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](mailto: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)

{

BitPosition

BitPosition

BitPosition

BitPosition

BitPosition

};

/ 0x000 / UINT64 Locked : 1; // 0

/ 0x000 / UINT64 Waiting : 1; // 1

/ 0x000 / UINT64 Waking : 1; // 2

/ 0x000 / UINT64 MultipleShared : 1; // 3

/ 0x000 / UINT64 Shared : 60; // 4

/ 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](mailto: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;

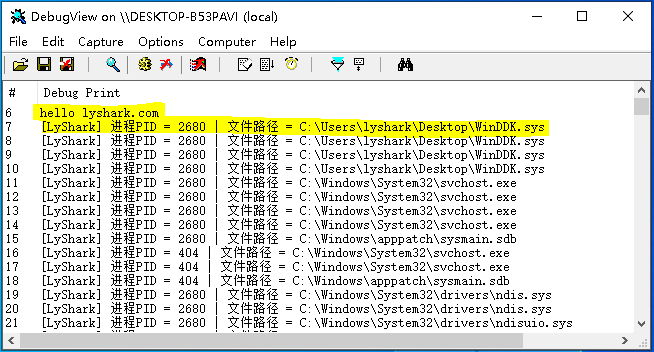
}

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;

}

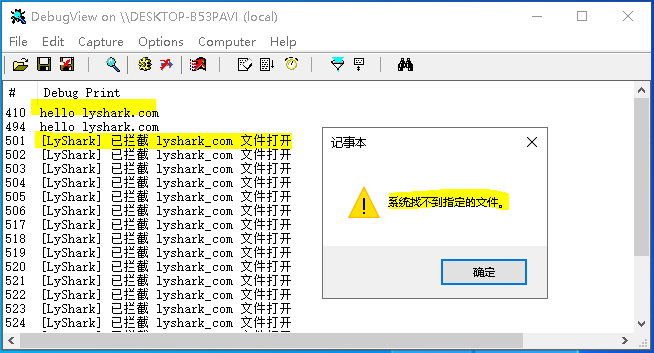
DbgPrint("[LyShark] 已拦截 lyshark\_com 文件打开 \n");

}

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： m [e@lyshark.com](mailto:e@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！