0A8*/     struct _LIST_ENTRY ServiceTagLinks;                       // 2
elements, 0x10 bytes (sizeof)
    /*0x0B8*/     struct _LIST_ENTRY StaticLinks;                           // 2
elements, 0x10 bytes (sizeof)
    /*0x0C8*/     VOID*        ContextInformation;
    /*0x0D0*/     UINT64       OriginalBase;
    /*0x0D8*/     union _LARGE_INTEGER LoadTime;                            // 4
elements, 0x8 bytes (sizeof)
}LDR_DATA, *PLDR_DATA;

typedef struct _OBJECT_TYPE_INITIALIZER

             // 25 elements, 0x70 bytes (sizeof)
{
    /*0x000*/     UINT16       Length;
    union

             // 2 elements, 0x1 bytes (sizeof)
    {
        /*0x002*/         UINT8        ObjectTypeFlags;
        struct

            // 7 elements, 0x1 bytes (sizeof)
        {
            /*0x002*/             UINT8        CaseInsensitive : 1;

                                   // 0 BitPosition
            /*0x002*/             UINT8        UnnamedObjectsOnly : 1;

                                  // 1 BitPosition
            /*0x002*/             UINT8        UseDefaultObject : 1;

                                  // 2 BitPosition
```

```
            /*0x002*/             UINT8        SecurityRequired : 1;

                                   // 3 BitPosition
            /*0x002*/             UINT8        MaintainHandleCount : 1;

                                    // 4 BitPosition
            /*0x002*/             UINT8        MaintainTypeList : 1;

                                   // 5 BitPosition
            /*0x002*/             UINT8        SupportsObjectCallbacks : 1;

                                    // 6 BitPosition
        };
    };
    /*0x004*/     ULONG32      ObjectTypeCode;
    /*0x008*/     ULONG32      InvalidAttributes;
    /*0x00C*/     struct _GENERIC_MAPPING GenericMapping;

                              // 4 elements, 0x10 bytes (sizeof)
    /*0x01C*/     ULONG32      ValidAccessMask;
    /*0x020*/     ULONG32      RetainAccess;
    /*0x024*/     enum _POOL_TYPE PoolType;
    /*0x028*/     ULONG32      DefaultPagedPoolCharge;
    /*0x02C*/     ULONG32      DefaultNonPagedPoolCharge;
    /*0x030*/     PVOID DumpProcedure;
    /*0x038*/     PVOID OpenProcedure;
    /*0x040*/     PVOID CloseProcedure;
    /*0x048*/     PVOID DeleteProcedure;
    /*0x050*/     PVOID ParseProcedure;
    /*0x058*/     PVOID SecurityProcedure;
    /*0x060*/     PVOID QueryNameProcedure;
    /*0x068*/     PVOID OkayToCloseProcedure;
}OBJECT_TYPE_INITIALIZER, *POBJECT_TYPE_INITIALIZER;

typedef struct _EX_PUSH_LOCK                // 7 elements, 0x8 bytes (sizeof)
{
    union                                   // 3 elements, 0x8 bytes (sizeof)
    {
        struct                              // 5 elements, 0x8 bytes (sizeof)
        {
            /*0x000*/             UINT64       Locked : 1;         // 0
BitPosition
            /*0x000*/             UINT64       Waiting : 1;        // 1
BitPosition
            /*0x000*/             UINT64       Waking : 1;         // 2
BitPosition
            /*0x000*/             UINT64       MultipleShared : 1; // 3
BitPosition
            /*0x000*/             UINT64       Shared : 60;        // 4
BitPosition
        };
        /*0x000*/         UINT64       Value;
        /*0x000*/         VOID*        Ptr;
    };
}EX_PUSH_LOCK, *PEX_PUSH_LOCK;
```

