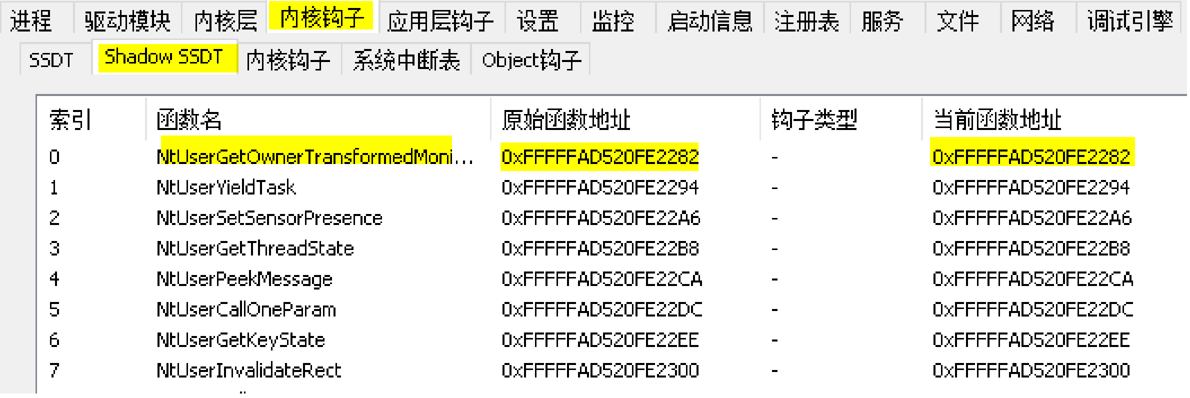
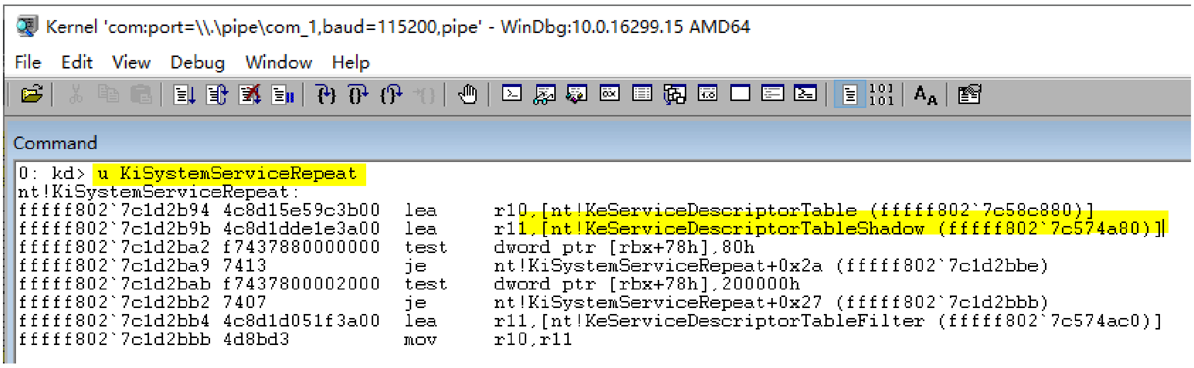
在笔者上一篇文章`《驱动开发：Win10枚举完整SSDT地址表》`实现了针对`SSDT`表的枚举功能，本章继续实现对`SSSDT`表的枚举，`ShadowSSDT`中文名`影子系统服务描述表`，SSSDT其主要的作用是管理系统中的图形化界面，其`Win32`子系统的内核实现是`Win32k.sys`驱动，属于GUI线程的一部分，其自身没有导出表，枚举`SSSDT`表其与`SSDT`原理基本一致。

如下是闭源ARK工具的枚举效果:



索引	函数名	原始函数地址	钩子类型	当前函数地址
0	NtUserGetOwnerTransformedMoni...	0xFFFFFAD520FE2282	-	0xFFFFFAD520FE2282
1	NtUserYieldTask	0xFFFFFAD520FE2294	-	0xFFFFFAD520FE2294
2	NtUserSetSensorPresence	0xFFFFFAD520FE22A6	-	0xFFFFFAD520FE22A6
3	NtUserGetThreadState	0xFFFFFAD520FE22B8	-	0xFFFFFAD520FE22B8
4	NtUserPeekMessage	0xFFFFFAD520FE22CA	-	0xFFFFFAD520FE22CA
5	NtUserCallOneParam	0xFFFFFAD520FE22DC	-	0xFFFFFAD520FE22DC
6	NtUserGetKeyState	0xFFFFFAD520FE22EE	-	0xFFFFFAD520FE22EE
7	NtUserInvalidateRect	0xFFFFFAD520FE2300	-	0xFFFFFAD520FE2300

