

在笔者上一篇文章《驱动开发：内核枚举Registry注册表回调》中我们通过特征码定位实现了对注册表回调的枚举，本篇文章 LyShark 将教大家如何枚举系统中的 ProcessObCall 进程回调以及 ThreadObCall 线程回调，之所以放在一起讲解是因为这两中回调在枚举是都需要使用通用结构体 _OB_CALLBACK 以及 _OBJECT_TYPE 所以放在一起讲解最好不过。

我们来看一款闭源ARK工具是如何实现的：

进程	驱动模块	内核层	内核钩子	应用层钩子	设置	监控	启动信息	注册表	服务	文件	网络	调试引擎
系统回调	过滤驱动	DPC定时器	IO定时器	系统线程	卸载的驱动							
回调入口		通知类型										
0xFFFFF80636A2D760		ThreadObCall										
0xFFFFF8063C39D890		Registry										
0xFFFFF8063A065BE0		Registry										
0xFFFFF8063C3A8410		ProcessObCall										
0xFFFFF80636A2D420		ProcessObCall										
0xFFFFF80636A2C550		LoadImage										
		模块路径										
		C:\Windows\System32\drivers\PYArkSafe.sys										
		C:\Windows\system32\drivers\WdFilter.sys										
		C:\Windows\system32\ntoskrnl.exe										
		C:\Windows\system32\drivers\WdFilter.sys										
		C:\Windows\System32\drivers\PYArkSafe.sys										
		C:\Windows\System32\drivers\PYArkSafe.sys										

首先我们需要定义好结构体，结构体是微软公开的，如果有其它需要请自行去微软官方去查。

```
typedef struct _OBJECT_TYPE_INITIALIZER
{
    USHORT Length; // Uint2B
    UCHAR ObjectTypeFlags; // Uchar
    ULONG ObjectTypeCode; // Uint4B
    ULONG InvalidAttributes; // Uint4B
    GENERIC_MAPPING GenericMapping; // _GENERIC_MAPPING
    ULONG ValidAccessMask; // Uint4B
    ULONG RetainAccess; // Uint4B
    POOL_TYPE PoolType; // _POOL_TYPE
    ULONG DefaultPagedPoolCharge; // Uint4B
    ULONG DefaultNonPagedPoolCharge; // Uint4B
    PVOID DumpProcedure; // Ptr64 void
    PVOID OpenProcedure; // Ptr64 long
    PVOID CloseProcedure; // Ptr64 void
    PVOID DeleteProcedure; // Ptr64 void
    PVOID ParseProcedure; // Ptr64 long
    PVOID SecurityProcedure; // Ptr64 long
    PVOID QueryNameProcedure; // Ptr64 long
    PVOID OkayToCloseProcedure; // Ptr64 unsigned char
    ULONG WaitObjectFlagMask; // Uint4B
    USHORT WaitObjectFlagOffset; // Uint2B
    USHORT WaitObjectPointerOffset; // Uint2B
}OBJECT_TYPE_INITIALIZER, *POBJECT_TYPE_INITIALIZER;

typedef struct _OBJECT_TYPE
{
    LIST_ENTRY TypeList; // _LIST_ENTRY
    UNICODE_STRING Name; // _UNICODE_STRING
    PVOID DefaultObject; // Ptr64 Void
    UCHAR Index; // Uchar
    ULONG TotalNumberOfObjects; // Uint4B
    ULONG TotalNumberOfHandles; // Uint4B
    ULONG HighWaterNumberOfObjects; // Uint4B
    ULONG HighWaterNumberOfHandles; // Uint4B
}
```

```

OBJECT_TYPE_INITIALIZER TypeInfo; // _OBJECT_TYPE_INITIALIZER
EX_PUSH_LOCK TypeLock;          // _EX_PUSH_LOCK
ULONG Key;                       // Uint4B
LIST_ENTRY CallbackList;         // _LIST_ENTRY
}OBJECT_TYPE, *POBJECT_TYPE;

#pragma pack(1)
typedef struct _OB_CALLBACK
{
    LIST_ENTRY ListEntry;
    ULONGLONG Unknown;
    HANDLE ObHandle;
    PVOID ObTypeAddr;
    PVOID PreCall;
    PVOID PostCall;
}OB_CALLBACK, *POB_CALLBACK;
#pragma pack()

