在笔者上篇文章`《内核扫描SSDT挂钩状态》`中简单介绍了如何扫描被挂钩的SSDT函数，并简单介绍了如何解析导出表，本章将继续延申PE导出表的解析，实现一系列灵活的解析如通过传入函数名解析出函数的RVA偏移，ID索引，Index下标等参数，并将其封装为可直接使用的函数，以在后期需要时可以被直接引用，同样为了节约篇幅本章中的`LoadKernelFile()`内存映射函数如需要使用请去前一篇文章中自行摘取。

首先实现`GetRvaFromModuleName()`函数，当用户传入参数后自动将函数名解析为对应的RVA偏移或Index下标索引值，该函数接收三个参数传递，分别是`wzFileName`模块名，`FunctionName`所在模块内的函数名，`Flag`标志参数，函数输出`ULONG64`类型的数据。

```
// 从指定模块中得到特定函数的RVA或相对序号相对偏移
ULONG64 GetRvaFromModuleName(WCHAR *wzFileName, UCHAR *FunctionName, INT Flag)
{
    // 加载内核模块
    PVOID BaseAddress = LoadKernelFile(wzFileName);

    // 取出导出表
    PIMAGE_DOS_HEADER pDosHeader;
    PIMAGE_NT_HEADERS pNtHeaders;
    PIMAGE_SECTION_HEADER pSectionHeader;
    ULONGLONG FileOffset;
    PIMAGE_EXPORT_DIRECTORY pExportDirectory;

    // DLL内存数据转成DOS头结构
    pDosHeader = (PIMAGE_DOS_HEADER)BaseAddress;

    // 取出PE头结构
    pNtHeaders = (PIMAGE_NT_HEADERS)((ULONGLONG)BaseAddress + pDosHeader->e_lfanew);

    // 判断PE头导出表是否为空
    if (pNtHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress == 0)
    {
        return 0;
    }

    // 取出导出表偏移
    FileOffset = pNtHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress;

    // 取出节头结构
    pSectionHeader = (PIMAGE_SECTION_HEADER)((ULONGLONG)pNtHeaders + sizeof(IMAGE_NT_HEADERS));
    PIMAGE_SECTION_HEADER pOldSectionHeader = pSectionHeader;

    // 遍历节结构进行地址运算
    for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
    {
        if (pSectionHeader->VirtualAddress <= FileOffset && FileOffset <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
        {
            FileOffset = FileOffset - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
        }
    }

    // 导出表地址
    pExportDirectory = (PIMAGE_EXPORT_DIRECTORY)((ULONGLONG)BaseAddress + FileOffset);

    // 取出导出表函数地址
    PULONG AddressOfFunctions;
    FileOffset = pExportDirectory->AddressOfFunctions;

    // 遍历节结构进行地址运算
    pSectionHeader = pOldSectionHeader;
    for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
    {
        if (pSectionHeader->VirtualAddress <= FileOffset && FileOffset <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
        {
            FileOffset = FileOffset - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
        }
    }
    AddressOfFunctions = (PULONG)((ULONGLONG)BaseAddress + FileOffset);

    // 取出导出表函数名字
    PUSHORT AddressOfNameOrdinals;
    FileOffset = pExportDirectory->AddressOfNameOrdinals;

    // 遍历节结构进行地址运算
    pSectionHeader = pOldSectionHeader;
    for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
    {
        if (pSectionHeader->VirtualAddress <= FileOffset && FileOffset <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
        {
            FileOffset = FileOffset - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
        }
    }
    AddressOfNameOrdinals = (PUSHORT)((ULONGLONG)BaseAddress + FileOffset);

    // 取出导出表函数序号
    PULONG AddressOfNames;
    FileOffset = pExportDirectory->AddressOfNames;

    // 遍历节结构进行地址运算
    pSectionHeader = pOldSectionHeader;
    for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
    {
        if (pSectionHeader->VirtualAddress <= FileOffset && FileOffset <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
        {
            FileOffset = FileOffset - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
        }
    }
    AddressOfNames = (PULONG)((ULONGLONG)BaseAddress + FileOffset);

    // 分析导出表
    ULONG uOffset;
    LPSTR FunName;
    ULONG uAddressOfNames;
    ULONG TargetOff = 0;

    for (ULONG uIndex = 0; uIndex < pExportDirectory->NumberOfNames; uIndex++, AddressOfNames++, AddressOfNameOrdinals++)
    {
        uAddressOfNames = *AddressOfNames;
        pSectionHeader = pOldSectionHeader;
        for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
        {
            if (pSectionHeader->VirtualAddress <= uAddressOfNames && uAddressOfNames <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
            {
                uOffset = uAddressOfNames - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
            }
        }
        FunName = (LPSTR)((ULONGLONG)BaseAddress + uOffset);

        // 如果找到则返回RVA
        if (!_stricmp((const char *)FunctionName, FunName))
        {
            // 等于1则返回RVA
            if (Flag == 1)
            {
                TargetOff = (ULONG)AddressOfFunctions[*AddressOfNameOrdinals];
                // DbgPrint("索引 [ %p ] 函数名 [ %s ] 相对RVA [ %p ] \n", *AddressOfNameOrdinals, FunName, TargetOff);
                return TargetOff;
            }
            // 返回索引
            else if (Flag == 0)
            {
                return *AddressOfNameOrdinals;
            }
        }
    }

    // 结束后释放内存
    ExFreePoolWithTag(BaseAddress, (ULONG)"LyShark");
    return 0;
}
```