```
typedef struct _MY_OBJECT_TYPE                    // 12 elements, 0xD0 bytes
(sizeof)
{
    /*0x000*/     struct _LIST_ENTRY TypeList;              // 2 elements, 0x10
bytes (sizeof)
    /*0x010*/     struct _UNICODE_STRING Name;              // 3 elements, 0x10
bytes (sizeof)
    /*0x020*/     VOID*         DefaultObject;
    /*0x028*/     UINT8         Index;
    /*0x029*/     UINT8         _PADDING0_[0x3];
    /*0x02C*/     ULONG32       TotalNumberOfObjects;
    /*0x030*/     ULONG32       TotalNumberOfHandles;
    /*0x034*/     ULONG32       HighWaterNumberOfObjects;
    /*0x038*/     ULONG32       HighWaterNumberOfHandles;
    /*0x03C*/     UINT8         _PADDING1_[0x4];
    /*0x040*/     struct _OBJECT_TYPE_INITIALIZER TypeInfo; // 25 elements, 0x70
bytes (sizeof)
    /*0x0B0*/     struct _EX_PUSH_LOCK TypeLock;            // 7 elements, 0x8
bytes (sizeof)
    /*0x0B8*/     ULONG32       Key;
    /*0x0BC*/     UINT8         _PADDING2_[0x4];
    /*0x0C0*/     struct _LIST_ENTRY CallbackList;          // 2 elements, 0x10
bytes (sizeof)
}MY_OBJECT_TYPE, *PMY_OBJECT_TYPE;
```

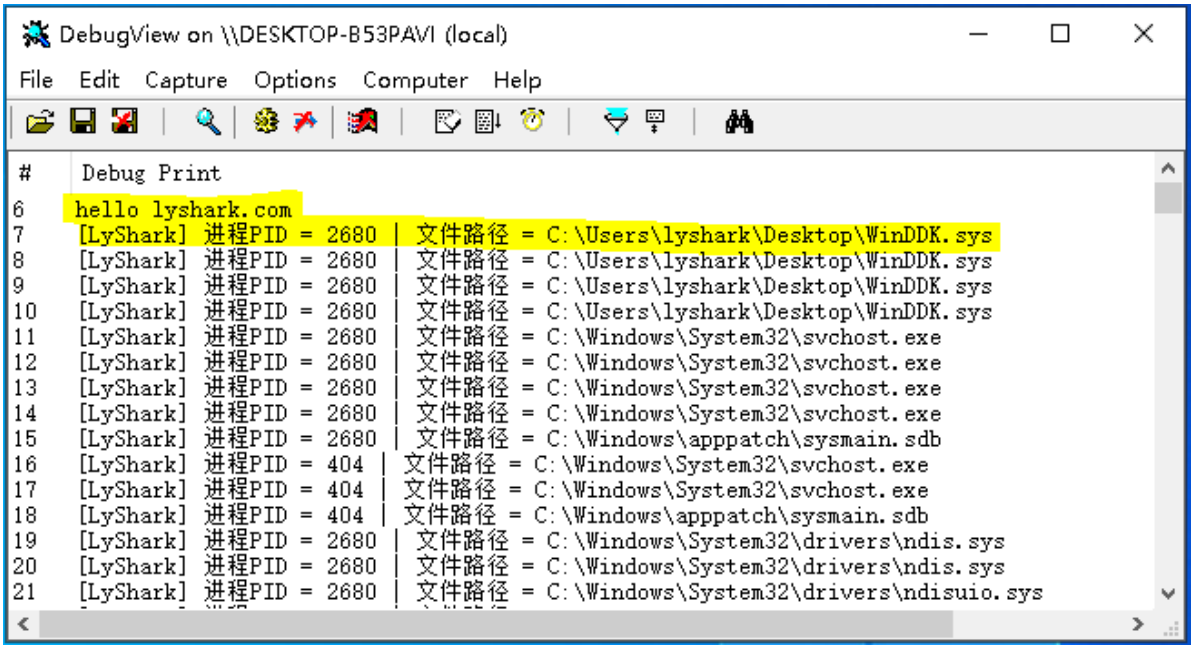
对于开启了`TypeInfo.SupportsObjectCallbacks`属性的驱动来说自然就支持文件路径转换，当系统中有文件被加载则自动执行`LySharkFileObjectpreCall`回调事件，过滤掉无效路径后即可直接输出，完整代码如下所示；

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

#include "lyshark.h"

PVOID obHandle;
DRIVER_INITIALIZE DriverEntry;

// 文件回调
OB_PREOP_CALLBACK_STATUS LySharkFileObjectpreCall(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION OperationInformation)
{
    UNICODE_STRING DosName;
    PFILE_OBJECT fileo = OperationInformation->Object;
    HANDLE CurrentProcessId = PsGetCurrentProcessId();
    UNREFERENCED_PARAMETER(RegistrationContext);

    if (OperationInformation->ObjectType != *IoFileObjectType)
    {
        return OB_PREOP_SUCCESS;
    }

    // 过滤无效指针
```

```
    if (fileo->FileName.Buffer == NULL ||
        !MmIsAddressValid(fileo->FileName.Buffer) ||
        fileo->DeviceObject == NULL ||
        !MmIsAddressValid(fileo->DeviceObject))
    {
        return OB_PREOP_SUCCESS;
    }

    // 过滤无效路径
