在笔者的上一篇文章《内核特征码扫描PE代码段》中LyShark带大家通过封装好的LySharkToolsUtilKernelBase函数实现了动态获取内核模块基址，并通过ntimage.h头文件中提供的系列函数解析了指定内核模块的PE节表参数，本章将继续延申这个话题，实现对PE文件导出表的解析任务，导出表无法动态获取，解析导出表则必须读入内核模块到内存才可继续解析，所以我们需要分两步走，首先读入内核磁盘文件到内存，然后再通过ntimage.h中的系列函数解析即可。

PE结构（Portable Executable Structure）是Windows操作系统用于执行可执行文件和动态链接库（DLL）的标准格式。导出表（Export Table）是PE结构中的一个部分，它记录了一个DLL中所有可供外部调用的函数和变量。

导出表通常位于PE结构的数据目录中。它包含两个重要的表格：导出名称表格和导出地址表格。导出名称表格列出了DLL中所有导出函数和变量的名称，而导出地址表格列出了这些函数和变量的内存地址。

当PE文件执行时Windows装载器将文件装入内存并将导入表中登记的DLL文件一并装入，再根据DLL文件中函数的导出信息对可执行文件的导入表(IAT)进行修正。导出函数在DLL文件中，导出信息被保存在导出表，导出表就是记载着动态链接库的一些导出信息。通过导出表，DLL文件可以向系统提供导出函数的名称、序号和入口地址等信息，以便Windows装载器能够通过这些信息来完成动态链接的整个过程。

导出函数存储在PE文件的导出表里，导出表的位置存放在PE文件头中的数据目录表中，与导出表对应的项目是数据目录中的首个IMAGE_DATA_DIRECTORY结构，从这个结构的VirtualAddress字段得到的就是导出表的RVA值，导出表同样可以使用函数名或序号这两种方法导出函数。

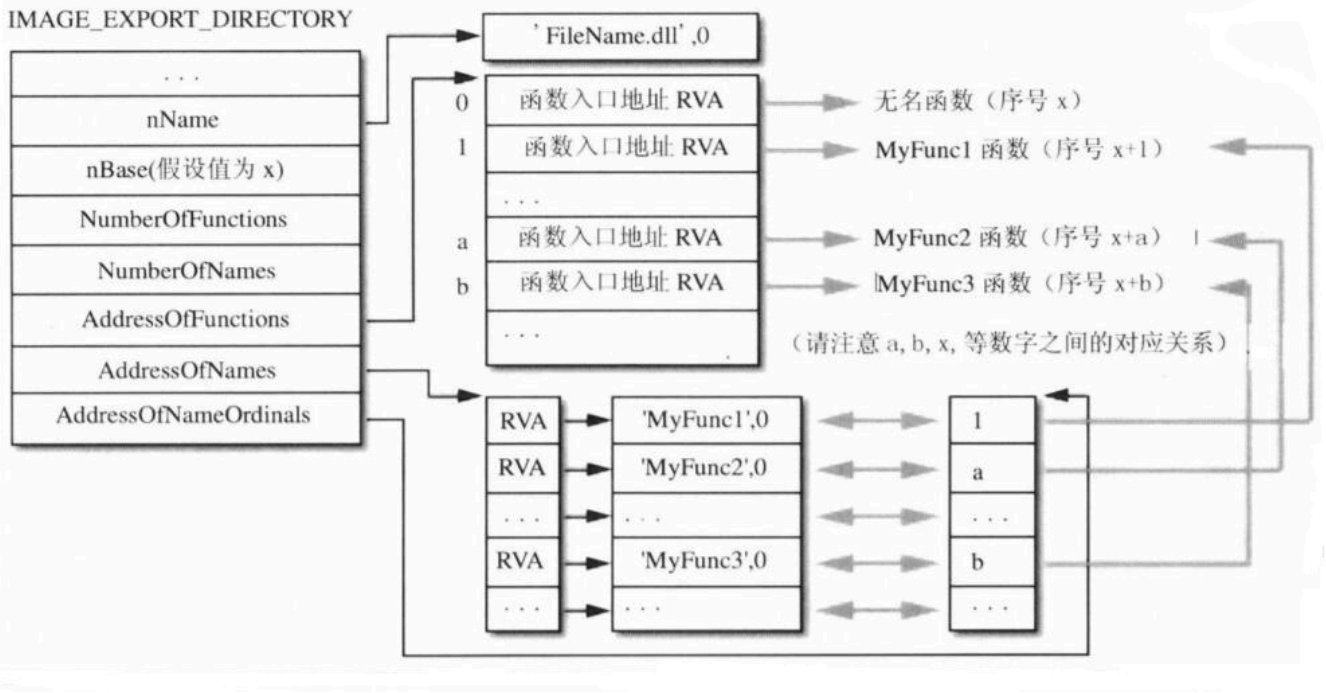
导出表的起始位置有一个IMAGE_EXPORT_DIRECTORY结构，与导入表中有多个IMAGE_IMPORT_DESCRIPTOR结构不同，导出表只有一个IMAGE_EXPORT_DIRECTORY结构，该结构定义如下：

