在前面的文章中`LyShark`一直在重复的实现对系统底层模块的枚举，今天我们将展开一个新的话题，内核监控，我们以`监控进程线程`创建为例，在`Win10`系统中监控进程与线程可以使用微软提供给我们的两个新函数来实现，此类函数的原理是创建一个回调事件，当有进程或线程被创建或者注销时，系统会通过回调机制将该进程相关信息优先返回给我们自己的函数待处理结束后再转向系统层。

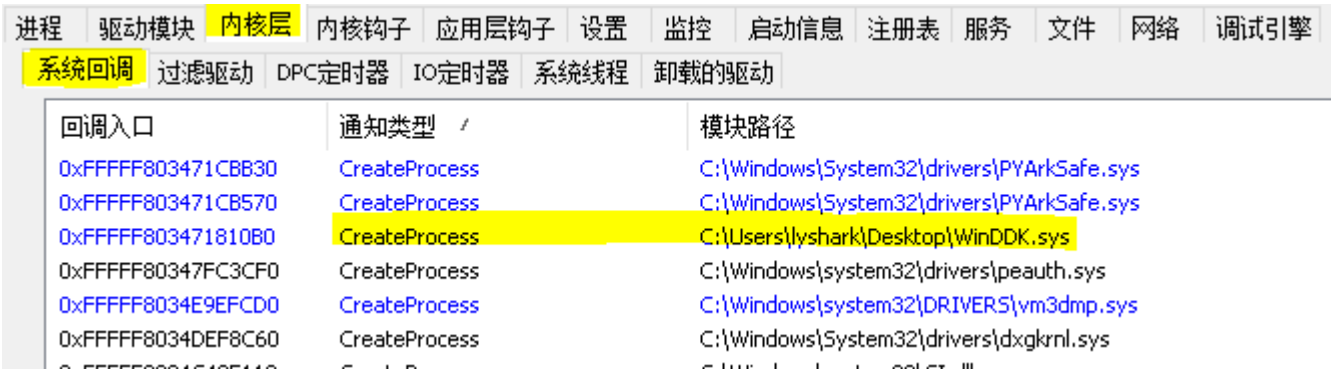
PsSetCreateProcessNotifyRoutineEx和PsSetCreateThreadNotifyRoutine是Windows操作系统提供的两个内核回调函数，它们允许开发者在进程或线程发生创建事件时拦截并处理这些事件。这两个函数提供的回调机制是操作系统提供的最基本、最常用的内核监控进程与线程的方式。

PsSetCreateProcessNotifyRoutineEx和PsSetCreateThreadNotifyRoutine的使用方式和参数类型类似，它们都需要开发者提供一个回调函数，当进程或线程被创建时，操作系统会调用这个回调函数。这个回调函数需要满足一定的约束条件，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。

PsSetCreateProcessNotifyRoutineEx和PsSetCreateThreadNotifyRoutine的主要区别在于它们所监控的事件不同。PsSetCreateProcessNotifyRoutineEx用于监控进程的创建事件，当有新的进程被创建时，操作系统会调用注册的回调函数。而PsSetCreateThreadNotifyRoutine用于监控线程的创建事件，当有新的线程被创建时，操作系统会调用注册的回调函数。

内核监控进程PsSetCreateProcessNotifyRoutineEx和线程PsSetCreateThreadNotifyRoutine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。需要注意的是，在Windows 8及更高版本的操作系统中，微软推荐开发者使用ExRegisterCallback和ExUnregisterCallback函数进行回调的注册和注销。

进程回调默认会设置`CreateProcess`通知，而线程回调则会设置`CreateThread`通知，我们来看ARK工具中的枚举效果。



- 通常情况下:
    - PsSetCreateProcessNotifyRoutineEx 用于监控进程
    - PsSetCreateThreadNotifyRoutine 用于监控线程

监控进程的启动与退出可以使用`PsSetCreateProcessNotifyRoutineEx`来创建回调，当新进程创建时会优先执行回调，我们看下微软是如何定义的结构。

```
// 参数1: 新进程回调函数
// 参数2: 是否注销
NTSTATUS PsSetCreateProcessNotifyRoutineEx(
  [in] PCREATE_PROCESS_NOTIFY_ROUTINE_EX NotifyRoutine,
  [in] BOOLEAN                           Remove
);
```

如上，该函数只有两个参数，第一个参数是回调函数，第二个参数是是否注销，通常在驱动退出时可以传入`TRUE`对该回调进行注销，通常情况下如果驱动关闭，则必须要注销回调，而对于`MyLySharkCreateProcessNotifyEx`自定义回调来说，则需要指定三个必须要有的参数传递。

```
// 参数1: 新进程的EProcess
// 参数2: 新进程PID
// 参数3: 新进程详细信息 (仅在创建进程时有效)

VOID MyLySharkCreateProcessNotifyEx(PEPROCESS Process, HANDLE ProcessId,
PPS_CREATE_NOTIFY_INFO CreateInfo)
```