    if (!_wcsicmp(fileo->FileName.Buffer, L"\\Endpoint") ||
        !_wcsicmp(fileo->FileName.Buffer, L"?") ||
        !_wcsicmp(fileo->FileName.Buffer, L"\\.\\.") ||
        !_wcsicmp(fileo->FileName.Buffer, L"\\"))
    {
        return OB_PREOP_SUCCESS;
    }

    // 将对象转为DOS路径
    RtlVolumeDeviceToDosName(fileo->DeviceObject, &DosName);
    DbgPrint("[LyShark] 进程PID = %ld | 文件路径 = %wZ%wZ \n",
(ULONG64)CurrentProcessId, &DosName, &fileo->FileName);

    return OB_PREOP_SUCCESS;
}

VOID EnableObType(POBJECT_TYPE ObjectType)
{
    PMY_OBJECT_TYPE myobtype = (PMY_OBJECT_TYPE)ObjectType;
    myobtype->TypeInfo.SupportsObjectCallbacks = 1;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    UNREFERENCED_PARAMETER(driver);
    ObUnRegisterCallbacks(obHandle);
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    NTSTATUS status = STATUS_SUCCESS;
    PLDR_DATA ldr;

    DbgPrint("hello lyshark.com \n");

    OB_CALLBACK_REGISTRATION obRegFileCallBack;
    OB_OPERATION_REGISTRATION opRegFileCallBack;

    // enable IoFileObjectType
    EnableObType(*IoFileObjectType);

    // bypass MmVerifyCallbackFunction
    ldr = (PLDR_DATA)Driver->DriverSection;
    ldr->Flags |= 0x20;

    // 初始化回调
```

```
    memset(&obRegFileCallBack, 0, sizeof(obRegFileCallBack));
    obRegFileCallBack.Version = ObGetFilterVersion();
    obRegFileCallBack.OperationRegistrationCount = 1;
    obRegFileCallBack.RegistrationContext = NULL;
    RtlInitUnicodeString(&obRegFileCallBack.Altitude, L"321000");
    obRegFileCallBack.OperationRegistration = &opRegFileCallBack;

    memset(&opRegFileCallBack, 0, sizeof(opRegFileCallBack));
    opRegFileCallBack.ObjectType = IoFileObjectType;
    opRegFileCallBack.Operations = OB_OPERATION_HANDLE_CREATE |
OB_OPERATION_HANDLE_DUPLICATE;
    opRegFileCallBack.PreOperation =
(POB_PRE_OPERATION_CALLBACK)&LySharkFileObjectpreCall;

    status = ObRegisterCallbacks(&obRegFileCallBack, &obHandle);
    if (!NT_SUCCESS(status))
    {
        DbgPrint("注册回调错误 \n");
        status = STATUS_UNSUCCESSFUL;
    }

    UNREFERENCED_PARAMETER(RegistryPath);
    Driver->DriverUnload = &UnDriver;
    return status;
}
```

