在笔者上一篇文章《内核枚举进程与线程ObCall回调》简单介绍了如何枚举系统中已经存在的进程与线程回调，本章LyShark将通过对象回调实现对进程线程的句柄监控，在内核中提供了ObRegisterCallbacks回调，使用这个内核回调函数，可注册一个对象回调，不过目前该函数只能监控进程与线程句柄操作，通过监控进程或线程句柄，可实现保护指定进程线程不被终止的目的。

ObRegisterCallbacks是Windows操作系统提供的一个内核API函数，它允许开发者注册一个回调函数，用于监控对象的创建、打开、关闭和删除等事件。对象可以是文件、目录、进程、线程、注册表键等等。

当操作系统创建、打开、关闭或删除一个对象时，它会触发注册的回调函数，然后在回调函数中调用开发者定义的代码。开发者可以在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。

ObRegisterCallbacks函数提供了多个回调函数的注册，这些回调函数包括：

- PreOperation: 在操作执行之前被调用，可以阻止或修改操作。
- PostOperation: 在操作执行之后被调用，可以记录操作结果或者清理资源。

需要注意的是，注册回调函数需要开发者有一定的内核开发经验，并且需要遵守一些约束条件，例如不能阻塞或挂起对象的操作，不能调用一些内核API函数等。

内核注册并监控对象回调ObRegisterCallbacks在安全软件、系统监控和调试工具等领域有着广泛的应用。开发者可以利用这个机制来监控系统对象的使用情况，以保护系统安全。

由于目前对象回调只能监控进程与线程，而这个监控是通过ObjectType这么一个成员控制的，如果成员是PsProcessType则代表监控进程，反之PsThreadType则是监控线程，无论监控进程还是线程都调用ObRegisterCallbacks这个函数来完成注册。

函数ObRegisterCallbacks其微软对他的定义是这样的，用户传入OB_OPERATION_REGISTRATION结构，以及OB_CALLBACK_REGISTRATION回调结构，其中PreOperation则是传入的回调函数，也是最重要的，其次是ObjectType指定成进程回调。

```
NTSTATUS ObRegisterCallbacks(
  [in]  POB_CALLBACK_REGISTRATION CallbackRegistration,
  [out] PVOID                     *RegistrationHandle
);
```