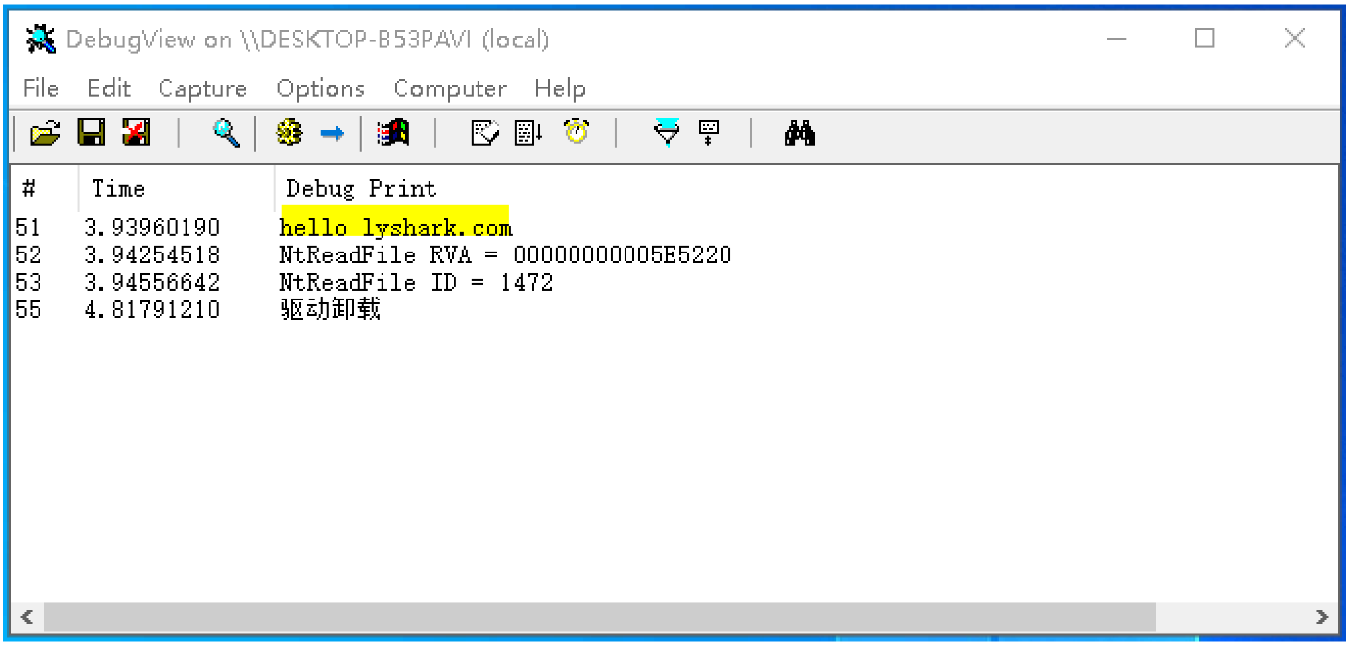
调用该函数很容易，传入模块路径以及该模块内的函数名，解析出RVA地址或Index下标。

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    // 函数分别传入 [模块路径,函数名,标志=1] 返回该导出函数的RVA
    ULONG64 get_rva = GetRvaFromModuleName(L"\\SystemRoot\\system32\\ntoskrnl.exe",
"NtReadFile", 1);
    DbgPrint("NtReadFile RVA = %p \n", get_rva);

    // 函数分别传入 [模块路径,函数名,标志=0] 返回该导出函数的ID下标
    ULONG64 get_id = GetRvaFromModuleName(L"\\SystemRoot\\system32\\ntoskrnl.exe",
"NtReadFile", 0);
    DbgPrint("NtReadFile ID = %d \n", get_id);