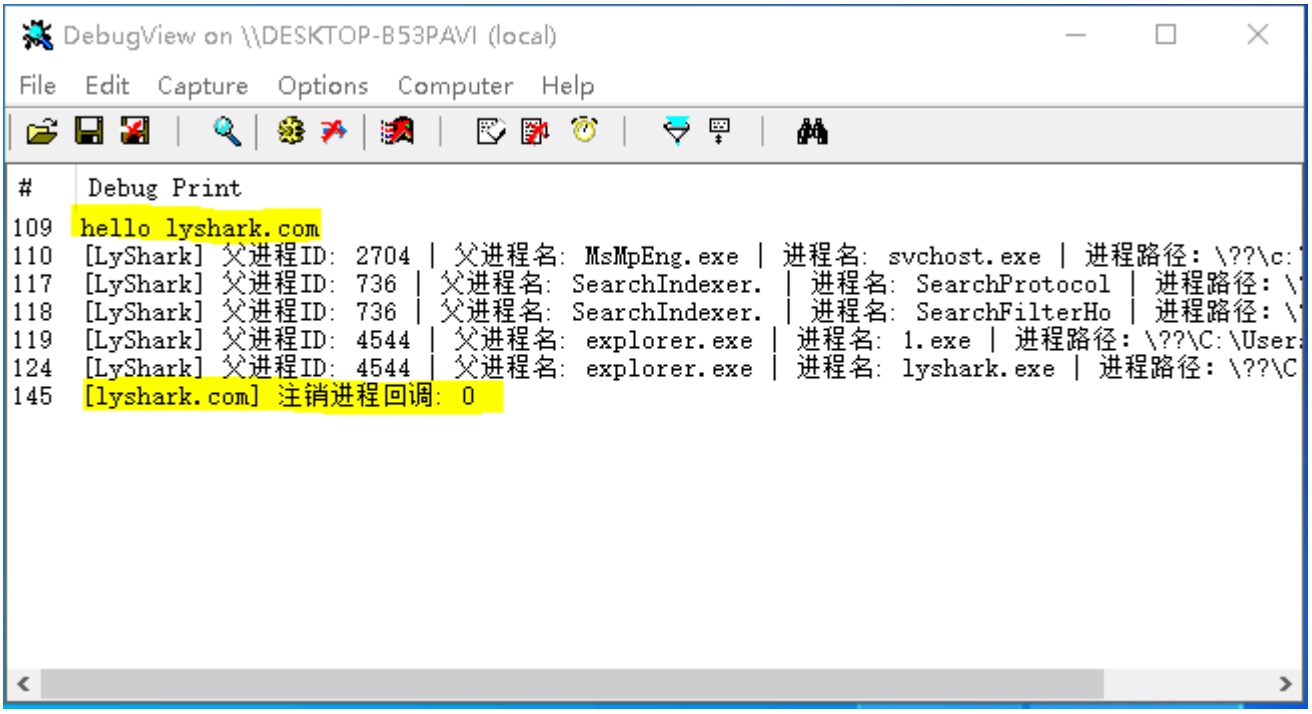
根据如上函数定义，就可以实现监控功能了，例如我们监控如果进程名是`lyshark.exe`则直接`CreateInfo->CreationStatus = STATUS_UNSUCCESSFUL`禁止该进程打开。

```
#include <ntifs.h>

// 两个未公开函数导出
NTKERNELAPI PCHAR PsGetProcessImageFileName(PEPROCESS Process);
NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS *Process);

// 通过PID获得进程名
PCHAR GetProcessNameByProcessId(HANDLE ProcessId)
{
    NTSTATUS st = STATUS_UNSUCCESSFUL;
    PEPROCESS ProcessObj = NULL;
    PCHAR string = NULL;
    st = PsLookupProcessByProcessId(ProcessId, &ProcessObj);
    if (NT_SUCCESS(st))
    {
        string = PsGetProcessImageFileName(ProcessObj);
        ObfDereferenceObject(ProcessObj);
    }
    return string;
}

// 绕过签名检查
BOOLEAN BypassCheckSign(PDRIVER_OBJECT pDriverObject)
{
#ifdef _WIN64
    typedef struct _KLDR_DATA_TABLE_ENTRY
    {
        LIST_ENTRY listEntry;
        ULONG64 __Undefined1;
        ULONG64 __Undefined2;
        ULONG64 __Undefined3;
        ULONG64 NonPagedDebugInfo;
        ULONG64 DllBase;
        ULONG64 EntryPoint;
        ULONG SizeOfImage;
        UNICODE_STRING path;
        UNICODE_STRING name;
        ULONG   Flags;
```