首先需要找到`SSSDT`表的位置，通过`《驱动开发：Win10内核枚举SSDT表基址》`文章中的分析可知，SSSDT就在SSDT的下面，只需要枚举`4c8d1dde1e3a00`特征即可，如果你找不到上一篇具体分析流程了，那么多半你是看到了转载文章。


Kernel 'com:port=\\.\pipe\com_1,baud=115200,pipe' - WinDbg:10.0.16299.15 AMD64
File
Edit
View
Debug
Window
Help
Command

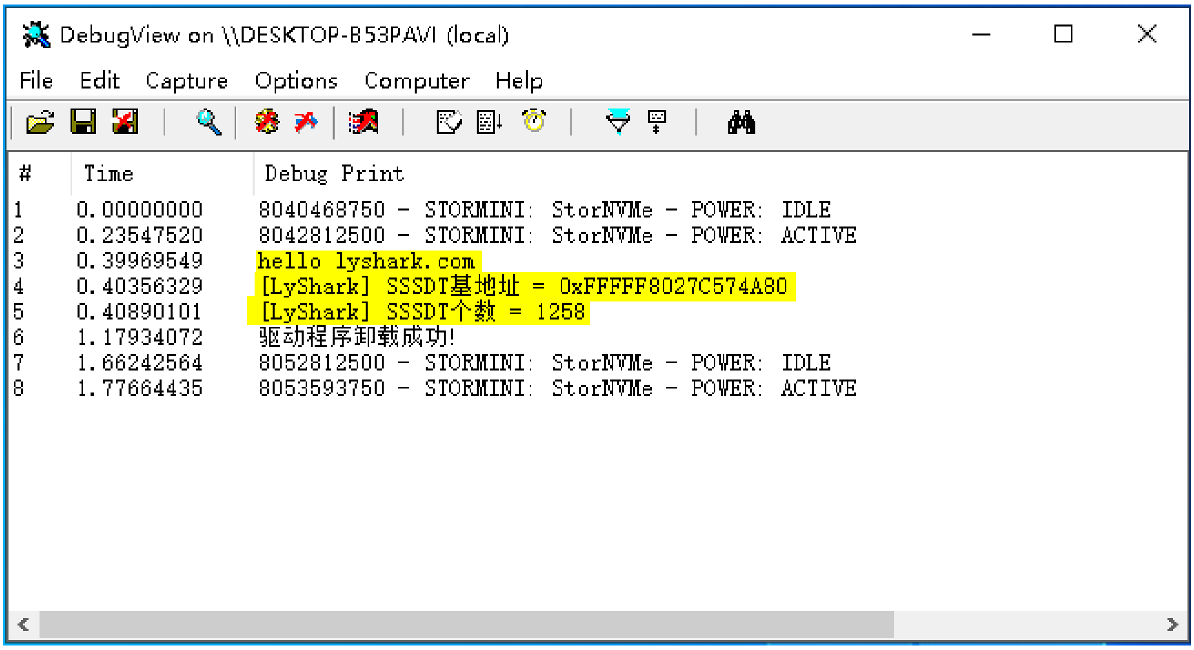
```
0: kd> u KiSystemServiceRepeat
nt!KiSystemServiceRepeat:
fffff802`7c1d2b94 4c8d15e59c3b00  lea     r10,[nt!KeServiceDescriptorTable (fffff802`7c58c880)]
fffff802`7c1d2b9b 4c8d1dde1e3a00  lea     r11,[nt!KeServiceDescriptorTableShadow (fffff802`7c574a80)]
fffff802`7c1d2ba2 f7437880000000  test    dword ptr [rbx+78h],80h
fffff802`7c1d2ba9 7413            je      nt!KiSystemServiceRepeat+0x2a (fffff802`7c1d2bbe)
fffff802`7c1d2bab f7437800002000  test    dword ptr [rbx+78h],200000h
fffff802`7c1d2bb2 7407            je      nt!KiSystemServiceRepeat+0x27 (fffff802`7c1d2bbb)
fffff802`7c1d2bb4 4c8d1d051f3a00  lea     r11,[nt!KeServiceDescriptorTableFilter (fffff802`7c574ac0)]
fffff802`7c1d2bbb 4d8bd3          mov     r10,r11
```