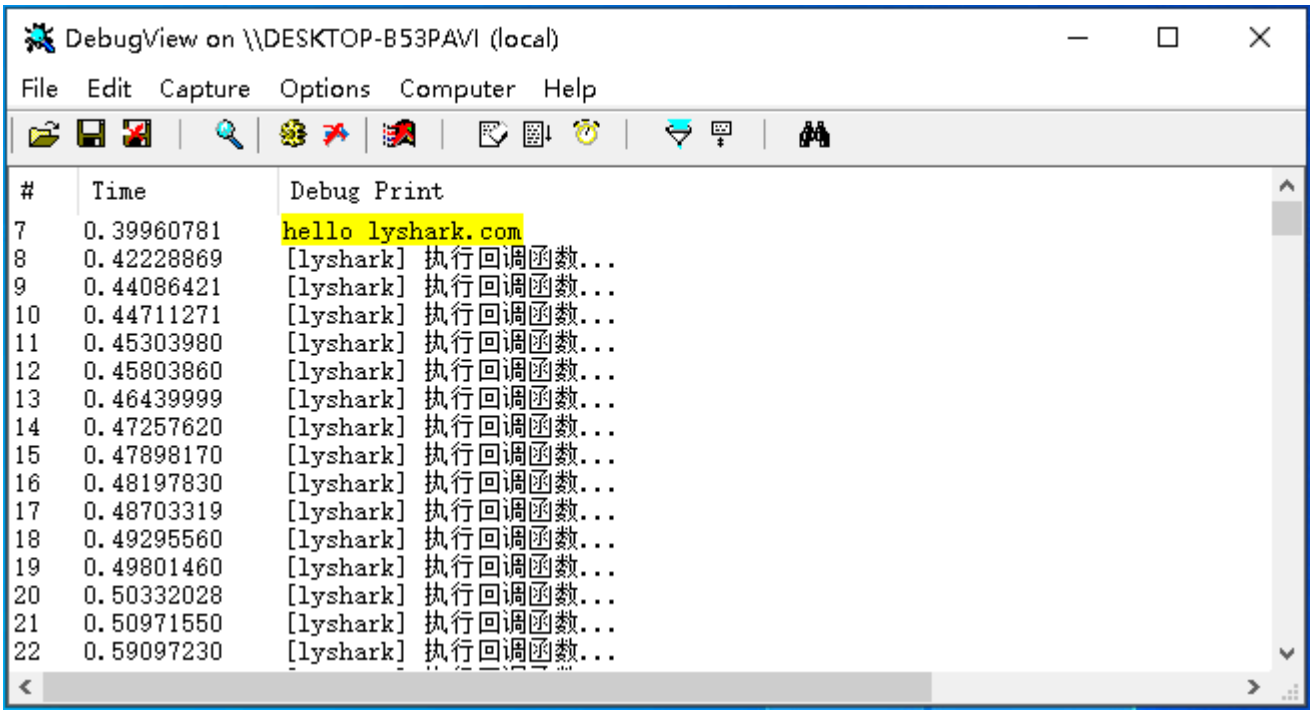
首先来实现一个检测的案例，注册一个进程回调对象MyLySharkComObjectCallBack，通过ObRegisterCallbacks注册的回调只需要传入一个填充好的OB_CALLBACK_REGISTRATION回调结构体，以及一个全局句柄即可，这个全局句柄的作用仅仅只是在程序结束时，调用ObUnRegisterCallbacks卸载监控而已，实现代码如下所示。

```
#include <ntddk.h>
#include <ntstrsafe.h>

PVOID Globle_Object_Handle;

// 绕过签名检测
void BypassCheckSign(PDRIVER_OBJECT pDriverObj)
{
    typedef struct _LDR_DATA
    {
        struct _LIST_ENTRY InLoadOrderLinks;
        struct _LIST_ENTRY InMemoryOrderLinks;
        struct _LIST_ENTRY InInitializationOrderLinks;
```

```
        VOID*        DllBase;
        VOID*        EntryPoint;
        ULONG32      SizeOfImage;
        UINT8        _PADDING0_[0x4];
        struct _UNICODE_STRING FullDllName;
        struct _UNICODE_STRING BaseDllName;
        ULONG32      Flags;
    }LDR_DATA, *PLDR_DATA;

    PLDR_DATA ldr;
    ldr = (PLDR_DATA)(pDriverObj->DriverSection);
    ldr->Flags |= 0x20;
}

// 自定义回调
OB_PREOP_CALLBACK_STATUS MyLySharkComObjectCallBack(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION OperationInformation)
{
    DbgPrint("[lyshark] 执行回调函数... \n");
    return STATUS_SUCCESS;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    ObUnRegisterCallbacks(Globle_Object_Handle);
    DbgPrint("回调卸载完成... \n");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");

    BypassCheckSign(Driver);

    OB_OPERATION_REGISTRATION Base;                          // 回调函数结构体
    OB_CALLBACK_REGISTRATION CallbackReg;                    // 回调函数

    CallbackReg.RegistrationContext = NULL;                  // 注册上下文(你回调函数返回参数)
    CallbackReg.Version = OB_FLT_REGISTRATION_VERSION;       // 注册回调版本
    CallbackReg.OperationRegistration = &Base;               // 回调结构体
    CallbackReg.OperationRegistrationCount = 1;              // 操作计数(下钩数量)

    RtlUnicodeStringInit(&CallbackReg.Altitude, L"600000");  // 长度
    Base.ObjectType = PsProcessType;                         // 进程操作类型.此处为进程操作
    Base.Operations = OB_OPERATION_HANDLE_CREATE;            // 操作句柄创建
    Base.PreOperation = MyLySharkComObjectCallBack;          // 你自己的回调函数
    Base.PostOperation = NULL;

    // 注册回调
    if (ObRegisterCallbacks(&CallbackReg, &Globle_Object_Handle))
    {
        DbgPrint("[lyshark message] 回调注册成功...");
    }
```

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

当驱动程序被加载以后，一旦有进程运行则会执行我们自己的`MyLySharkComObjectCallBack`回调，而在回调函数内则可以执行任意功能，运行如下所示。



如上所示只是演示基本的回调申请流程，回调函数通常需要包含两个值，其一`RegistrationContext`用于标注上下文，其二`POB_PRE_OPERATION_INFORMATION`则用于标注进程或者线程创建的信息结构体。

```
OB_PREOP_CALLBACK_STATUS MyLySharkComObjectCallBack(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION OperationInformation)
```