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

编译并运行程序，分别获取到`ntoskrnl.exe`模块内`NtReadFile`函数的RVA,Index索引，调用效果如下；



```
#    Time          Debug Print
51   3.93960190    hello lyshark.com
52   3.94254518    NtReadFile RVA = 00000000005E5220
53   3.94556642    NtReadFile ID = 1472
55   4.81791210    驱动卸载
```

第二个函数`GetModuleNameFromRVA()`则实现传入RVA或者函数Index序号，解析出函数名，具体实现方法与如上函数基本一致，仅仅只是在过滤时做了调整。

```
// 根据传入的函数RVA或Index下标，获取该函数的函数名
PCHAR GetModuleNameFromRVA(WCHAR *wzFileName, ULONG64 uRVA, INT Flag)
{
    // 加载内核模块
    PVOID BaseAddress = LoadKernelFile(wzFileName);

    // 取出导出表
    PIMAGE_DOS_HEADER pDosHeader;
    PIMAGE_NT_HEADERS pNtHeaders;
    PIMAGE_SECTION_HEADER pSectionHeader;
    ULONGLONG FileOffset;
    PIMAGE_EXPORT_DIRECTORY pExportDirectory;

    // DLL内存数据转成DOS头结构
    pDosHeader = (PIMAGE_DOS_HEADER)BaseAddress;

    // 取出PE头结构
    pNtHeaders = (PIMAGE_NT_HEADERS)((ULONGLONG)BaseAddress + pDosHeader->e_lfanew);

    // 判断PE头导出表是否为空
    if (pNtHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress == 0)
    {
        return 0;
    }

    // 取出导出表偏移
    FileOffset = pNtHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress;

    // 取出节头结构
    pSectionHeader = (PIMAGE_SECTION_HEADER)((ULONGLONG)pNtHeaders + sizeof(IMAGE_NT_HEADERS));
    PIMAGE_SECTION_HEADER pOldSectionHeader = pSectionHeader;

    // 遍历节结构进行地址运算
    for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
    {
        if (pSectionHeader->VirtualAddress <= FileOffset && FileOffset <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
        {
            FileOffset = FileOffset - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
        }
    }

    // 导出表地址
    pExportDirectory = (PIMAGE_EXPORT_DIRECTORY)((ULONGLONG)BaseAddress + FileOffset);

    // 取出导出表函数地址
    PULONG AddressOfFunctions;
    FileOffset = pExportDirectory->AddressOfFunctions;

    // 遍历节结构进行地址运算
    pSectionHeader = pOldSectionHeader;
    for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
    {
        if (pSectionHeader->VirtualAddress <= FileOffset && FileOffset <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
        {
            FileOffset = FileOffset - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
        }
    }
    AddressOfFunctions = (PULONG)((ULONGLONG)BaseAddress + FileOffset);

    // 取出导出表函数名字
    PUSHORT AddressOfNameOrdinals;
    FileOffset = pExportDirectory->AddressOfNameOrdinals;

    // 遍历节结构进行地址运算
    pSectionHeader = pOldSectionHeader;
    for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
    {
        if (pSectionHeader->VirtualAddress <= FileOffset && FileOffset <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
        {
            FileOffset = FileOffset - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
        }
    }
    AddressOfNameOrdinals = (PUSHORT)((ULONGLONG)BaseAddress + FileOffset);

    // 取出导出表函数序号
    PULONG AddressOfNames;
    FileOffset = pExportDirectory->AddressOfNames;

    // 遍历节结构进行地址运算
    pSectionHeader = pOldSectionHeader;
    for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
    {
        if (pSectionHeader->VirtualAddress <= FileOffset && FileOffset <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
        {
            FileOffset = FileOffset - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
        }
    }
    AddressOfNames = (PULONG)((ULONGLONG)BaseAddress + FileOffset);

    // 分析导出表
    ULONG uOffset;
    LPSTR FunName;
    ULONG uAddressOfNames;
    ULONG TargetOff = 0;

    for (ULONG uIndex = 0; uIndex < pExportDirectory->NumberOfNames; uIndex++, AddressOfNames++, AddressOfNameOrdinals++)
    {
        uAddressOfNames = *AddressOfNames;
        pSectionHeader = pOldSectionHeader;
        for (UINT16 Index = 0; Index < pNtHeaders->FileHeader.NumberOfSections; Index++, pSectionHeader++)
        {
            if (pSectionHeader->VirtualAddress <= uAddressOfNames && uAddressOfNames <= pSectionHeader->VirtualAddress + pSectionHeader->SizeOfRawData)
            {
                uOffset = uAddressOfNames - pSectionHeader->VirtualAddress + pSectionHeader->PointerToRawData;
            }
        }

        FunName = (LPSTR)((ULONGLONG)BaseAddress + uOffset);
        TargetOff = (ULONG)AddressOfFunctions[*AddressOfNameOrdinals];

        // 等于1则通过RVA返回函数名
        if (Flag == 1)
        {
            if (uRVA == TargetOff)
            {
                return FunName;
            }
        }
        // 返回索引
        else if (Flag == 0)
        {
            if (uRVA == *AddressOfNameOrdinals)
            {
                return FunName;
            }
        }
    }

    // 结束后释放内存
    ExFreePoolWithTag(BaseAddress, (ULONG)"LyShark");
    return "None";
}
```