```

代码部分的实现很容易，由于进程与线程句柄的枚举很容易，直接通过（POBJECT_TYPE）（*PsProcessType）->CallbackList 就可以拿到链表头结构，得到后将其解析为 POB_CALLBACK 并循环输出即可。

```

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

typedef struct _OBJECT_TYPE_INITIALIZER
{
    USHORT Length; // Uint2B
    UCHAR ObjectTypeFlags; // UChar
    ULONG ObjectTypeCode; // Uint4B
    ULONG InvalidAttributes; // Uint4B
    GENERIC_MAPPING GenericMapping; // _GENERIC_MAPPING
    ULONG ValidAccessMask; // Uint4B
    ULONG RetainAccess; // Uint4B
    POOL_TYPE PoolType; // _POOL_TYPE
    ULONG DefaultPagedPoolCharge; // Uint4B
    ULONG DefaultNonPagedPoolCharge; // Uint4B
    PVOID DumpProcedure; // Ptr64 void
    PVOID OpenProcedure; // Ptr64 long
    PVOID CloseProcedure; // Ptr64 void
    PVOID DeleteProcedure; // Ptr64 void
    PVOID ParseProcedure; // Ptr64 long
    PVOID SecurityProcedure; // Ptr64 long
    PVOID QueryNameProcedure; // Ptr64 long
    PVOID OkayToCloseProcedure; // Ptr64 unsigned char
    ULONG WaitObjectFlagMask; // Uint4B
    USHORT WaitObjectFlagOffset; // Uint2B
    USHORT WaitObjectPointerOffset; // Uint2B
}OBJECT_TYPE_INITIALIZER, *POBJECT_TYPE_INITIALIZER;

```

```

typedef struct _OBJECT_TYPE
{
    LIST_ENTRY TypeList;           // _LIST_ENTRY
    UNICODE_STRING Name;          // _UNICODE_STRING
    PVOID DefaultObject;          // Ptr64 Void
    UCHAR Index;                  // UChar
    ULONG TotalNumberOfObjects;    // Uint4B
    ULONG TotalNumberOfHandles;    // Uint4B
    ULONG HighWaterNumberOfObjects; // Uint4B
    ULONG HighWaterNumberOfHandles; // Uint4B
    OBJECT_TYPE_INITIALIZER TypeInfo; // _OBJECT_TYPE_INITIALIZER
    EX_PUSH_LOCK TypeLock;        // _EX_PUSH_LOCK
    ULONG Key;                    // Uint4B
    LIST_ENTRY CallbackList;      // _LIST_ENTRY
}OBJECT_TYPE, *POBJECT_TYPE;

#pragma pack(1)
typedef struct _OB_CALLBACK
{
    LIST_ENTRY ListEntry;
    ULONGLONG Unknown;
    HANDLE ObHandle;
    PVOID ObTypeAddr;
    PVOID PreCall;
    PVOID PostCall;
}OB_CALLBACK, *POB_CALLBACK;
#pragma pack()

VOID DriverUnload(PDRIVER_OBJECT pDriverObject)
{
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriverObject, PUNICODE_STRING pRegPath)
{
    NTSTATUS status = STATUS_SUCCESS;

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

    POB_CALLBACK pobCallback = NULL;

    // 直接获取 CallbackList 链表
    LIST_ENTRY CallbackList = ((POBJECT_TYPE)(*PsProcessType))->CallbackList;

    // 开始遍历
    pobCallback = (POB_CALLBACK)CallbackList.Flink;
    do
    {
        if (FALSE == MmIsAddressValid(pobCallback))
        {
            break;
        }
        if (NULL != pobCallback->ObHandle)
        {
            // 显示

