在前面的文章`《内核解析PE结构导出表》`中我们封装了两个函数`KernelMapFile()`函数可用来读取内核文件，`GetAddressFromFunction()`函数可用来在导出表中寻找指定函数的导出地址，本章将以此为基础实现对特定`SSDT`函数的`Hook`挂钩操作，与`《内核层InlineHook挂钩函数》`所使用的挂钩技术基本一致，不同点是前者使用了`CR3`的方式改写内存，而今天所讲的是通过`MDL映射`实现，此外前者挂钩中所取到的地址是通过`GetProcessAddress()`取到的动态地址，而今天所使用的方式是通过读取导出表寻找。

挂钩的目的就是要为特定函数增加功能，挂钩的实现方式无非就是替换原函数地址，我们以内核函数`ZwQueryDirectoryFile()`为例，`ZwQueryDirectoryFile`例程返回给定文件句柄指定的目录中文件的各种信息，其微软定义如下；

```
NTSYSAPI NTSTATUS ZwQueryDirectoryFile(
  [in]           HANDLE                 FileHandle,
  [in, optional] HANDLE                 Event,
  [in, optional] PIO_APC_ROUTINE        ApcRoutine,
  [in, optional] PVOID                  ApcContext,
  [out]          PIO_STATUS_BLOCK       IoStatusBlock,
  [out]          PVOID                  FileInformation,
  [in]           ULONG                  Length,
  [in]           FILE_INFORMATION_CLASS FileInformationClass,
  [in]           BOOLEAN                ReturnSingleEntry,
  [in, optional] PUNICODE_STRING        FileName,
  [in]           BOOLEAN                RestartScan
);
```

如果需要`Hook`一个函数则你需要去微软官方得到该函数的具体声明部分包括其返回值，而`Hook`的目的只是为函数增加或处理新功能，则在执行完自定义函数后一定要跳回到原始函数上，此时定义一个`typedef_ZwQueryDirectoryFile`函数指针在调用结束后即可很容易的跳转回原函数上，保证流程被正确执行，如果需要Hook其他函数其编写模板也是如下所示；

```
// 保存原函数地址
PVOID gOldFunctionAddress = NULL;

// Hook后被替换的新函数
NTSTATUS MyZwQueryDirectoryFile(
    IN HANDLE                 FileHandle,
    IN HANDLE                 Event OPTIONAL,
    IN PIO_APC_ROUTINE        ApcRoutine OPTIONAL,
    IN PVOID                  ApcContext OPTIONAL,
    OUT PIO_STATUS_BLOCK      IoStatusBlock,
    OUT PVOID                 FileInformation,
    IN ULONG                  Length,
    IN FILE_INFORMATION_CLASS FileInformationClass,
    IN BOOLEAN                ReturnSingleEntry,
    IN PUNICODE_STRING        FileMask OPTIONAL,
    IN BOOLEAN                RestartScan
    )
{
    NTSTATUS status = STATUS_SUCCESS;

    // 定义函数指针
    typedef NTSTATUS(*typedef_ZwQueryDirectoryFile)(
        IN HANDLE               FileHandle,
        IN HANDLE               Event OPTIONAL,
        IN PIO_APC_ROUTINE      ApcRoutine OPTIONAL,
        IN PVOID                ApcContext OPTIONAL,
        OUT PIO_STATUS_BLOCK    IoStatusBlock,
        OUT PVOID               FileInformation,
        IN ULONG                Length,
        IN FILE_INFORMATION_CLASS FileInformationClass,
        IN BOOLEAN              ReturnSingleEntry,
        IN PUNICODE_STRING      FileMask OPTIONAL,
        IN BOOLEAN              RestartScan
        );

    DbgPrint("MyZwQueryDirectoryFile 自定义功能 \n");

    // 执行原函数
    status = ((typedef_ZwQueryDirectoryFile)gOldFunctionAddress)(FileHandle,
        Event,
        ApcRoutine,
        ApcContext,
        IoStatusBlock,
        FileInformation,
        Length,
        FileInformationClass,
        ReturnSingleEntry,
        FileMask,
        RestartScan);

    return status;
}
```

