在笔者上一篇文章`《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，之所以放在一起来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`所以放在一起来讲解最好不过。

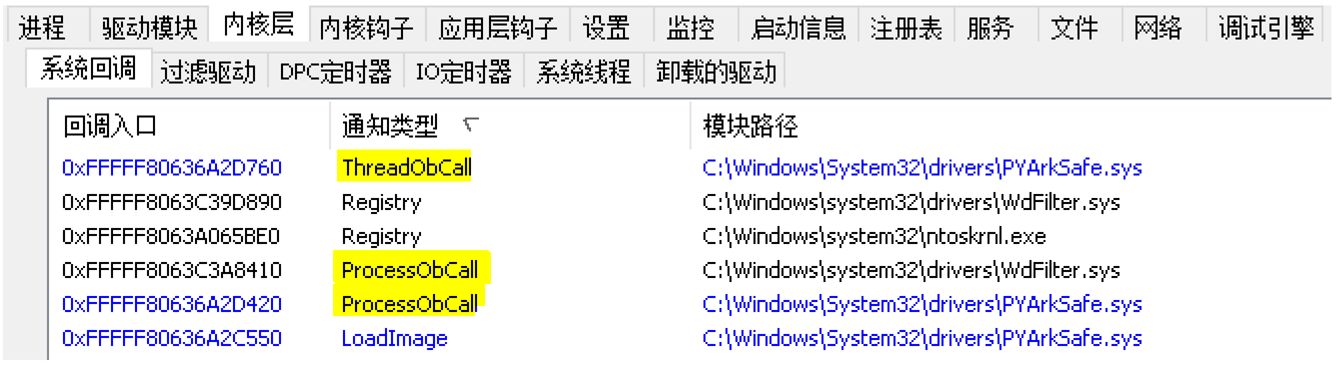
进程与线程ObCall回调是Windows操作系统提供的一种机制，它允许开发者在进程或线程发生创建、销毁、访问、修改等事件时拦截并处理这些事件。进程与线程ObCall回调是通过操作系统提供的回调机制来实现的。

当操作系统创建、销毁、访问或修改进程或线程时，它会触发进程与线程ObCall回调事件，然后在回调事件中调用注册的进程与线程ObCall回调函数。开发者可以在进程与线程ObCall回调函数中执行自定义的逻辑，例如记录日志，过滤敏感数据，或者阻止某些操作。

进程与线程ObCall回调可以通过操作系统提供的回调函数PsSetCreateProcessNotifyRoutine、PsSetCreateThreadNotifyRoutine、PsSetLoadImageNotifyRoutine等来进行注册。同时，进程与线程ObCall回调函数需要遵守一定的约束条件，例如不能阻塞或挂起进程或线程的创建或访问，不能调用一些内核API函数等。

进程与线程ObCall回调在安全软件、系统监控和调试工具等领域有着广泛的应用。

我们来看一款闭源ARK工具是如何实现的：



回调入口	通知类型	模块路径
0xFFFFF80636A2D760	ThreadObCall	C:\Windows\System32\drivers\PYArkSafe.sys
0xFFFFF8063C39D890	Registry	C:\Windows\system32\drivers\WdFilter.sys
0xFFFFF8063A065BE0	Registry	C:\Windows\system32\ntoskrnl.exe
0xFFFFF8063C3A8410	ProcessObCall	C:\Windows\system32\drivers\WdFilter.sys
0xFFFFF80636A2D420	ProcessObCall	C:\Windows\System32\drivers\PYArkSafe.sys
0xFFFFF80636A2C550	LoadImage	C:\Windows\System32\drivers\PYArkSafe.sys