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;         // 文件的产生时刻
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   Name;                   // 指向文件名的RVA
    DWORD   Base;                   // 导出函数的起始序号
    DWORD   NumberOfFunctions;      // 导出函数总数
    DWORD   NumberOfNames;          // 以名称导出函数的总数
    DWORD   AddressOfFunctions;     // 导出函数地址表的RVA
    DWORD   AddressOfNames;         // 函数名称地址表的RVA
    DWORD   AddressOfNameOrdinals; // 函数名序号表的RVA
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

其中，Name字段指向了该DLL的名称字符串，Base字段为该DLL的加载基地址，NumberOfFunctions和NumberOfNames分别表示导出函数和变量的数量，AddressOfFunctions、AddressOfNames和AddressOfNameOrdinals则是三个表格的地址。

总的来说，导出表是DLL中非常重要的一个部分，它提供了一种方便的方法，使其他程序可以调用DLL中的函数和变量。

上面的_IMAGE_EXPORT_DIRECTORY结构如果总结成一张图，如下所示：



在上图中最左侧`AddressOfNames`结构成员指向了一个数组，数组里保存着一组RVA，每个RVA指向一个字符串即导出的函数名，与这个函数名对应的是`AddressOfNameOrdinals`中的结构成员，该对应项存储的正是函数的唯一编号并与`AddressOfFunctions`结构成员相关联，形成了一个导出链式结构体。

获取导出函数地址时，先在`AddressOfNames`中找到对应的名字`MyFunc1`，该函数在`AddressOfNames`中是第1项，然后从`AddressOfNameOrdinals`中取出第1项的值这里是1，然后就可以通过导出函数的序号`AddressOfFunctions[1]`取出函数的入口RVA，然后通过RVA加上模块基址便是第一个导出函数的地址，向后每次相加导出函数偏移即可依次遍历出所有的导出函数地址。

其解析过程与应用层基本保持一致，如果不懂应用层如何解析也可以去看我以前写过的`《PE格式：手写PE结构解析工具》`里面具体详细的分析了解析流程。

首先使用`InitializeObjectAttributes()`打开文件，打开后可获取到该文件的句柄，`InitializeObjectAttributes`宏初始化一个`OBJECT_ATTRIBUTES`结构体，当一个例程打开对象时由此结构体指定目标对象的属性，此函数的微软定义如下；

```
VOID InitializeObjectAttributes(
  [out]           POBJECT_ATTRIBUTES   p,      // 权限
  [in]            PUNICODE_STRING      n,      // 文件名
  [in]            ULONG                a,      // 输出文件
  [in]            HANDLE               r,      // 权限
  [in, optional] PSECURITY_DESCRIPTOR s        // 0
);
```

当权限句柄被初始化后则即调用`ZwOpenFile()`打开一个文件使用权限`FILE_SHARE_READ`打开，打开文件函数微软定义如下；

```
NTSYSAPI NTSTATUS ZwOpenFile(
  [out] PHANDLE            FileHandle,          // 返回打开文件的句柄
  [in]  ACCESS_MASK        DesiredAccess,       // 打开的权限，一般设为GENERIC_ALL。
  [in]  POBJECT_ATTRIBUTES ObjectAttributes,    // OBJECT_ATTRIBUTES结构
  [out] PIO_STATUS_BLOCK   IoStatusBlock,       // 指向一个结构体的指针。该结构体指明打开文件的状态。
  [in]  ULONG              ShareAccess,         // 共享的权限。可以是FILE_SHARE_READ 或者
FILE_SHARE_WRITE。
  [in]  ULONG              OpenOptions          // 打开选项，一般设为
FILE_SYNCHRONOUS_IO_NONALERT。
);
```