调用`GetModuleNameFromRVA()`并传入相应的RVA偏移或Index下标。

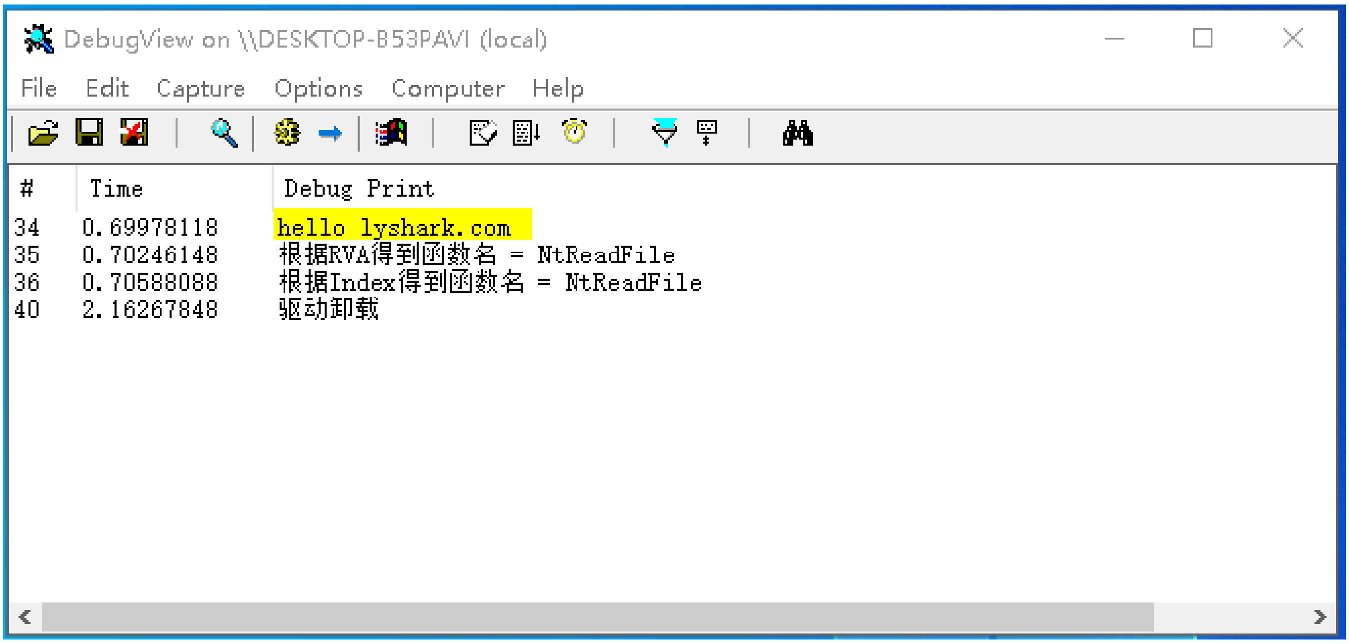
```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark.com \n");

    PCHAR function_name;

    // 传入函数RVA得到函数名
    function_name = GetModuleNameFromRVA(L"\\SystemRoot\\system32\\ntoskrnl.exe", 0x5e5220, 1);
    DbgPrint("根据RVA得到函数名 = %s \n", function_name);

    // 传入函数下标得到函数名
    function_name = GetModuleNameFromRVA(L"\\SystemRoot\\system32\\ntoskrnl.exe", 1472, 0);
    DbgPrint("根据Index得到函数名 = %s \n", function_name);
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

编译并运行程序，调用后分别获取到`RVA=0x5e5220`或`Index=1472`的函数名；



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

```
#   Time        Debug Print
34  0.69978118  hello lyshark.com
35  0.70246148  根据RVA得到函数名 = NtReadFile
36  0.70588088  根据Index得到函数名 = NtReadFile
40  2.16267848  驱动卸载
```