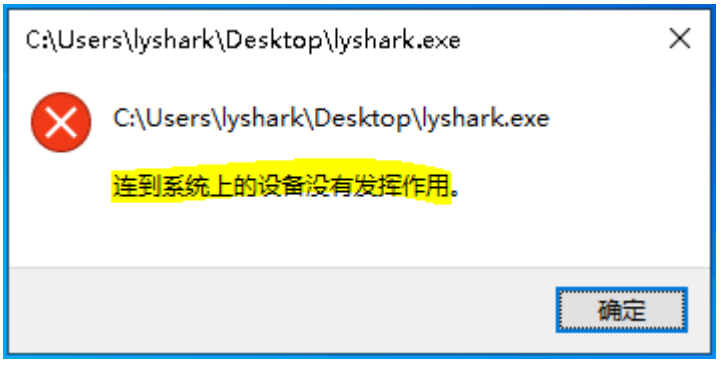
```
        USHORT  LoadCount;
        USHORT  __Undefined5;
        ULONG64 __Undefined6;
        ULONG   CheckSum;
        ULONG   __padding1;
        ULONG   TimeDateStamp;
        ULONG   __padding2;
    } KLDR_DATA_TABLE_ENTRY, *PKLDR_DATA_TABLE_ENTRY;
#else
    typedef struct _KLDR_DATA_TABLE_ENTRY
    {
        LIST_ENTRY listEntry;
        ULONG unknown1;
        ULONG unknown2;
        ULONG unknown3;
        ULONG unknown4;
        ULONG unknown5;
        ULONG unknown6;
        ULONG unknown7;
        UNICODE_STRING path;
        UNICODE_STRING name;
        ULONG   Flags;
    } KLDR_DATA_TABLE_ENTRY, *PKLDR_DATA_TABLE_ENTRY;
#endif

    PKLDR_DATA_TABLE_ENTRY pLdrData = (PKLDR_DATA_TABLE_ENTRY)pDriverObject->DriverSection;
    pLdrData->Flags = pLdrData->Flags | 0x20;

    return TRUE;
}

// 进程回调函数
VOID My_LyShark_Com_CreateProcessNotifyEx(PEPROCESS Process, HANDLE ProcessId,
PPS_CREATE_NOTIFY_INFO CreateInfo)
{
    char ProcName[16] = { 0 };
    if (CreateInfo != NULL)
    {
        strcpy_s(ProcName, 16, PsGetProcessImageFileName(Process));
        DbgPrint("[LyShark] 父进程ID: %ld | 父进程名: %s | 进程名: %s | 进程路径: %wZ \n",
CreateInfo->ParentProcessId, GetProcessNameByProcessId(CreateInfo->ParentProcessId),
PsGetProcessImageFileName(Process), CreateInfo->ImageFileName);

        // 判断是否为指定进程
        if (0 == _stricmp(ProcName, "lyshark.exe"))
        {
            // 禁止打开
            CreateInfo->CreationStatus = STATUS_UNSUCCESSFUL;
        }
    }
    else
    {
        strcpy_s(ProcName, 16, PsGetProcessImageFileName(Process));
```

```
        DbgPrint("[LyShark] 进程[ %s ] 退出了, 程序被关闭", ProcName);
    }
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DWORD32 ref = 0;

    // 注销进程回调
    ref =
PsSetCreateProcessNotifyRoutineEx((PCREATE_PROCESS_NOTIFY_ROUTINE_EX)My_LyShark_Com_CreatePr
ocessNotifyEx, TRUE);
    DbgPrint("[lyshark.com] 注销进程回调: %d \n", ref);
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    NTSTATUS status;

    // 绕过签名检查
    // LINKER_FLAGS=/INTEGRITYCHECK
    BypassCheckSign(Driver);

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

    // 创建进程回调
    // 参数1: 新进程的EProcess
    // 参数2: 新进程PID
    // 参数3: 新进程详细信息 (仅在创建进程时有效)
    status =
PsSetCreateProcessNotifyRoutineEx((PCREATE_PROCESS_NOTIFY_ROUTINE_EX)My_LyShark_Com_CreatePr
ocessNotifyEx, FALSE);
    if (!NT_SUCCESS(status))
    {
        DbgPrint("[lyshark.com] 创建进程回调错误");
    }
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

编译并运行这个驱动程序，我们可以在`ARK`工具中看到这个驱动所加载的`CreateProcess`的回调事件。



当驱动加载后，如果你尝试打开 `lyshark.exe` 那么会提示连接的设备没有发挥作用，我们则成功拦截了这次打开，当然如果在打开进程之前扫描其特征并根据特征拒绝进程打开，那么就可以实现一个简单的防恶意程序，进程监控在防恶意程序中也是用的最多的。



说完了 `PsSetCreateProcessNotifyRoutineEx` 回调的使用方式，LyShark将继续带大家看看 `线程监控` 如何实现，监控线程创建与监控进程差不多，检测线程需要调用 `PsSetCreateThreadNotifyRoutine` 创建回调函数，之后就可监控系统所有线程的创建，具体实现代码如下。