先实现第一个功能，得到`SSSDT`表的基地址以及`SSDT`函数个数，完整代码如下所示。

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

#include <ntifs.h>
#pragma intrinsic(__readmsr)

typedef struct _SYSTEM_SERVICE_TABLE
{
    PVOID           ServiceTableBase;
    PVOID           ServiceCounterTableBase;
    ULONGLONG       NumberOfServices;
    PVOID           ParamTableBase;
} SYSTEM_SERVICE_TABLE, *PSYSTEM_SERVICE_TABLE;

PSYSTEM_SERVICE_TABLE KeServiceDescriptorTableShadow = 0;
ULONG64 ul64W32pServiceTable = 0;
```

```
// 获取 KeServiceDescriptorTableShadow 首地址
ULONGLONG GetKeServiceDescriptorTableShadow()
{
    // 设置起始位置
    PUCHAR StartSearchAddress = (PUCHAR)__readmsr(0xC0000082) - 0x1808FE;

    // 设置结束位置
    PUCHAR EndSearchAddress = StartSearchAddress + 0x8192;
    // DbgPrint("扫描起始地址: %p --> 扫描结束地址: %p \n", StartSearchAddress, EndSearchAddress);

    PUCHAR ByteCode = NULL;

    UCHAR OpCodeA = 0, OpCodeB = 0, OpCodeC = 0;
    ULONGLONG addr = 0;
    ULONG templong = 0;

    for (ByteCode = StartSearchAddress; ByteCode < EndSearchAddress; ByteCode++)
    {
        // 使用MmIsAddressValid()函数检查地址是否有页面错误
        if (MmIsAddressValid(ByteCode) && MmIsAddressValid(ByteCode + 1) && MmIsAddressValid(ByteCode + 2))
        {
            OpCodeA = *ByteCode;
            OpCodeB = *(ByteCode + 1);
            OpCodeC = *(ByteCode + 2);

            // 对比特征值 寻找 nt!KeServiceDescriptorTable 函数地址
            /*
            lyshark.com kd> u KiSystemServiceRepeat
                nt!KiSystemServiceRepeat:
                fffff802`7c1d2b94 4c8d15e59c3b00  lea     r10,[nt!KeServiceDescriptorTable (fffff802`7c58c880)]
                fffff802`7c1d2b9b 4c8d1dde1e3a00  lea     r11,[nt!KeServiceDescriptorTableShadow (fffff802`7c574a80)]
                fffff802`7c1d2ba2 f7437880000000  test    dword ptr [rbx+78h],80h
                fffff802`7c1d2ba9 7413            je      nt!KiSystemServiceRepeat+0x2a (fffff802`7c1d2bbe)
                fffff802`7c1d2bab f7437800002000  test    dword ptr [rbx+78h],200000h
                fffff802`7c1d2bb2 7407            je      nt!KiSystemServiceRepeat+0x27 (fffff802`7c1d2bbb)
                fffff802`7c1d2bb4 4c8d1d051f3a00  lea     r11,[nt!KeServiceDescriptorTableFilter (fffff802`7c574ac0)]
                fffff802`7c1d2bbb 4d8bd3          mov     r10,r11
            */
            if (OpCodeA == 0x4c && OpCodeB == 0x8d && OpCodeC == 0x1d)
            {
                // 获取高位地址fffff802
                memcpy(&templong, ByteCode + 3, 4);

                // 与低位64da4880地址相加得到完整地址
                addr = (ULONGLONG)templong + (ULONGLONG)ByteCode + 7;
                return addr;
            }
        }
    }
    return  0;
}