接着文件被打开后，我们还需要调用`ZwCreateSection()`该函数的作用是创建一个`Section`节对象，并以PE结构中的`SectionALignment`大小对齐映射文件，其微软定义如下；

```
NTSYSAPI NTSTATUS ZwCreateSection(
  [out]           PHANDLE            SectionHandle,            // 指向 HANDLE 变量的指针，该变量
接收 section 对象的句柄。
  [in]            ACCESS_MASK        DesiredAccess,            // 指定一个 ACCESS_MASK 值，该值
确定对 对象的请求访问权限。
  [in, optional] POBJECT_ATTRIBUTES ObjectAttributes,         // 指向 OBJECT_ATTRIBUTES 结构
的指针，该结构指定对象名称和其他属性。
  [in, optional] PLARGE_INTEGER     MaximumSize,              // 指定节的最大大小（以字节为单
位）。
  [in]            ULONG              SectionPageProtection,    // 指定要在 节中的每个页面上放置的
保护。
  [in]            ULONG              AllocationAttributes,     // 指定确定节的分配属性的SEC_XXX
标志的位掩码。
  [in, optional] HANDLE             FileHandle                // （可选）指定打开的文件对象的句
柄。
);
```

最后读取导出表就要将一个磁盘中的文件映射到内存中，内存映射核心文件时`ZwMapViewOfSection()`该系列函数在应用层名叫`MapViewOfSection()`只是一个是内核层一个应用层，这两个函数参数传递基本一致，以`ZwMapViewOfSection`为例，其微软定义如下；

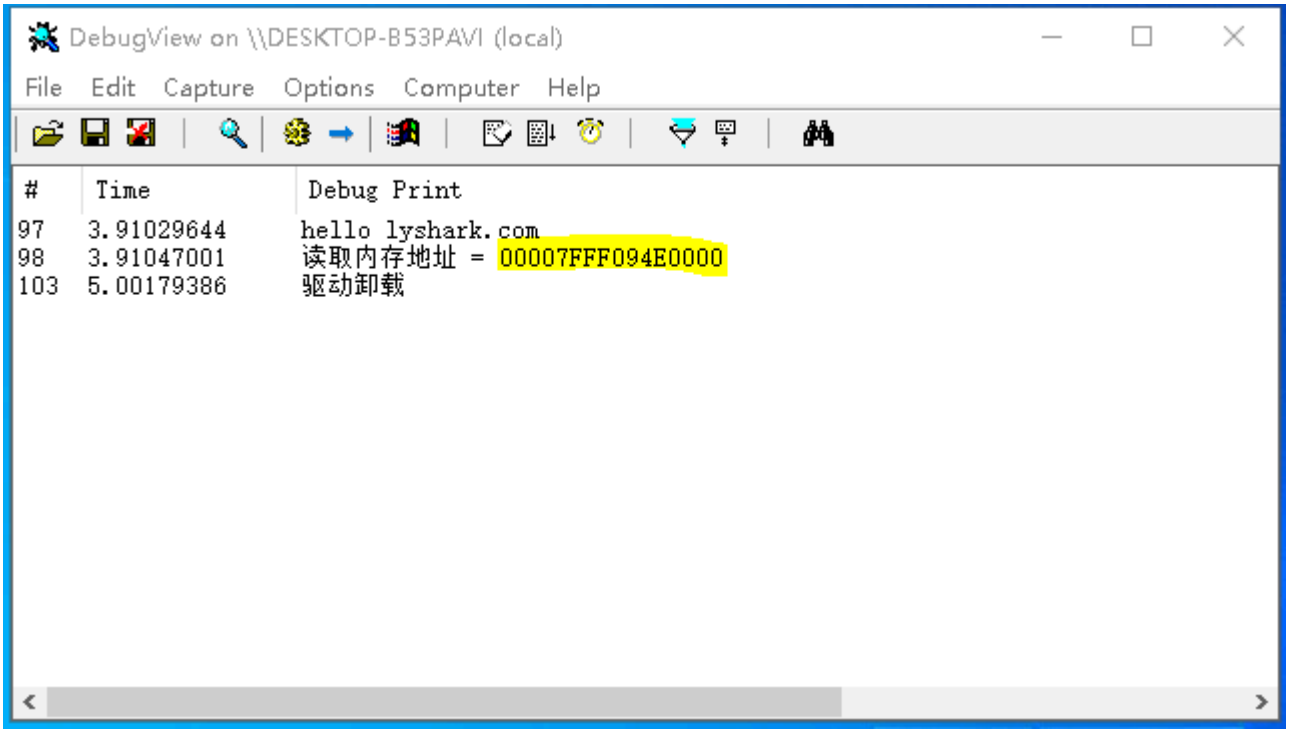
```
NTSYSAPI NTSTATUS ZwMapViewOfSection(
  [in]                HANDLE          SectionHandle,          // 接收一个节对象
  [in]                HANDLE          ProcessHandle,          // 进程句柄,此处使用
NtCurrentProcess()获取自身句柄
  [in, out]           PVOID           *BaseAddress,           // 指定填充地址
  [in]                ULONG_PTR       ZeroBits,               // 0
  [in]                SIZE_T          CommitSize,             // 每次提交大小 1024
  [in, out, optional] PLARGE_INTEGER  SectionOffset,          // 0
  [in, out]           PSIZE_T         ViewSize,               // 浏览大小
  [in]                SECTION_INHERIT InheritDisposition,     // ViewShare
  [in]                ULONG           AllocationType,         // 分配类型 MEM_TOP_DOWN
  [in]                ULONG           Win32Protect            // 权限 PAGE_READWRITE(读写)
);
```