运行这个驱动程序，当系统中有新文件被加载时则自动输出该文件所属进程PID以及该文件的详细路径。

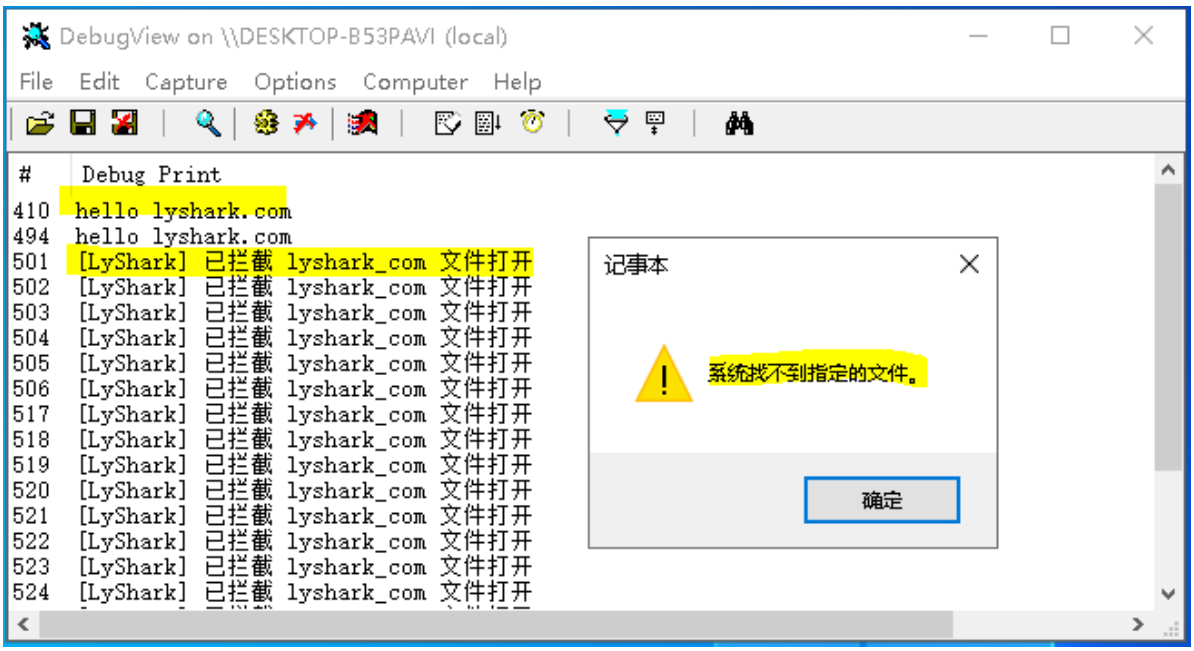


至于如何阻止打开一个文件其实与`《驱动开发：内核注册并监控对象回调》`文章中使用的方法是一致的，首先判断`OperationInformation->Operation`是不是`OB_OPERATION_HANDLE_CREATE`或`OB_OPERATION_HANDLE_DUPLICATE`如果是，则直接设置`Parameters->CreateHandleInformation.DesiredAccess`为0直接拒绝加载。

```
// 文件回调
OB_PREOP_CALLBACK_STATUS LySharkFileObjectpreCall(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION OperationInformation)
{
```

```
    UNICODE_STRING DosName;
    PFILE_OBJECT fileo = OperationInformation->Object;
    HANDLE CurrentProcessId = PsGetCurrentProcessId();
    UNREFERENCED_PARAMETER(RegistrationContext);

    if (OperationInformation->ObjectType != *IoFileObjectType)
    {
        return OB_PREOP_SUCCESS;
    }

    // 过滤无效指针
    if (fileo->FileName.Buffer == NULL ||
        !MmIsAddressValid(fileo->FileName.Buffer) ||
        fileo->DeviceObject == NULL ||
        !MmIsAddressValid(fileo->DeviceObject))
    {
        return OB_PREOP_SUCCESS;
    }

    // 过滤无效路径
    if (!_wcsicmp(fileo->FileName.Buffer, L"\\Endpoint") ||
        !_wcsicmp(fileo->FileName.Buffer, L"?") ||
        !_wcsicmp(fileo->FileName.Buffer, L"\\.\\.") ||
        !_wcsicmp(fileo->FileName.Buffer, L"\\"))
    {
        return OB_PREOP_SUCCESS;
    }

    // 阻止打开lyshark_com.txt文本
    if (wcsstr(_wcslwr(fileo->FileName.Buffer), L"lyshark_com.txt"))
    {
        if (OperationInformation->Operation == OB_OPERATION_HANDLE_CREATE)
        {
            OperationInformation->Parameters-
>CreateHandleInformation.DesiredAccess = 0;
        }
        if (OperationInformation->Operation == OB_OPERATION_HANDLE_DUPLICATE)
        {
            OperationInformation->Parameters-
>DuplicateHandleInformation.DesiredAccess = 0;
        }
        DbgPrint("[LyShark] 已拦截 lyshark_com 文件打开 \n");
    }

    return OB_PREOP_SUCCESS;
}
```

运行修改后的驱动程序，然后尝试打开`lyshark_com.txt`则会提示系统找不到指定文件。



作者： 王瑞 (LyShark)

作者邮箱： me@lyshark.com