// 得到SSSDT个数
ULONGLONG GetSSSDTCount()
{
    PSYSTEM_SERVICE_TABLE pWin32k;
    ULONGLONG W32pServiceTable;

    pWin32k = (PSYSTEM_SERVICE_TABLE)((ULONG64)KeServiceDescriptorTableShadow + sizeof(SYSTEM_SERVICE_TABLE));
    W32pServiceTable = (ULONGLONG)(pWin32k->ServiceTableBase);
    // DbgPrint("Count => %d \n", pWin32k->NumberOfServices);

    return pWin32k->NumberOfServices;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("驱动程序卸载成功！ \n"));
}

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

    KeServiceDescriptorTableShadow = (PSYSTEM_SERVICE_TABLE)GetKeServiceDescriptorTableShadow();

    DbgPrint("[LyShark] SSSDT基地址 = 0x%p \n", KeServiceDescriptorTableShadow);

    ULONGLONG count = GetSSSDTCount();

    DbgPrint("[LyShark] SSSDT个数 = %d \n", count);

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

这段代码运行后即可得到`SSSDT`表基地址，以及该表中函数个数。


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

```
#   Time          Debug Print
1   0.00000000    8040468750 - STORMINI: StorNVMe - POWER: IDLE
2   0.23547520    8042812500 - STORMINI: StorNVMe - POWER: ACTIVE
3   0.39969549    hello lyshark.com
4   0.40356329    [LyShark] SSSDT基地址 = 0xFFFF8027C574A80
5   0.40890101    [LyShark] SSSDT个数 = 1258
6   1.17934072    驱动程序卸载成功!
7   1.66242564    8052812500 - STORMINI: StorNVMe - POWER: IDLE
8   1.77664435    8053593750 - STORMINI: StorNVMe - POWER: ACTIVE
```

在此基础之上增加枚举计算过程即可，完整源代码如下所示。

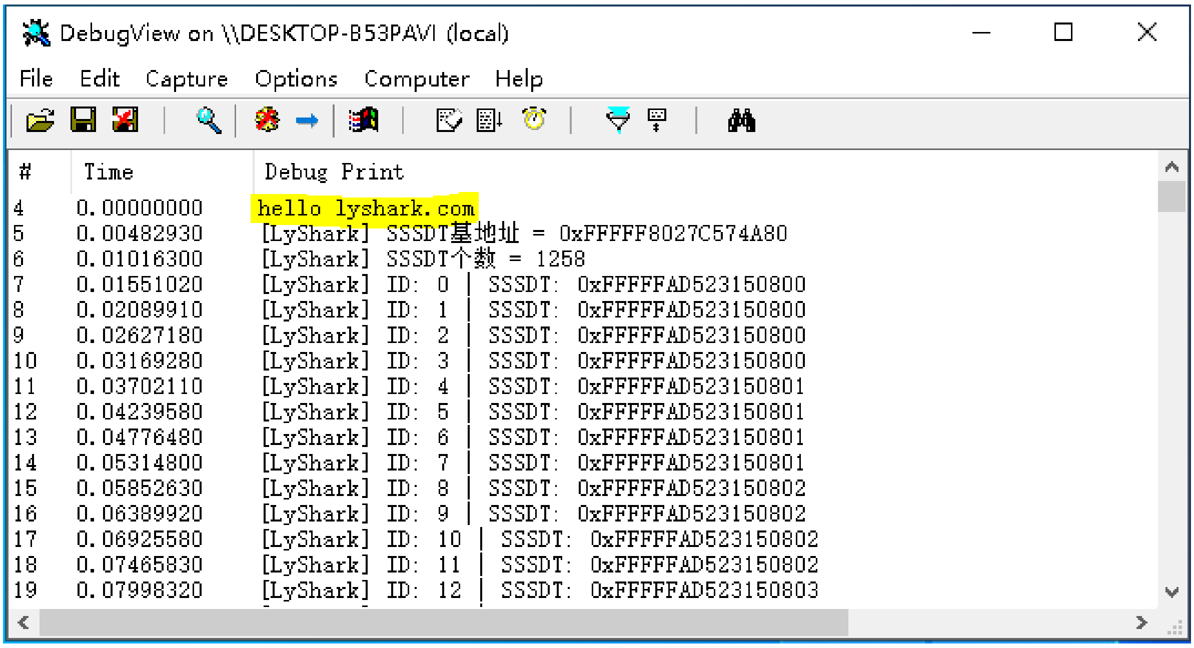
SSSDT 函数起始index是`0x1000`，但`W32pServiceTable`是从基址开始记录的，这个误差则需要`(index-0x1000)`来得到，至于`+4`则是下一个元素与上一个元素的偏移。