接着就是如何挂钩并让其中转到我们自己的代码流程中的问题，由于挂钩与恢复代码是一样的此处就以挂钩为例，首先调用`MmCreateMdl()`创建MDL，接着调用`MmBuildMdlForNonPagedPool()`接收一个 MDL，该MDL指定非分页虚拟内存缓冲区，并对其进行更新以描述基础物理页。调用`MmMapLockedPages()`将此段内存提交为锁定状态，最后就是调用`RtlCopyMemory()`将新函数地址写出到内存中实现替换，最后释放MDL句柄即可，这段代码如下所示，看过`驱动读写篇`的你一定很容易就能理解。

```
// 挂钩SSDT函数
BOOLEAN SSDTFunctionHook(ULONG64 FunctionAddress)
{
    PMDL pMdl = NULL;
    PVOID pNewAddress = NULL;
    ULONG ulNewFuncAddr = 0;

    gOldFunctionAddress = FunctionAddress;

    // 使用MDL修改SSDT
    pMdl = MmCreateMdl(NULL, &FunctionAddress, sizeof(ULONG));
    if (NULL == pMdl)
    {
        return FALSE;
    }
    MmBuildMdlForNonPagedPool(pMdl);

    // 锁定内存
    pNewAddress = MmMapLockedPages(pMdl, KernelMode);
    if (NULL == pNewAddress)
    {
        IoFreeMdl(pMdl);
        return FALSE;
    }

    // 写入新函数地址
    ulNewFuncAddr = (ULONG)MyZwQueryDirectoryFile;
    RtlCopyMemory(pNewAddress, &ulNewFuncAddr, sizeof(ULONG));

    // 释放
    MmUnmapLockedPages(pNewAddress, pMdl);
    IoFreeMdl(pMdl);

    return TRUE;
}
```