将如上函数研究明白那么代码就变得很容易了，首先`InitializeObjectAttributes`设置文件权限与属性，然后调用`ZwOpenFile`打开文件，接着调用`ZwCreateSection`创建节对象，最后调用`ZwMapViewOfSection`将磁盘文件映射到内存，这段代码实现起来很简单，完整案例如下所示；

```
#include <ntifs.h>
#include <ntimage.h>
#include <ntstrsafe.h>

// 内存映射文件
NTSTATUS KernelMapFile(UNICODE_STRING FileName, HANDLE *phFile, HANDLE *phSection, PVOID
*ppBaseAddress)
{
    NTSTATUS status = STATUS_SUCCESS;
    HANDLE hFile = NULL;
    HANDLE hSection = NULL;
    OBJECT_ATTRIBUTES objectAttr = { 0 };
    IO_STATUS_BLOCK iosb = { 0 };
    PVOID pBaseAddress = NULL;
    SIZE_T viewSize = 0;

    // 设置文件权限
    InitializeObjectAttributes(&objectAttr, &FileName, OBJ_CASE_INSENSITIVE |
OBJ_KERNEL_HANDLE, NULL, NULL);

    // 打开文件
    status = ZwOpenFile(&hFile, GENERIC_READ, &objectAttr, &iosb, FILE_SHARE_READ,
FILE_SYNCHRONOUS_IO_NONALERT);
    if (!NT_SUCCESS(status))
    {
        return status;
    }

    // 创建节对象
    status = ZwCreateSection(&hSection, SECTION_MAP_READ | SECTION_MAP_WRITE, NULL, 0,
PAGE_READWRITE, 0x1000000, hFile);
    if (!NT_SUCCESS(status))
    {
        ZwClose(hFile);
        return status;
    }
    // 映射到内存
    status = ZwMapViewOfSection(hSection, NtCurrentProcess(), &pBaseAddress, 0, 1024, 0,
&viewSize, ViewShare, MEM_TOP_DOWN, PAGE_READWRITE);
    if (!NT_SUCCESS(status))
    {
        ZwClose(hSection);
        ZwClose(hFile);
        return status;
    }

    // 返回数据
    *phFile = hFile;
```

```
    *phSection = hSection;
    *ppBaseAddress = pBaseAddress;

    return status;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("驱动卸载 \n");
}

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};

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

    // 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);
    }

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

运行这段程序，即可读取到 `ntoskrnl.exe` 磁盘所在文件的内存映像基地址，效果如下所示；



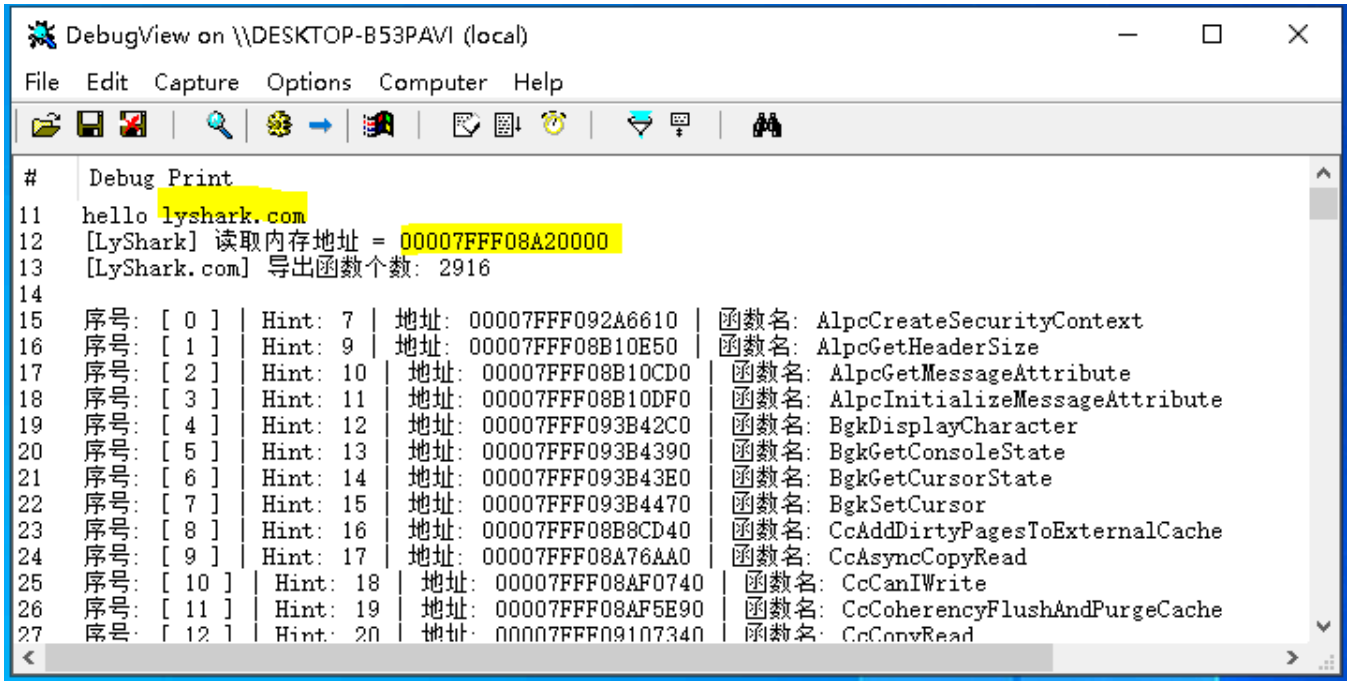
如上代码读入了 `ntoskrnl.exe` 文件，接下来就是解析导出表，首先将 `pBaseAddress` 解析为 `PIMAGE_DOS_HEADER` 获取DOS头，并在DOS头中寻找 `PIMAGE_NT_HEADERS` 头，接着在 `NTHeader` 头中得到数据目录表，此处指向的就是导出表 `PIMAGE_EXPORT_DIRECTORY` 通过 `pExportTable->NumberOfNames` 可得到导出表的数量，通过 `(PUCHAR)pDosHeader + pExportTable->AddressOfNames` 得到导出表的地址，依次循环读取即可得到完整的导出表。

