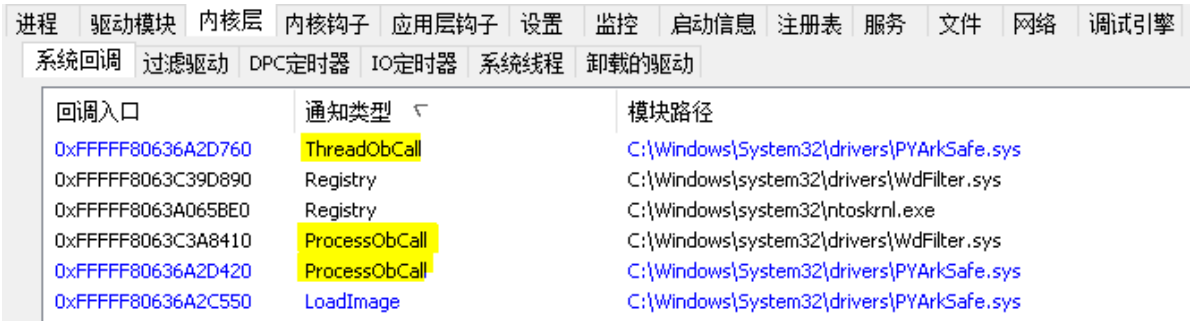
在笔者上一篇文章《驱动开发：内核枚举Registry注册表回调》中我们通过特征码定位实现了对注册表回调的枚举，本篇文章LyShark将教大家如何枚举系统中的ProcessObCall进程回调以及ThreadObCall线程回调，之所以放在一起来讲解是因为这两中回调在枚举是都需要使用通用结构体_OB_CALLBACK以及_OBJECT_TYPE所以放在一起来讲解最好不过。

我们来看一款闭源ARK工具是如何实现的：



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()
```

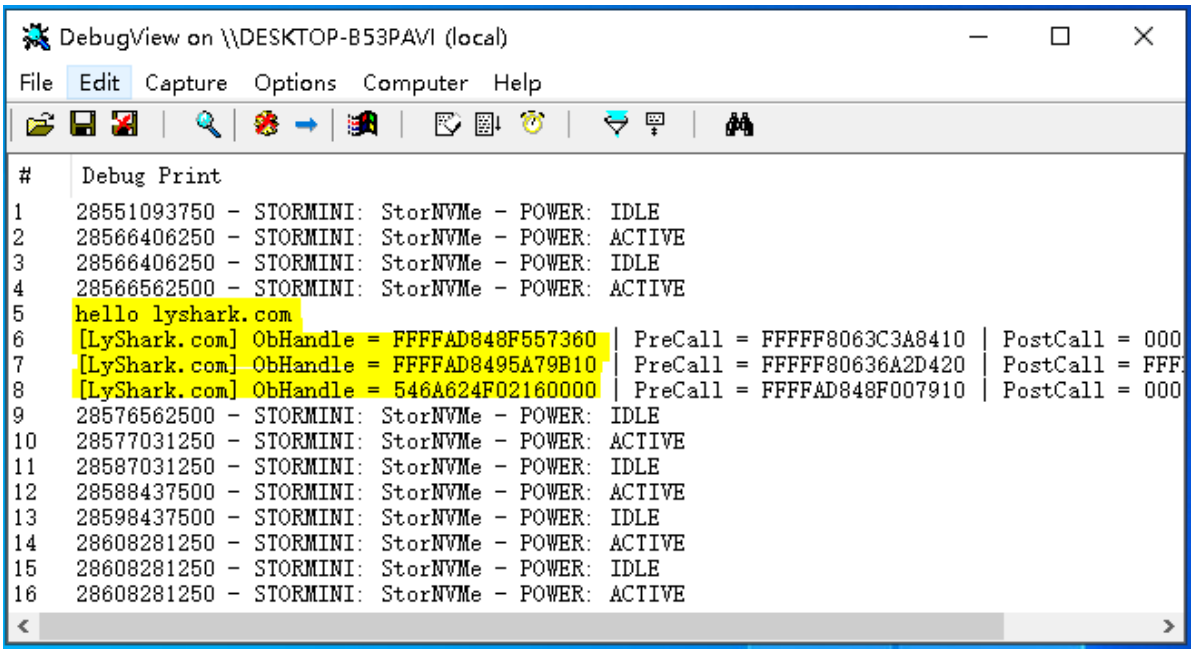
代码部分的实现很容易，由于进程与`线程句柄`的枚举很容易，直接通过`(POBJECT_TYPE)(*PsProcessType))->CallbackList`就可以拿到链表头结构，得到后将其解析为`POB_CALLBACK`并循环输出即可。

```
// 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:



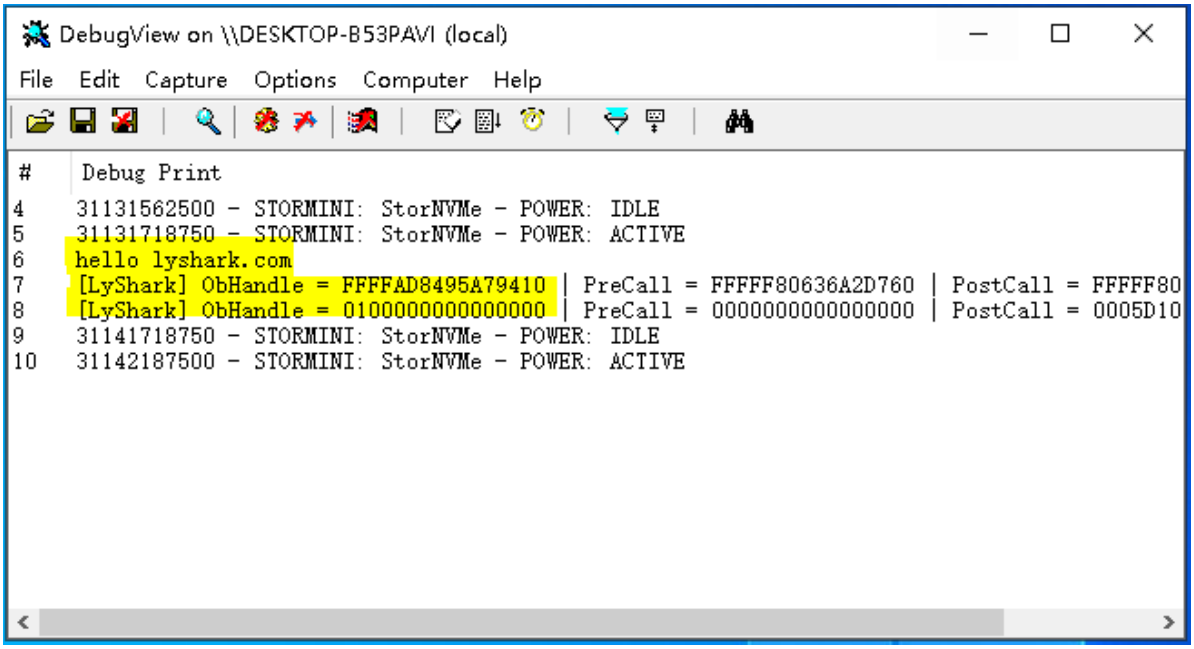
当然了如上是 `进程句柄` 的枚举，如果是想要输出线程句柄，则只需要替换代码中的 `PsProcessType` 为 `((POBJECT_TYPE)(*PsThreadType))->CallbackList` 即可，修改后的代码如下。

```
// 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:



> 作者： 王瑞 (LyShark)
>
> 作者邮箱： me@lyshark.com