那么如何实现`拦截进程启动`这个功能呢，我们可以在回调函数中写入以下代码进行拦截。

- CreateHandleInformation.DesiredAccess 将打开句柄的权限清零
- CreateHandleInformation.OriginalDesiredAccess 判断是否终止

```
if (pOperationInformation->Operation == OB_OPERATION_HANDLE_CREATE)
{
    DbgPrint("lyshark.exe 进程打开 \n");
    pOperationInformation->Parameters->CreateHandleInformation.DesiredAccess=0;
    if ((pOperationInformation->Parameters->CreateHandleInformation.OriginalDesiredAccess &
PROCESS_TERMINATE) == PROCESS_TERMINATE)
    {
        pOperationInformation->Parameters->CreateHandleInformation.DesiredAccess &=
~PROCESS_TERMINATE;
    }
}
```

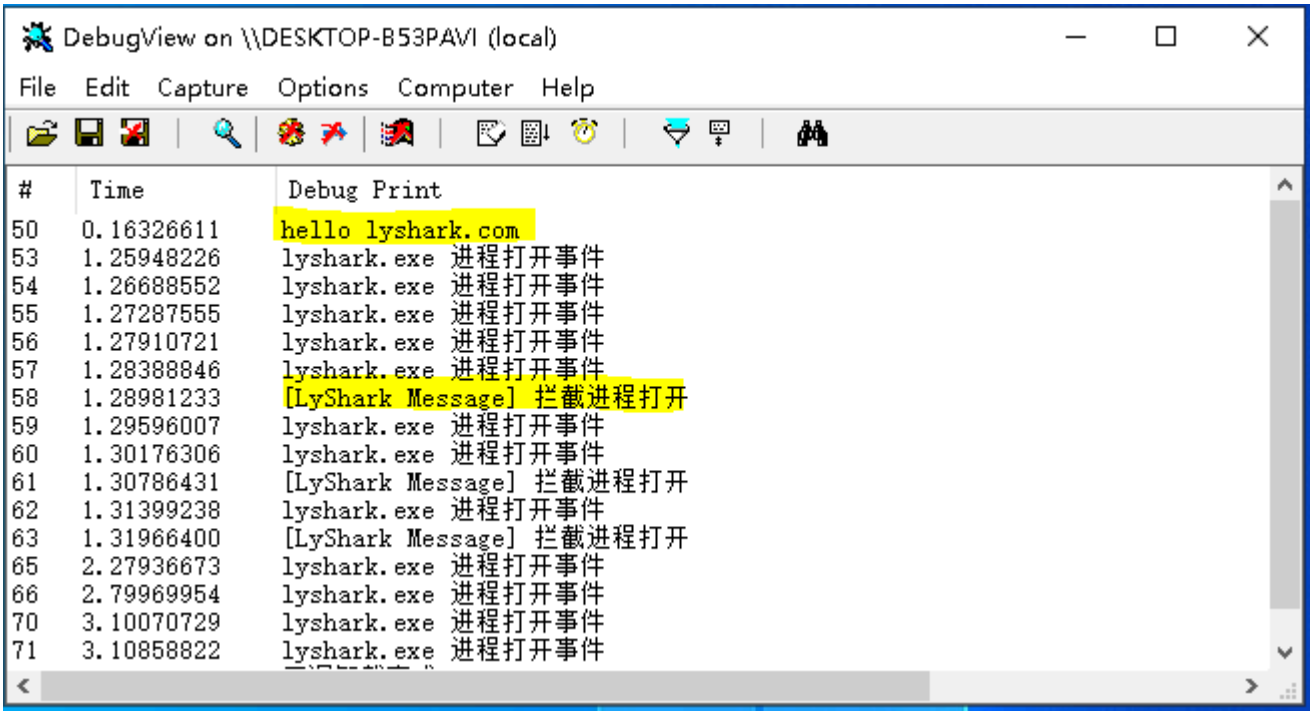
拦截进程创建核心代码如下所示。

```
#include <ntddk.h>
#include <ntstrsafe.h>

#define PROCESS_TERMINATE 0x1

// 导出两个API
NTKERNELAPI PEPROCESS IoThreadToProcess(PETHREAD Thread);
NTKERNELAPI char* PsGetProcessImageFileName(PEPROCESS Process);

// 全局句柄
PVOID Globle_Object_Handle = NULL;

// 绕过签名检测
void BypassCheckSign(PDRIVER_OBJECT pDriverObj)
{
    typedef struct _LDR_DATA
    {
        struct _LIST_ENTRY InLoadOrderLinks;
        struct _LIST_ENTRY InMemoryOrderLinks;
        struct _LIST_ENTRY InInitializationOrderLinks;
        VOID*        DllBase;
        VOID*        EntryPoint;
        ULONG32      SizeOfImage;
        UINT8        _PADDING0_[0x4];
        struct _UNICODE_STRING FullDllName;
        struct _UNICODE_STRING BaseDllName;
        ULONG32      Flags;
    }LDR_DATA, *PLDR_DATA;

    PLDR_DATA ldr;
    ldr = (PLDR_DATA)(pDriverObj->DriverSection);
    ldr->Flags |= 0x20;
}

// 判断是否是需要保护的进程
BOOLEAN CheckProcess(PEPROCESS eprocess)
{
    char *Name = PsGetProcessImageFileName(eprocess);
    if (!_stricmp("lyshark.exe", Name))
        return TRUE;
    else
        return FALSE;
}

// 进程回调
OB_PREOP_CALLBACK_STATUS MyLySharkProcessObjectCallBack(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION pOperationInformation)
{
    HANDLE pid;

    // 只取出进程回调
    if (pOperationInformation->ObjectType != *PsProcessType)
    {
```

```
        return OB_PREOP_SUCCESS;
    }

    // 得到所有进程的ID
    pid = PsGetProcessId((PEPROCESS)pOperationInformation->Object);
    // DbgPrint("进程PID= %ld \n", pid);

    UNREFERENCED_PARAMETER(RegistrationContext);

    // 验证是否是需要的进程
    if (CheckProcess((PEPROCESS)pOperationInformation->Object))
    {
        // 创建句柄
        if (pOperationInformation->Operation == OB_OPERATION_HANDLE_CREATE)
        {
            DbgPrint("lyshark.exe 进程打开事件 \n");