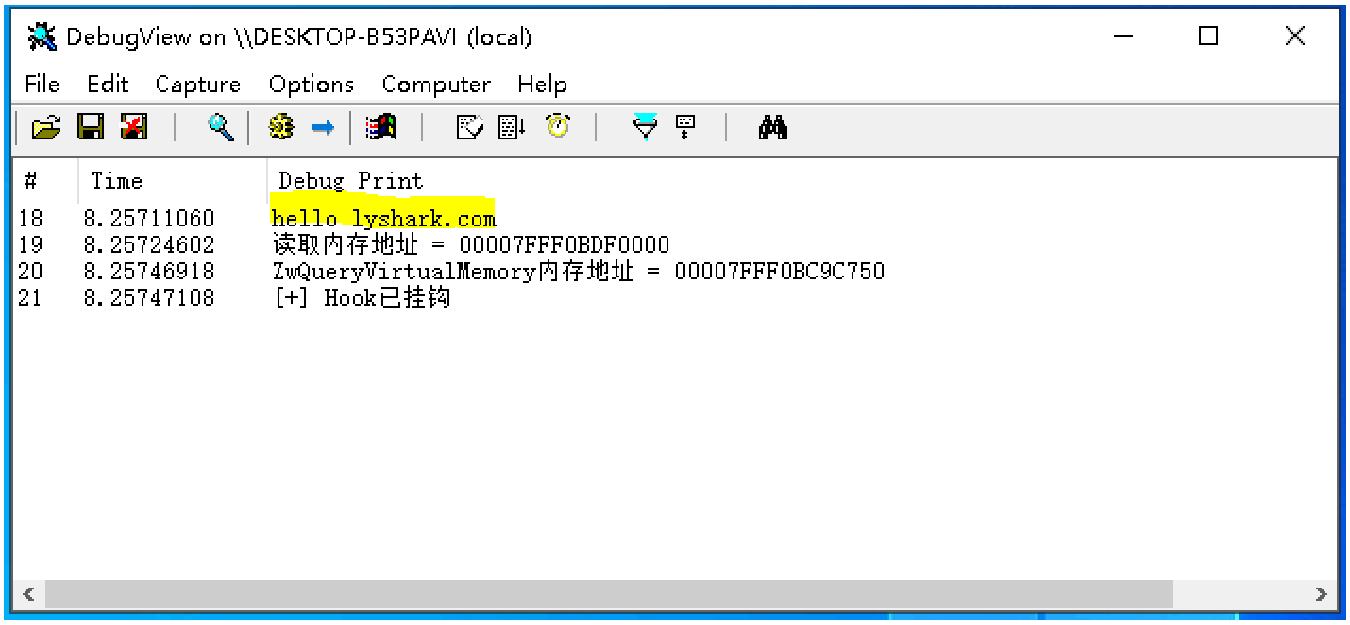
Hook核心代码如下所示，为了节约篇幅，如果您找不到程序中的核心功能，请看前面的几篇文章，这里就不在赘述了。

```
// 保存原函数地址
PVOID gOldFunctionAddress = NULL;

// Hook后被替换的新函数
NTSTATUS MyZwQueryDirectoryFile(
    IN HANDLE                 FileHandle,
    IN HANDLE                 Event OPTIONAL,
    IN PIO_APC_ROUTINE        ApcRoutine OPTIONAL,
    IN PVOID                  ApcContext OPTIONAL,
    OUT PIO_STATUS_BLOCK      IoStatusBlock,
    OUT PVOID                 FileInformation,
    IN ULONG                  Length,
    IN FILE_INFORMATION_CLASS FileInformationClass,
    IN BOOLEAN                ReturnSingleEntry,
    IN PUNICODE_STRING        FileMask OPTIONAL,
    IN BOOLEAN                RestartScan
    )
{
    NTSTATUS status = STATUS_SUCCESS;

    // 定义函数指针
    typedef NTSTATUS(*typedef_ZwQueryDirectoryFile)(
        IN HANDLE                 FileHandle,
        IN HANDLE                 Event OPTIONAL,
        IN PIO_APC_ROUTINE        ApcRoutine OPTIONAL,
        IN PVOID                  ApcContext OPTIONAL,
        OUT PIO_STATUS_BLOCK      IoStatusBlock,
        OUT PVOID                FileInformation,
        IN ULONG                 Length,
        IN FILE_INFORMATION_CLASS FileInformationClass,
        IN BOOLEAN               ReturnSingleEntry,
        IN PUNICODE_STRING       FileMask OPTIONAL,
        IN BOOLEAN               RestartScan
        );

    DbgPrint("MyZwQueryDirectoryFile 自定义功能 \n");

    // 执行原函数
    status = ((typedef_ZwQueryDirectoryFile)gOldFunctionAddress)(FileHandle,
        Event,
        ApcRoutine,
        ApcContext,
        IoStatusBlock,
        FileInformation,
        Length,
        FileInformationClass,
        ReturnSingleEntry,
        FileMask,
        RestartScan);

    return status;
}

// 挂钩SSDT函数
BOOLEAN SSDTFunctionHook(ULONG64 FunctionAddress)
{
    PMDL pMdl = NULL;
    PVOID pNewAddress = NULL;
    ULONG ulNewFuncAddr = 0;

    gOldFunctionAddress = FunctionAddress;

    // 使用MDL修改SSDT
    pMdl = MmCreateMdl(NULL, &FunctionAddress, sizeof(ULONG));
    if (NULL == pMdl)
    {
        return FALSE;
    }

    MmBuildMdlForNonPagedPool(pMdl);

    // 锁定内存
    pNewAddress = MmMapLockedPages(pMdl, KernelMode);
    if (NULL == pNewAddress)
    {
        IoFreeMdl(pMdl);
        return FALSE;
    }

    // 写入新函数地址
```

```c
    ulNewFuncAddr = (ULONG)MyZwQueryDirectoryFile;
    RtlCopyMemory(pNewAddress, &ulNewFuncAddr, sizeof(ULONG));

    // 释放
    MmUnmapLockedPages(pNewAddress, pMdl);
    IoFreeMdl(pMdl);

    return TRUE;
}

// 恢复SSDT函数
BOOLEAN SSDTFunctionUnHook(ULONG64 FunctionAddress)
{
    PMDL pMdl = NULL;
    PVOID pNewAddress = NULL;
    ULONG ulOldFuncAddr = 0;

    gOldFunctionAddress = FunctionAddress;

    // 使用MDL修改SSDT
    pMdl = MmCreateMdl(NULL, &FunctionAddress, sizeof(ULONG));
    if (NULL == pMdl)
    {
        return FALSE;
    }

    MmBuildMdlForNonPagedPool(pMdl);

    // 锁定内存
    pNewAddress = MmMapLockedPages(pMdl, KernelMode);
    if (NULL == pNewAddress)
    {
        IoFreeMdl(pMdl);
        return FALSE;
    }

    // 写入新函数地址
    ulOldFuncAddr = (ULONG)gOldFunctionAddress;
    RtlCopyMemory(pNewAddress, &ulOldFuncAddr, sizeof(ULONG));

    // 释放
    MmUnmapLockedPages(pNewAddress, pMdl);
    IoFreeMdl(pMdl);

    return TRUE;
}

// 关闭驱动
VOID UnDriver(PDRIVER_OBJECT driver)
{
    SSDTFunctionUnHook(gOldFunctionAddress);
    DbgPrint("驱动卸载 \n");
}
```

```c
// 驱动入口
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");

    NTSTATUS status = STATUS_SUCCESS;

    HANDLE hFile = NULL;
    HANDLE hSection = NULL;
    PVOID pBaseAddress = NULL;
    UNICODE_STRING FileName = { 0 };
    ULONG64 FunctionAddress = 0;

    // 初始化字符串
    RtlInitUnicodeString(&FileName, L"\\??\\C:\\Windows\\System32\\ntdll.dll");

    // 内存映射文件
    status = KernelMapFile(FileName, &hFile, &hSection, &pBaseAddress);
    if (NT_SUCCESS(status))
    {
        DbgPrint("读取内存地址 = %p \n", pBaseAddress);
    }

    // 获取指定模块导出函数地址
    FunctionAddress = GetAddressFromFunction(FileName, "ZwQueryDirectoryFile");
    DbgPrint("ZwQueryVirtualMemory内存地址 = %p \n", FunctionAddress);

    // 开始Hook挂钩
    if (FunctionAddress != 0)
    {
        BOOLEAN ref = SSDTFunctionHook(FunctionAddress);
        if (ref == TRUE)
        {
            DbgPrint("[+] Hook已挂钩 \n");
        }
    }

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

编译并运行这段驱动程序，则你会看到挂钩成功的提示信息；


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

```
#    Time          Debug Print
18   8.25711060    hello lyshark.com
19   8.25724602    读取内存地址 = 00007FFF0BDF0000
20   8.25746918    ZwQueryVirtualMemory内存地址 = 00007FFF0BC9C750
21   8.25747108    [+] Hook已挂钩
```