```
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 };
    LONG FunctionIndex = 0;

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

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

    // Dos 头
    PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)pBaseAddress;

    // NT 头
```

```
    PIMAGE_NT_HEADERS pNtHeaders = (PIMAGE_NT_HEADERS)((PUCHAR)pDosHeader + pDosHeader-
>e_lfanew);

    // 导出表
    PIMAGE_EXPORT_DIRECTORY pExportTable = (PIMAGE_EXPORT_DIRECTORY)((PUCHAR)pDosHeader +
pNtHeaders->OptionalHeader.DataDirectory[0].VirtualAddress);

    // 有名称的导出函数个数
    ULONG ulNumberOfNames = pExportTable->NumberOfNames;
    DbgPrint("[LyShark.com] 导出函数个数: %d \n\n", ulNumberOfNames);

    // 导出函数名称地址表
    PULONG lpNameArray = (PULONG)((PUCHAR)pDosHeader + pExportTable->AddressOfNames);
    PCHAR lpName = NULL;

    // 开始遍历导出表(输出ulNumberOfNames导出函数)
    for (ULONG i = 0; i < ulNumberOfNames; i++)
    {
        lpName = (PCHAR)((PUCHAR)pDosHeader + lpNameArray[i]);

        // 获取导出函数地址
        USHORT uHint = *(USHORT *)((PUCHAR)pDosHeader + pExportTable->AddressOfNameOrdinals
+ 2 * i);
        ULONG ulFuncAddr = *(PULONG)((PUCHAR)pDosHeader + pExportTable->AddressOfFunctions +
4 * uHint);
        PVOID lpFuncAddr = (PVOID)((PUCHAR)pDosHeader + ulFuncAddr);

        // 获取SSDT函数Index
        FunctionIndex = *(ULONG *)((PUCHAR)lpFuncAddr + 4);

        DbgPrint("序号: [ %d ] | Hint: %d | 地址: %p | 函数名: %s \n", i, uHint, lpFuncAddr,
lpName);
    }

    // 释放指针
    ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress);
    ZwClose(hSection);
    ZwClose(hFile);

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

代码运行后即可获取到当前`ntoskrnl.exe`程序中的所有导出函数，输出效果如下所示；

- SSDT表通常会解析`\\??\\C:\\Windows\\System32\\ntoskrnl.exe`
- SSSDT表通常会解析`\\??\\C:\\Windows\\System32\\win32k.sys`