            pOperationInformation->Parameters->CreateHandleInformation.DesiredAccess=0;
            if ((pOperationInformation->Parameters->CreateHandleInformation.OriginalDesiredAccess & PROCESS_TERMINATE) == PROCESS_TERMINATE)
            {
                DbgPrint("[LyShark Message] 拦截进程打开 \n");
                pOperationInformation->Parameters->CreateHandleInformation.DesiredAccess &= ~PROCESS_TERMINATE;
            }
        }
        // 复制句柄
        if (pOperationInformation->Operation == OB_OPERATION_HANDLE_DUPLICATE)
        {
            DbgPrint("lyshark.exe 进程被关闭 \n");
            pOperationInformation->Parameters->DuplicateHandleInformation.DesiredAccess=0;
            if ((pOperationInformation->Parameters->DuplicateHandleInformation.OriginalDesiredAccess & PROCESS_TERMINATE) == PROCESS_TERMINATE)
            {
                pOperationInformation->Parameters->DuplicateHandleInformation.DesiredAccess &= ~PROCESS_TERMINATE;
            }
        }
    }
    return OB_PREOP_SUCCESS;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    ObUnRegisterCallbacks(Globle_Object_Handle);
    DbgPrint("回调卸载完成... \n");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");

    BypassCheckSign(Driver);
```

```
    OB_OPERATION_REGISTRATION ob_process_callback;
    OB_CALLBACK_REGISTRATION op_process_operation;

    memset(&ob_process_callback, 0, sizeof(ob_process_callback));
    ob_process_callback.ObjectType = PsProcessType;
    ob_process_callback.Operations = OB_OPERATION_HANDLE_CREATE |
OB_OPERATION_HANDLE_DUPLICATE;
    ob_process_callback.PreOperation = MyLySharkProcessObjectCallBack;
    ob_process_callback.PostOperation = NULL;

    RtlUnicodeStringInit(&op_process_operation.Altitude, L"600000");
    op_process_operation.RegistrationContext = NULL;
    op_process_operation.Version = OB_FLT_REGISTRATION_VERSION;
    op_process_operation.OperationRegistration = &ob_process_callback;
    op_process_operation.OperationRegistrationCount = 1;

    // 注册进程回调
    if (ObRegisterCallbacks(&op_process_operation, &Globle_Object_Handle))
    {
        DbgPrint("进程回调注册成功...");
    }

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

加载这个驱动，当有进程被创建时，则首先判断是否是`lyshark.exe`如果是则直接禁止打开，也就是终止掉。



同理进程可以被拦截，那么如果增加更多的过滤条件，则`线程`同样可以被拦截，拦截线程代码如下所示。

```
#include <ntddk.h>
#include <ntstrsafe.h>
```

```
#define THREAD_TERMINATE2 0x1

// 导出两个API
NTKERNELAPI PEPROCESS IoThreadToProcess(PETHREAD Thread);
NTKERNELAPI char* PsGetProcessImageFileName(PEPROCESS Process);