计算公式：

- W32pServiceTable + 4 * (index-0x1000)

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

#include <ntifs.h>
#pragma intrinsic(__readmsr)

typedef struct _SYSTEM_SERVICE_TABLE
{
    PVOID           ServiceTableBase;
    PVOID           ServiceCounterTableBase;
    ULONGLONG       NumberOfServices;
    PVOID           ParamTableBase;
} SYSTEM_SERVICE_TABLE, *PSYSTEM_SERVICE_TABLE;

PSYSTEM_SERVICE_TABLE KeServiceDescriptorTableShadow = 0;
ULONG64 ul64W32pServiceTable = 0;

// 获取 KeServiceDescriptorTableShadow 首地址
ULONGLONG GetKeServiceDescriptorTableShadow()
{
    // 设置起始位置
    PUCHAR StartSearchAddress = (PUCHAR)__readmsr(0xC0000082) - 0x1808FE;

    // 设置结束位置
    PUCHAR EndSearchAddress = StartSearchAddress + 0x8192;
    // DbgPrint("扫描起始地址: %p --> 扫描结束地址: %p \n", StartSearchAddress, EndSearchAddress);
    PUCHAR ByteCode = NULL;

    UCHAR OpCodeA = 0, OpCodeB = 0, OpCodeC = 0;
    ULONGLONG addr = 0;
    ULONG templong = 0;

    for (ByteCode = StartSearchAddress; ByteCode < EndSearchAddress; ByteCode++)
    {
        // 使用MmIsAddressValid()函数检查地址是否有页面错误
        if (MmIsAddressValid(ByteCode) && MmIsAddressValid(ByteCode + 1) &&
MmIsAddressValid(ByteCode + 2))
        {
            OpCodeA = *ByteCode;
            OpCodeB = *(ByteCode + 1);
            OpCodeC = *(ByteCode + 2);

            // 对比特征值 寻找 nt!KeServiceDescriptorTable 函数地址
            /*
            lyshark.com kd> u KiSystemServiceRepeat
            nt!KiSystemServiceRepeat:
            fffff802`7c1d2b94 4c8d15e59c3b00  lea     r10,
[nt!KeServiceDescriptorTable (fffff802`7c58c880)]
            fffff802`7c1d2b9b 4c8d1dde1e3a00  lea     r11,
[nt!KeServiceDescriptorTableShadow (fffff802`7c574a80)]
            fffff802`7c1d2ba2 f7437880000000  test    dword ptr [rbx+78h],80h
            fffff802`7c1d2ba9 7413            je
nt!KiSystemServiceRepeat+0x2a (fffff802`7c1d2bbe)
            fffff802`7c1d2bab f7437800002000  test    dword ptr
[rbx+78h],200000h
            fffff802`7c1d2bb2 7407            je
nt!KiSystemServiceRepeat+0x27 (fffff802`7c1d2bbb)
            fffff802`7c1d2bb4 4c8d1d051f3a00  lea     r11,
[nt!KeServiceDescriptorTableFilter (fffff802`7c574ac0)]
            fffff802`7c1d2bbb 4d8bd3          mov     r10,r11
            */
            if (OpCodeA == 0x4c && OpCodeB == 0x8d && OpCodeC == 0x1d)
            {
                // 获取高位地址fffff802
                memcpy(&templong, ByteCode + 3, 4);

                // 与低位64da4880地址相加得到完整地址
                addr = (ULONGLONG)templong + (ULONGLONG)ByteCode + 7;
                return addr;
            }
        }
    }
    return  0;
}

// 得到SSSDT个数
ULONGLONG GetSSSDTCount()
{
    PSYSTEM_SERVICE_TABLE pWin32k;
    ULONGLONG W32pServiceTable;
    pWin32k = (PSYSTEM_SERVICE_TABLE)((ULONG64)KeServiceDescriptorTableShadow + sizeof(SYSTEM_SERVICE_TABLE));
    W32pServiceTable = (ULONGLONG)(pWin32k->ServiceTableBase);
    // DbgPrint("Count => %d \n", pWin32k->NumberOfServices);

    return pWin32k->NumberOfServices;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("驱动程序卸载成功！ \n"));
}