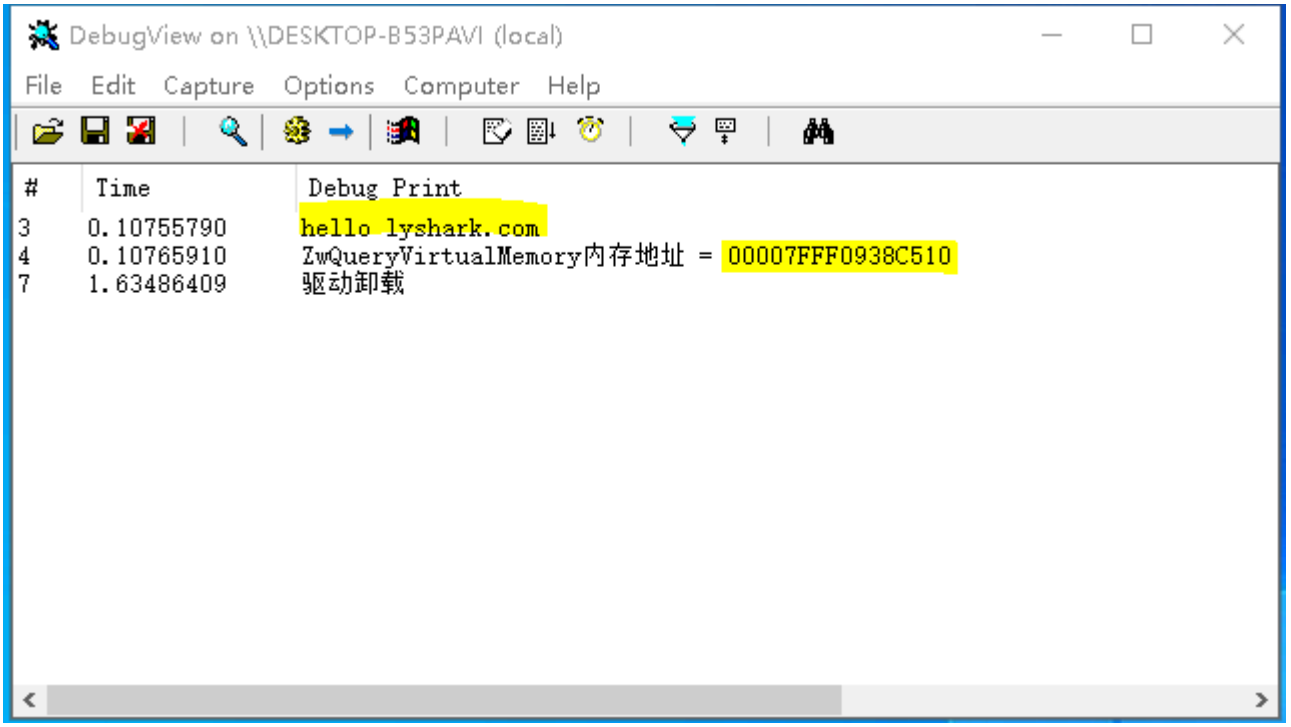


根据上方的函数流程将其封装为`GetAddressFromFunction()`用户传入`DllFileName`指定的PE文件，以及需要读取的`pszFunctionName`函数名，即可输出该函数的导出地址。

```
// 寻找指定函数得到内存地址
ULONG64 GetAddressFromFunction(UNICODE_STRING DllFileName, PCHAR pszFunctionName)
{
    NTSTATUS status = STATUS_SUCCESS;
    HANDLE hFile = NULL;
    HANDLE hSection = NULL;
    PVOID pBaseAddress = NULL;

    // 内存映射文件
    status = KernelMapFile(DllFileName, &hFile, &hSection, &pBaseAddress);
    if (!NT_SUCCESS(status))
    {
        return 0;
    }
    PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)pBaseAddress;
    PIMAGE_NT_HEADERS pNtHeaders = (PIMAGE_NT_HEADERS)((PUCHAR)pDosHeader + pDosHeader->e_lfanew);
    PIMAGE_EXPORT_DIRECTORY pExportTable = (PIMAGE_EXPORT_DIRECTORY)((PUCHAR)pDosHeader + pNtHeaders->OptionalHeader.DataDirectory[0].VirtualAddress);
    ULONG ulNumberOfNames = pExportTable->NumberOfNames;
    PULONG lpNameArray = (PULONG)((PUCHAR)pDosHeader + pExportTable->AddressOfNames);
    PCHAR lpName = NULL;

    for (ULONG i = 0; i < ulNumberOfNames; i++)
    {
        lpName = (PCHAR)((PUCHAR)pDosHeader + lpNameArray[i]);
        USHORT uHint = *(USHORT *)((PUCHAR)pDosHeader + pExportTable->AddressOfNameOrdinals + 2 * i);