```
#include <ntifs.h>

// 两个未公开函数导出
NTKERNELAPI PCHAR PsGetProcessImageFileName(PEPROCESS Process);
NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS *Process);
NTKERNELAPI NTSTATUS PsLookupThreadByThreadId(HANDLE ThreadId, PETHREAD *Thread);

// 绕过签名检查
BOOLEAN BypassCheckSign(PDRIVER_OBJECT pDriverObject)
{
#ifdef _WIN64
    typedef struct _KLDR_DATA_TABLE_ENTRY
    {
        LIST_ENTRY listEntry;
        ULONG64 __Undefined1;
```

```
        ULONG64 __Undefined2;
        ULONG64 __Undefined3;
        ULONG64 NonPagedDebugInfo;
        ULONG64 DllBase;
        ULONG64 EntryPoint;
        ULONG SizeOfImage;
        UNICODE_STRING path;
        UNICODE_STRING name;
        ULONG   Flags;
        USHORT  LoadCount;
        USHORT  __Undefined5;
        ULONG64 __Undefined6;
        ULONG   CheckSum;
        ULONG   __padding1;
        ULONG   TimeDateStamp;
        ULONG   __padding2;
    } KLDR_DATA_TABLE_ENTRY, *PKLDR_DATA_TABLE_ENTRY;
#else
    typedef struct _KLDR_DATA_TABLE_ENTRY
    {
        LIST_ENTRY listEntry;
        ULONG unknown1;
        ULONG unknown2;
        ULONG unknown3;
        ULONG unknown4;
        ULONG unknown5;
        ULONG unknown6;
        ULONG unknown7;
        UNICODE_STRING path;
        UNICODE_STRING name;
        ULONG   Flags;
    } KLDR_DATA_TABLE_ENTRY, *PKLDR_DATA_TABLE_ENTRY;
#endif

    PKLDR_DATA_TABLE_ENTRY pLdrData = (PKLDR_DATA_TABLE_ENTRY)pDriverObject->DriverSection;
    pLdrData->Flags = pLdrData->Flags | 0x20;

    return TRUE;
}

// 线程回调函数
VOID MyCreateThreadNotify(HANDLE ProcessId, HANDLE ThreadId, BOOLEAN CreateInfo)
{
    PEPROCESS eprocess = NULL;
    PETHREAD ethread = NULL;
    UCHAR *pWin32Address = NULL;

    // 通过此函数拿到程序的EPROCESS结构
    PsLookupProcessByProcessId(ProcessId, &eprocess);
    PsLookupThreadByThreadId(ThreadId, &ethread);

    if (CreateInfo)
    {
```

```
        DbgPrint("[lyshark.com] 线程TID: %1d | 所属进程名: %s | 进程PID: %1d \n", ThreadId,
PsGetProcessImageFileName(eprocess), PsGetProcessId(eprocess));
        /*
        if (0 == _stricmp(PsGetProcessImageFileName(eprocess), "lyshark.exe"))
        {
        DbgPrint("线程TID: %1d | 所属进程名: %s | 进程PID: %1d \n", ThreadId,
PsGetProcessImageFileName(eprocess), PsGetProcessId(eprocess));

        // dt _kthread
        // 寻找里面的 Win32StartAddress 并写入ret
        pWin32Address = *(UCHAR**)((UCHAR*)ethread + 0x1c8);
        if (MmIsAddressValid(pWin32Address))
        {
        *pWin32Address = 0xC3;
        }
        }
        */
    }
    else
    {
        DbgPrint("[LyShark] %s 线程已退出...", ThreadId);
    }

    if (eprocess)
        ObDereferenceObject(eprocess);
    if (ethread)
        ObDereferenceObject(ethread);
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    NTSTATUS status;

    // 注销进程回调
    status = PsRemoveCreateThreadNotifyRoutine(MyCreateThreadNotify);
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    NTSTATUS status;

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

    // 绕过签名检查
    // LINKER_FLAGS=/INTEGRITYCHECK
    BypassCheckSign(Driver);

    // 创建线程回调
    // 参数1: 新线程ProcessID
    // 参数2: 新线程ThreadID
    // 参数3: 线程创建/退出标志
    status = PsSetCreateThreadNotifyRoutine(MyCreateThreadNotify);
    if (!NT_SUCCESS(status))
```

```
    {
        DbgPrint("创建线程回调错误");
    }

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

运行后则可监控到系统总所有线程的创建与退出，效果如下所示：