// 全局句柄
PVOID Globle_Object_Handle = NULL;

// 绕过签名检测
void BypassCheckSign(PDRIVER_OBJECT pDriverObj)
{
    typedef struct _LDR_DATA
    {
        struct _LIST_ENTRY InLoadOrderLinks;
        struct _LIST_ENTRY InMemoryOrderLinks;
        struct _LIST_ENTRY InInitializationOrderLinks;
        VOID*        DllBase;
        VOID*        EntryPoint;
        ULONG32      SizeOfImage;
        UINT8        _PADDING0_[0x4];
        struct _UNICODE_STRING FullDllName;
        struct _UNICODE_STRING BaseDllName;
        ULONG32      Flags;
    }LDR_DATA, *PLDR_DATA;

    PLDR_DATA ldr;
    ldr = (PLDR_DATA)(pDriverObj->DriverSection);
    ldr->Flags |= 0x20;
}

// 判断是否是需要保护的进程
BOOLEAN CheckProcess(PEPROCESS eprocess)
{
    char *Name = PsGetProcessImageFileName(eprocess);
    if (!_stricmp("lyshark.exe", Name))
        return TRUE;
    else
        return FALSE;
}

// 线程回调
OB_PREOP_CALLBACK_STATUS MyThreadObjectCallBack(PVOID RegistrationContext,
POB_PRE_OPERATION_INFORMATION pOperationInformation)
{
    PEPROCESS ep;
    PETHREAD et;
    HANDLE pid;

    // 线程过滤
    if (pOperationInformation->ObjectType != *PsThreadType)
    {
        return OB_PREOP_SUCCESS;
```

```
    }

    et = (PETHREAD)pOperationInformation->Object;
    ep = IoThreadToProcess(et);
    pid = PsGetProcessId(ep);

    // DbgPrint("线程PID= %ld | TID= %ld \n", pid, PsGetThreadId(et));
    UNREFERENCED_PARAMETER(RegistrationContext);

    if (CheckProcess(ep))
    {
        if (pOperationInformation->Operation == OB_OPERATION_HANDLE_CREATE)
        {
            pOperationInformation->Parameters->CreateHandleInformation.DesiredAccess=0;
            if ((pOperationInformation->Parameters-
>CreateHandleInformation.OriginalDesiredAccess & THREAD_TERMINATE2) == THREAD_TERMINATE2)
            {
                DbgPrint("[LyShark] 拦截lyshark.exe进程内 %d 线程创建 \n", PsGetThreadId(et));
                pOperationInformation->Parameters->CreateHandleInformation.DesiredAccess &=
~THREAD_TERMINATE2;
            }
        }
        if (pOperationInformation->Operation == OB_OPERATION_HANDLE_DUPLICATE)
        {
            pOperationInformation->Parameters->DuplicateHandleInformation.DesiredAccess=0;
            if ((pOperationInformation->Parameters-
>DuplicateHandleInformation.OriginalDesiredAccess & THREAD_TERMINATE2) == THREAD_TERMINATE2)
            {
                pOperationInformation->Parameters->DuplicateHandleInformation.DesiredAccess
&= ~THREAD_TERMINATE2;
            }
        }
    }
    return OB_PREOP_SUCCESS;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    ObUnRegisterCallbacks(Globle_Object_Handle);
    DbgPrint("回调卸载完成... \n");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");

    BypassCheckSign(Driver);

    OB_OPERATION_REGISTRATION ob_thread_callback;
    OB_CALLBACK_REGISTRATION op_thread_operation;

    memset(&ob_thread_callback, 0, sizeof(ob_thread_callback));
    ob_thread_callback.ObjectType = PsThreadType;
```

```
    ob_thread_callback.Operations = OB_OPERATION_HANDLE_CREATE |
OB_OPERATION_HANDLE_DUPLICATE;
    ob_thread_callback.PreOperation = MyThreadObjectCallBack;
    ob_thread_callback.PostOperation = NULL;

    RtlUnicodeStringInit(&op_thread_operation.Altitude, L"600001");
    op_thread_operation.RegistrationContext = NULL;
    op_thread_operation.Version = OB_FLT_REGISTRATION_VERSION;
    op_thread_operation.OperationRegistration = &ob_thread_callback;
    op_thread_operation.OperationRegistrationCount = 1;

    // 注册进程回调
    if (ObRegisterCallbacks(&op_thread_operation, &Globle_Object_Handle))
    {
        DbgPrint("进程回调注册成功...");
    }

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

这段驱动加载后，如果有新线程被创建，则会被拦截并打印输出，效果图如下。