        ULONG ulFuncAddr = *(PULONG)((PUCHAR)pDosHeader + pExportTable->AddressOfFunctions + 4 * uHint);
        PVOID lpFuncAddr = (PVOID)((PUCHAR)pDosHeader + ulFuncAddr);
```

```
        if (_strnicmp(pszFunctionName, lpName, strlen(pszFunctionName)) == 0)
        {
            ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress);
            ZwClose(hSection);
            ZwClose(hFile);

            return (ULONG64)lpFuncAddr;
        }
    }
    ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress);
    ZwClose(hSection);
    ZwClose(hFile);
    return 0;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("驱动卸载 \n");
}

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

    UNICODE_STRING FileName = { 0 };
    ULONG64 FunctionAddress = 0;

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

    // 取函数内存地址
    FunctionAddress = GetAddressFromFunction(FileName, "ZwQueryVirtualMemory");
    DbgPrint("ZwQueryVirtualMemory内存地址 = %p \n", FunctionAddress);

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

如上程序所示，当运行后即可获取到 `ntdll.dll` 模块内 `ZwQueryVirtualMemory` 的导出地址，输出效果如下所示；


DebugView on \\DESKTOP-B53PAVI (local)
File Edit Capture Options Computer Help
Time Debug Print
3 0.10755790 hello lyshark.com
4 0.10765910 ZwQueryVirtualMemory内存地址 = 00007FFF0938C510
7 1.63486409 驱动卸载