```

```

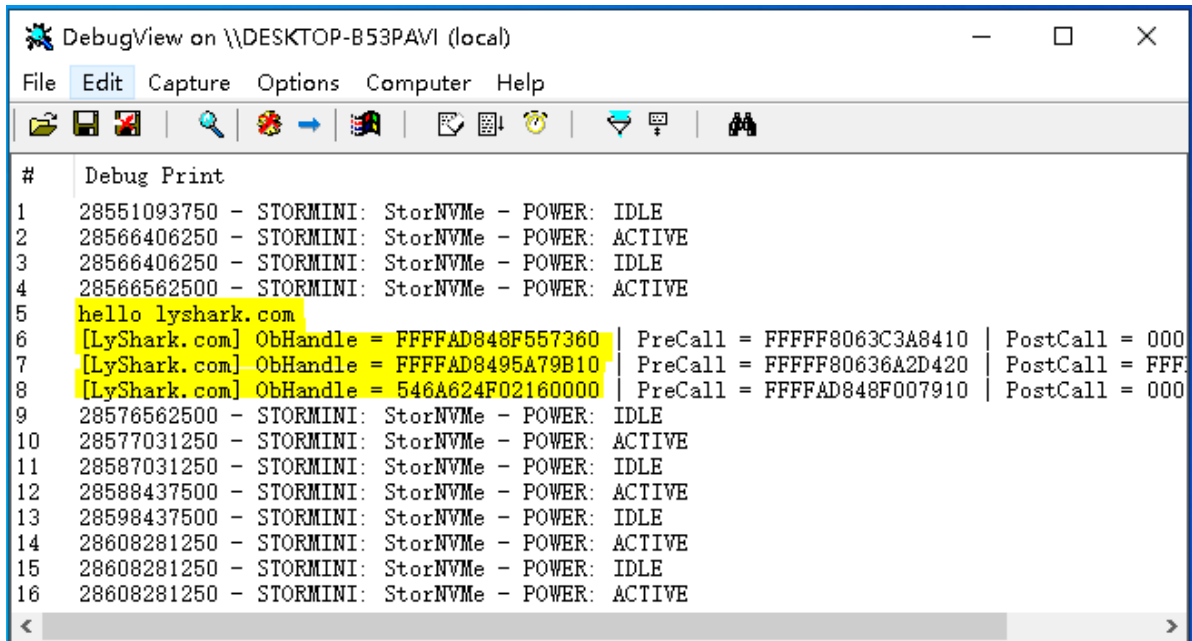
        DbgPrint("[LyShark.com] ObHandle = %p | PreCall = %p | PostCall = %p\n", pobCallback->ObHandle, pobCallback->PreCall, pobCallback->PostCall);

    }
    // 获取下一链表信息
    pobCallback = (POB_CALLBACK)pobCallback->ListEntry.Flink;

} while (CallbackList.Flink != (PLIST_ENTRY)pobCallback);
return status;
}

```

运行这段驱动程序，即可得到 `进程句柄` 回调：



```

#   Debug Print
1   28551093750 - STORMINI: StorNVMe - POWER: IDLE
2   28566406250 - STORMINI: StorNVMe - POWER: ACTIVE
3   28566406250 - STORMINI: StorNVMe - POWER: IDLE
4   28566562500 - STORMINI: StorNVMe - POWER: ACTIVE
5   hello lyshark.com
6   [LyShark.com] ObHandle = FFFFAD848F557360 | PreCall = FFFFF8063C3A8410 | PostCall = 000
7   [LyShark.com] ObHandle = FFFFAD8495A79B10 | PreCall = FFFFF80636A2D420 | PostCall = FFF
8   [LyShark.com] ObHandle = 546A624F02160000 | PreCall = FFFFAD848F007910 | PostCall = 000
9   28576562500 - STORMINI: StorNVMe - POWER: IDLE
10  28577031250 - STORMINI: StorNVMe - POWER: ACTIVE
11  28587031250 - STORMINI: StorNVMe - POWER: IDLE
12  28588437500 - STORMINI: StorNVMe - POWER: ACTIVE
13  28598437500 - STORMINI: StorNVMe - POWER: IDLE
14  28608281250 - STORMINI: StorNVMe - POWER: ACTIVE
15  28608281250 - STORMINI: StorNVMe - POWER: IDLE
16  28608281250 - STORMINI: StorNVMe - POWER: ACTIVE

```

当然了如是是 `进程句柄` 的枚举，如果是想要输出线程句柄，则只需要替换代码中的 `PsProcessType` 为 `((POBJECT_TYPE)(*PsThreadType))->CallbackList` 即可，修改后的代码如下。

```

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

typedef struct _OBJECT_TYPE_INITIALIZER
{
    USHORT Length; // Uint2B
    UCHAR ObjectTypeFlags; // Uchar
    ULONG ObjectTypeCode; // Uint4B
    ULONG InvalidAttributes; // Uint4B
    GENERIC_MAPPING GenericMapping; // _GENERIC_MAPPING
    ULONG ValidAccessMask; // Uint4B
    ULONG RetainAccess; // Uint4B
    POOL_TYPE PoolType; // _POOL_TYPE
    ULONG DefaultPagedPoolCharge; // Uint4B
    ULONG DefaultNonPagedPoolCharge; // Uint4B
    PVOID DumpProcedure; // Ptr64 void
}

