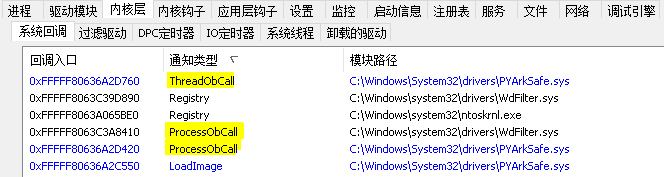
在笔者上一篇文章 《驱动开发：内核枚举Registry注册表回调》 中我们通过特征码定位实现了对注册表回调的枚举，本篇文章 LyShark 将教大家如何枚举系统中的 ProcessObCall 进程回调以及

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](mailto: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);

}

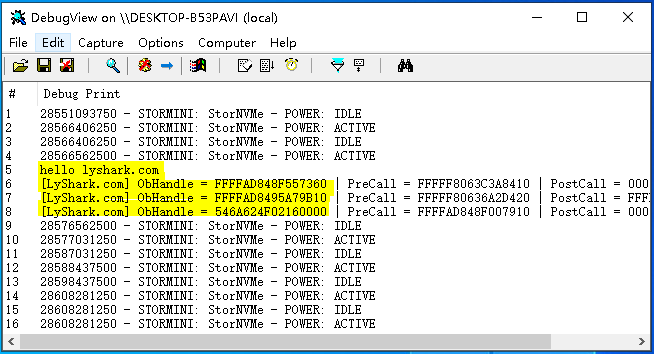
// 获取下一链表信息

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](mailto: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

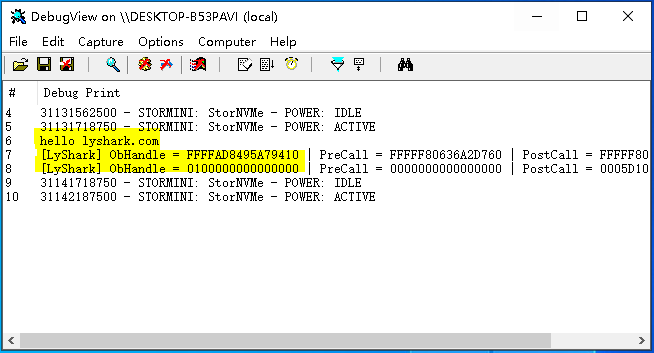
pObCallback = (POB\_CALLBACK)pObCallback->ListEntry.Flink;

} 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： 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！