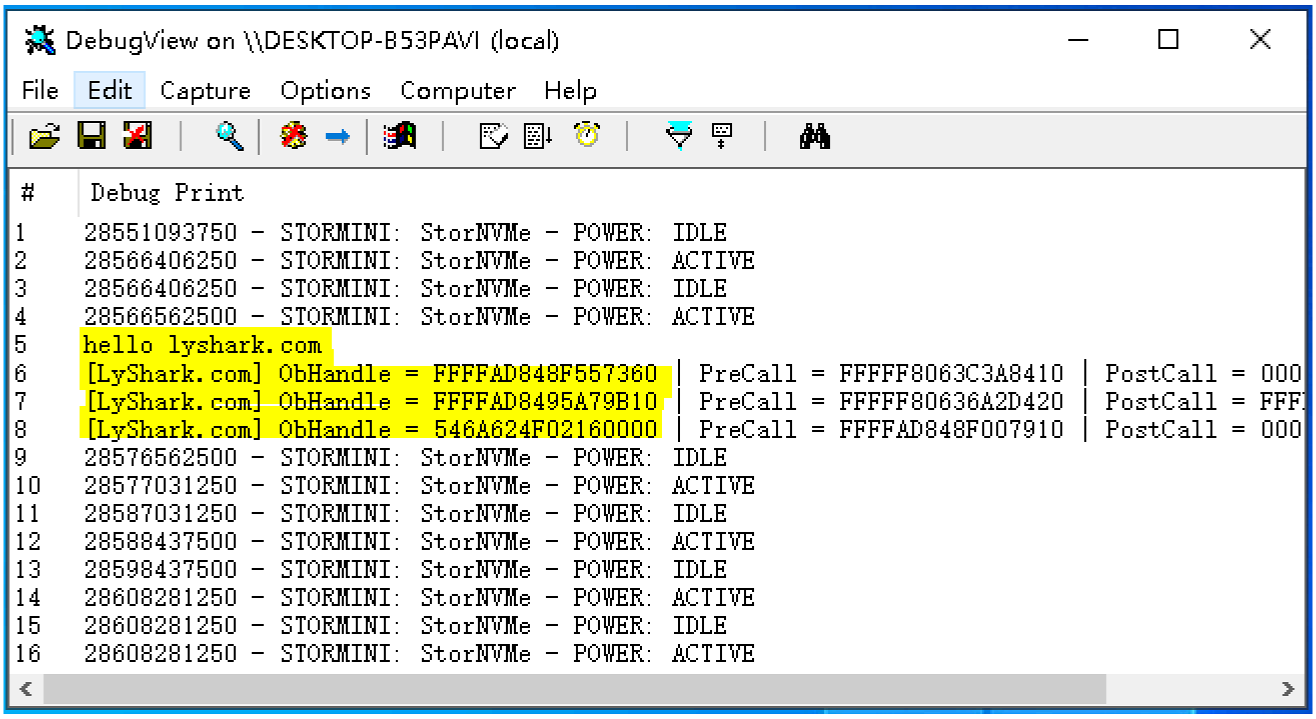
首先我们需要定义好结构体，结构体是微软公开的，如果有其它需要请自行去微软官方去查。

```
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。

```
#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
```

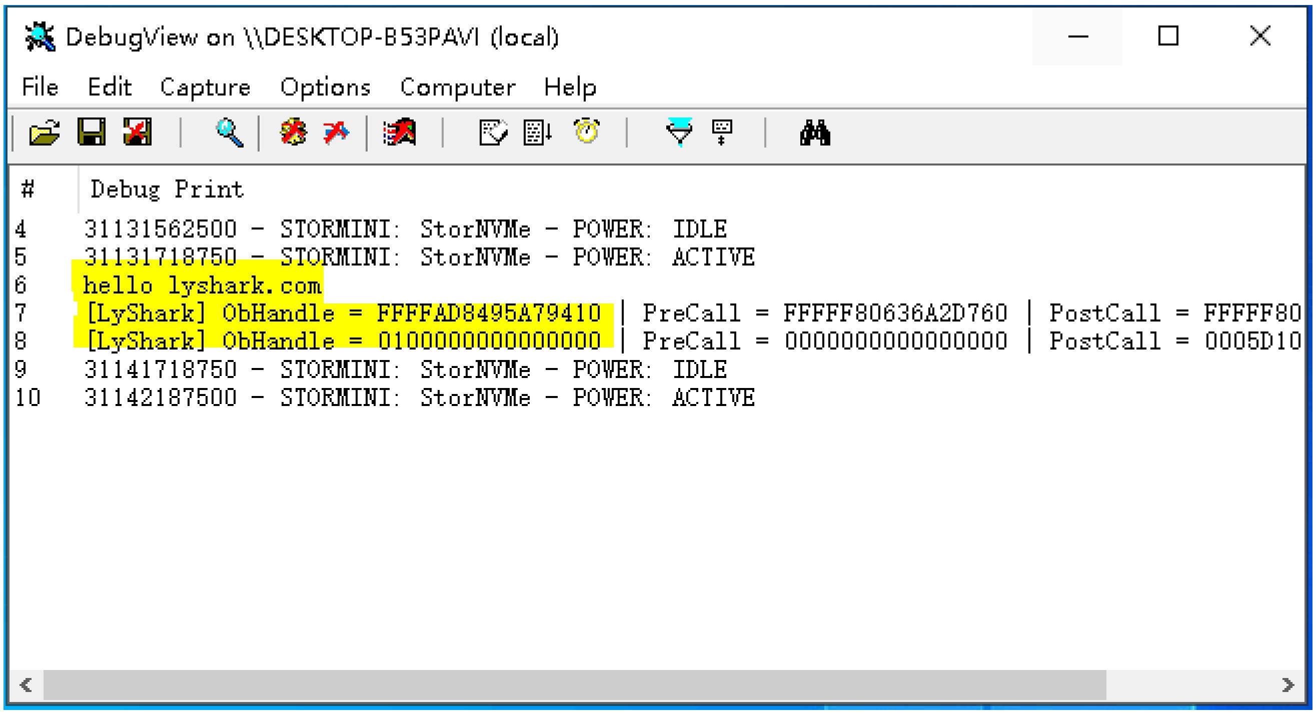
当然了如上是`进程句柄`的枚举，如果是想要输出线程句柄，则只需要替换代码中的`PsProcessType`为`((POBJECT_TYPE)(*PsThreadType))->CallbackList`即可，修改后的代码如下。

```
#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)(*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:


DebugView on \\DESKTOP-B53PAVI (local)
File Edit Capture Options Computer Help

```
#    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
```