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

    KeServiceDescriptorTableShadow = (PSYSTEM_SERVICE_TABLE)GetKeServiceDescriptorTableShadow();

    DbgPrint("[LyShark] SSSDT基地址 = 0x%p \n", KeServiceDescriptorTableShadow);

    ULONGLONG count = GetSSSDTCount();

    DbgPrint("[LyShark] SSSDT个数 = %d \n", count);

    // 循环枚举SSSDT
    for (size_t Index = 0; Index < count; Index++)
    {

        PSYSTEM_SERVICE_TABLE pWin32k;
        ULONGLONG W32pServiceTable;

        pWin32k = (PSYSTEM_SERVICE_TABLE)((ULONG64)KeServiceDescriptorTableShadow + sizeof(SYSTEM_SERVICE_TABLE));
        W32pServiceTable = (ULONGLONG)(pWin32k->ServiceTableBase);

        // 获取SSSDT地址
        //ln win32k!W32pServiceTable+((poi(win32k!W32pServiceTable+4*(1-1000))&0x00000000`ffffffff)>>4)-10000000
        //u win32k!W32pServiceTable+((poi(win32k!W32pServiceTable+4*(Index-0x1000))&0x00000000`ffffffff)>>4)-0x10000000

        //u poi(win32k!W32pServiceTable+4*(1-0x1000))
        //u poi(win32k!W32pServiceTable+4*(1-0x1000))&0x00000000`ffffffff
        //u (poi(win32k!W32pServiceTable+4*(1-0x1000))&0x00000000`ffffffff)>>4

        //u win32k!W32pServiceTable+((poi(win32k!W32pServiceTable+4*(1-0x1000))&0x00000000`ffffffff)>>4)-0x10000000

        ULONGLONG qword_temp = 0;
        LONG dw = 0;

        // SSSDT 下标从1000开始，而W32pServiceTable是从0开始
        // + 4 则是每次向下4字节就是下一个地址
        qword_temp = W32pServiceTable + 4 * (Index - 0x1000);

        dw = *(PLONG)qword_temp;
        // dw = qword_temp & 0x00000000ffffffff;
        dw = dw >> 4;
        qword_temp = W32pServiceTable + (LONG64)dw;

        DbgPrint("[LyShark] ID: %d | SSSDT: 0x%p \n", Index, qword_temp);
    }

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

枚举效果如下所示（存在问题）：



```
#     Time          Debug Print
4     0.00000000    hello lyshark.com
5     0.00482930    [LyShark] SSSDT基地址 = 0xFFFFF8027C574A80
6     0.01016300    [LyShark] SSSDT个数 = 1258
7     0.01551020    [LyShark] ID: 0 | SSSDT: 0xFFFFAD523150800
8     0.02089910    [LyShark] ID: 1 | SSSDT: 0xFFFFAD523150800
9     0.02627180    [LyShark] ID: 2 | SSSDT: 0xFFFFAD523150800
10    0.03169280    [LyShark] ID: 3 | SSSDT: 0xFFFFAD523150800
11    0.03702110    [LyShark] ID: 4 | SSSDT: 0xFFFFAD523150801
12    0.04239580    [LyShark] ID: 5 | SSSDT: 0xFFFFAD523150801
13    0.04776480    [LyShark] ID: 6 | SSSDT: 0xFFFFAD523150801
14    0.05314800    [LyShark] ID: 7 | SSSDT: 0xFFFFAD523150801
15    0.05852630    [LyShark] ID: 8 | SSSDT: 0xFFFFAD523150802
16    0.06389920    [LyShark] ID: 9 | SSSDT: 0xFFFFAD523150802
17    0.06925580    [LyShark] ID: 10 | SSSDT: 0xFFFFAD523150802
18    0.07465830    [LyShark] ID: 11 | SSSDT: 0xFFFFAD523150802
19    0.07998320    [LyShark] ID: 12 | SSSDT: 0xFFFFAD523150803
```

注这一步必须要在GUI线程中执行，否则会异常，建议将枚举过程写成DLL文件，注入到`explorer.exe`进程内执行。

> 本书作者： 王瑞 (LyShark)
> 作者邮箱： me@lyshark.com
> 作者博客： https://lyshark.cnblogs.com
> 团队首页： www.lyshark.com