```

```

PVOID OpenProcedure;      // Ptr64    long
PVOID CloseProcedure;     // Ptr64    void
PVOID DeleteProcedure;    // Ptr64    void
PVOID ParseProcedure;     // Ptr64    long
PVOID SecurityProcedure;  // Ptr64    long
PVOID QueryNameProcedure; // Ptr64    long
PVOID OkayToCloseProcedure; // Ptr64    unsigned char
ULONG WaitObjectFlagMask; // Uint4B
USHORT WaitObjectFlagOffset; // Uint2B
USHORT WaitObjectPointerOffset; // Uint2B
}OBJECT_TYPE_INITIALIZER, *POBJECT_TYPE_INITIALIZER;

typedef struct _OBJECT_TYPE
{
    LIST_ENTRY TypeList;          // _LIST_ENTRY
    UNICODE_STRING Name;         // _UNICODE_STRING
    PVOID DefaultObject;         // Ptr64 Void
    UCHAR Index;                 // Uchar
    ULONG TotalNumberOfObjects;   // Uint4B
    ULONG TotalNumberOfHandles;   // Uint4B
    ULONG HighWaterNumberOfObjects; // Uint4B
    ULONG HighWaterNumberOfHandles; // Uint4B
    OBJECT_TYPE_INITIALIZER TypeInfo; // _OBJECT_TYPE_INITIALIZER
    EX_PUSH_LOCK TypeLock;       // _EX_PUSH_LOCK
    ULONG Key;                   // Uint4B
    LIST_ENTRY CallbackList;     // _LIST_ENTRY
}OBJECT_TYPE, *POBJECT_TYPE;

#pragma pack(1)
typedef struct _OB_CALLBACK
{
    LIST_ENTRY ListEntry;
    ULONGLONG Unknown;
    HANDLE ObHandle;
    PVOID ObTypeAddr;
    PVOID PreCall;
    PVOID PostCall;
}OB_CALLBACK, *POB_CALLBACK;
#pragma pack()

// 移除回调
NTSTATUS RemoveObCallback(PVOID RegistrationHandle)
{
    ObUnRegisterCallbacks(RegistrationHandle);

    return STATUS_SUCCESS;
}

VOID DriverUnload(PDRIVER_OBJECT pDriverObject)
{
}

NTSTATUS DriverEntry(PDRIVER_OBJECT pDriverObject, PUNICODE_STRING pRegPath)
{
    NTSTATUS status = STATUS_SUCCESS;

```

```

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

POB_CALLBACK pObCallback = NULL;

// 直接获取 CallbackList 链表
LIST_ENTRY CallbackList = ((POBJECT_TYPE)(*PsThreadType))->CallbackList;

// 开始遍历
pObCallback = (POB_CALLBACK)CallbackList.Flink;
do
{
    if (FALSE == MmIsAddressValid(pObCallback))
    {
        break;
    }
    if (NULL != pObCallback->ObHandle)
    {
        // 显示
        DbgPrint("[LyShark] ObHandle = %p | PreCall = %p | PostCall = %p\n", pObCallback->ObHandle, pObCallback->PreCall, pObCallback->PostCall);
    }
    // 获取下一链表信息
    pObCallback = (POB_CALLBACK)pObCallback->ListEntry.Flink;

} while (CallbackList.Flink != (PLIST_ENTRY)pObCallback);

return status;
}

```

运行这段驱动程序，即可得到 线程句柄 回调：

```

# Debug Print
4 31131562500 - STORMINI: StorNVMe - POWER: IDLE
5 31131718750 - STORMINI: StorNVMe - POWER: ACTIVE
6 hello lyshark.com
7 [LyShark] ObHandle = FFFFAD8495A79410 | PreCall = FFFFF80636A2D760 | PostCall = FFFFF80
8 [LyShark] ObHandle = 0100000000000000 | PreCall = 0000000000000000 | PostCall = 0005D10
9 31141718750 - STORMINI: StorNVMe - POWER: IDLE
10 31142187500 - STORMINI: StorNVMe - POWER: ACTIVE

```

作者：王瑞 (LyShark)

作者邮箱：me@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！