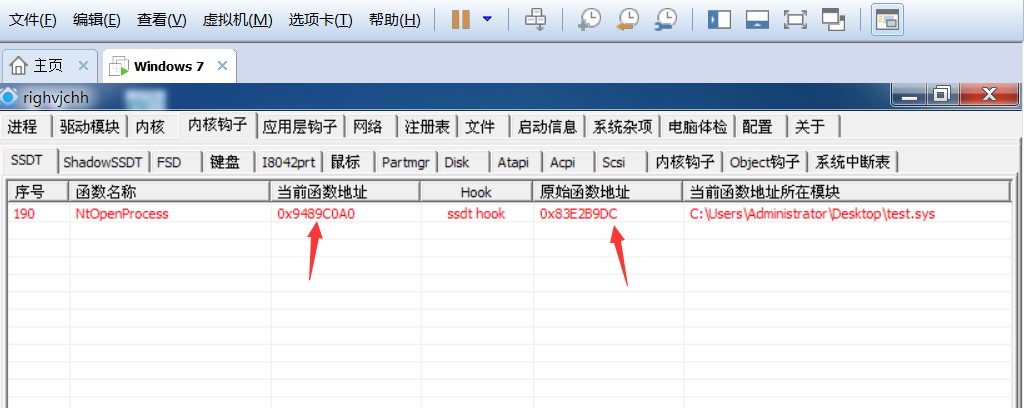
# SSDT 中文名称为系统服务描述符表，该表的作用是将Ring3应用层与Ring0内核层，两者的API函数连接起来，起到承上启下的作用，SSDT并不仅仅只包含一个庞大的地址索引表，它还包含着一些其它有用的信息， 诸如地址索引的基址、服务函数个数等，SSDT 通过修改此表的函数地址可以对常用 Windows 函数进行内核级的Hook，从而实现对一些核心的系统动作进行过滤、监控的目的。

通过前面的学习我们已经可以编写一个驱动程序并挂钩到指定的内核函数上了，接下来我们将一步步的通过

编写驱动程序，手动的来解除 函数的驱动保护，以此来模拟如何一步步干掉游戏保护。

NtOpenProcess

一般情况下当游戏启动的时候都会加载保护，而这种保护通常都是通过在SSDT层挂钩来实现的，而一旦函数被挂钩那么通过前面的读取方式就无法读取到函数的原始地址了，如下图是一个被Hook过的函数，可以看到函数的当前地址与原始地址已经发生了明显的变化。



那么如何获取到原始函数地址呢？很简单只需要使用系统提供给我们的 函数即可获取到原始函数的地址，最终测试代码如下:

MmGetSystemRoutineAddress

#include <ntddk.h>

extern "C" LONG KerServiceDescriptorTable;

ULONG Get\_SSDTAddr(){

UNICODE\_STRING NtOpen; ULONG SSDT\_Addr;

// 存放函数的Unicode字符串

// 用于存放原始的SSDT地址

// 将NtOpenProcess字符串以Uncode格式写入到NtOpen变量中

RtlInitUnicodeString(&NtOpen, L"NtOpenProcess");

// 获取系统程序地址，取得NtOpenProcess的原始地址

SSDT\_Addr = (ULONG)MmGetSystemRoutineAddress(&NtOpen);

DbgPrint("原始函数的地址是： %x\n", SSDT\_Addr); return SSDT\_Addr;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint(("驱动卸载成功 ! \n"));

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)

{

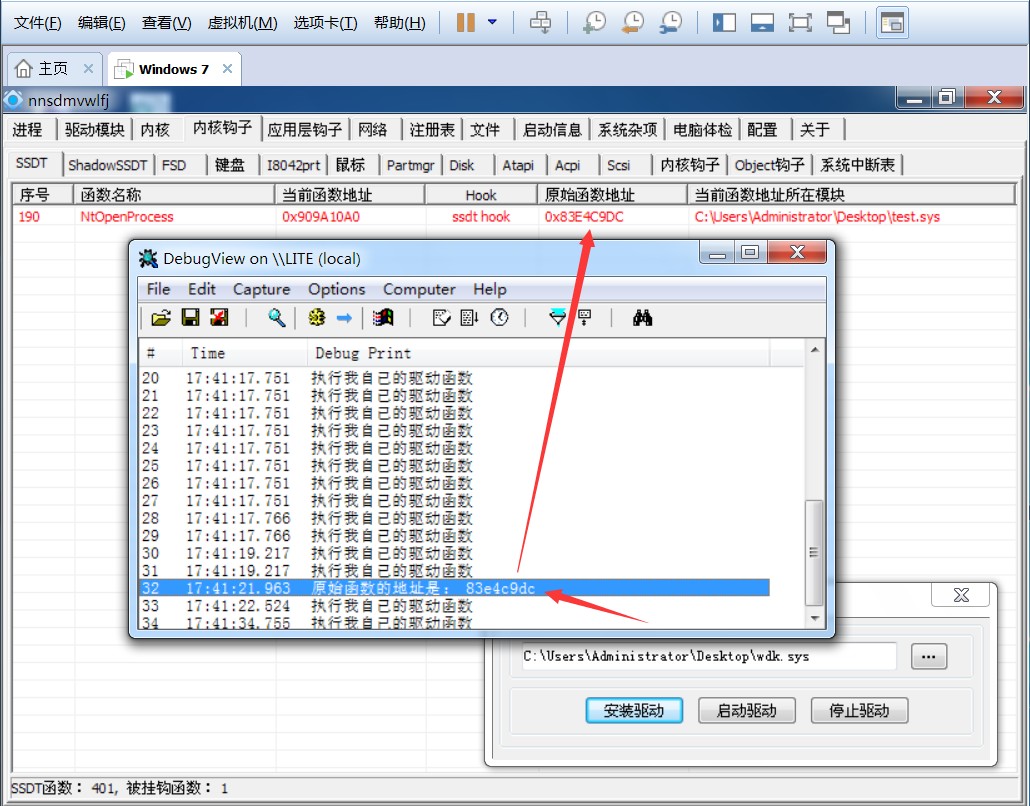
Get\_SSDTAddr();

DriverObject->DriverUnload = UnDriver;

return STATUS\_SUCCESS;

}

编译这段驱动代码，然后回到虚拟机并加载这段驱动，手动验证一下观察:



上方的驱动代码也可以改用汇编来实现，其效果是相同的，贴出汇编代码的实现流程，这里就不演示了。

#include <ntddk.h>

extern "C" LONG KerServiceDescriptorTable;

ULONG Get\_SSDTAddr(){

UNICODE\_STRING NtOpen; ULONG SSDT\_Addr;

// 存放函数的Unicode字符串

// 用于存放原始的SSDT地址

// 将NtOpenProcess字符串以Uncode格式写入到NtProcess变量中

RtlInitUnicodeString(&NtOpen, L"NtOpenProcess");

asm

{

lea eax, NtOpen // 将初始化的NtOpenProcess地址给EAX

push eax // EAX压入堆栈 等待调用call DWORD ptr DS:[MmGetSystemRoutineAddress] mov SSDT\_Addr,eax // 将结果赋值给变量

}

DbgPrint("原始函数的地址是： %x\n", SSDT\_Addr);

return SSDT\_Addr;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint(("驱动卸载成功 ! \n"));

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)

{

Get\_SSDTAddr();

DriverObject->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

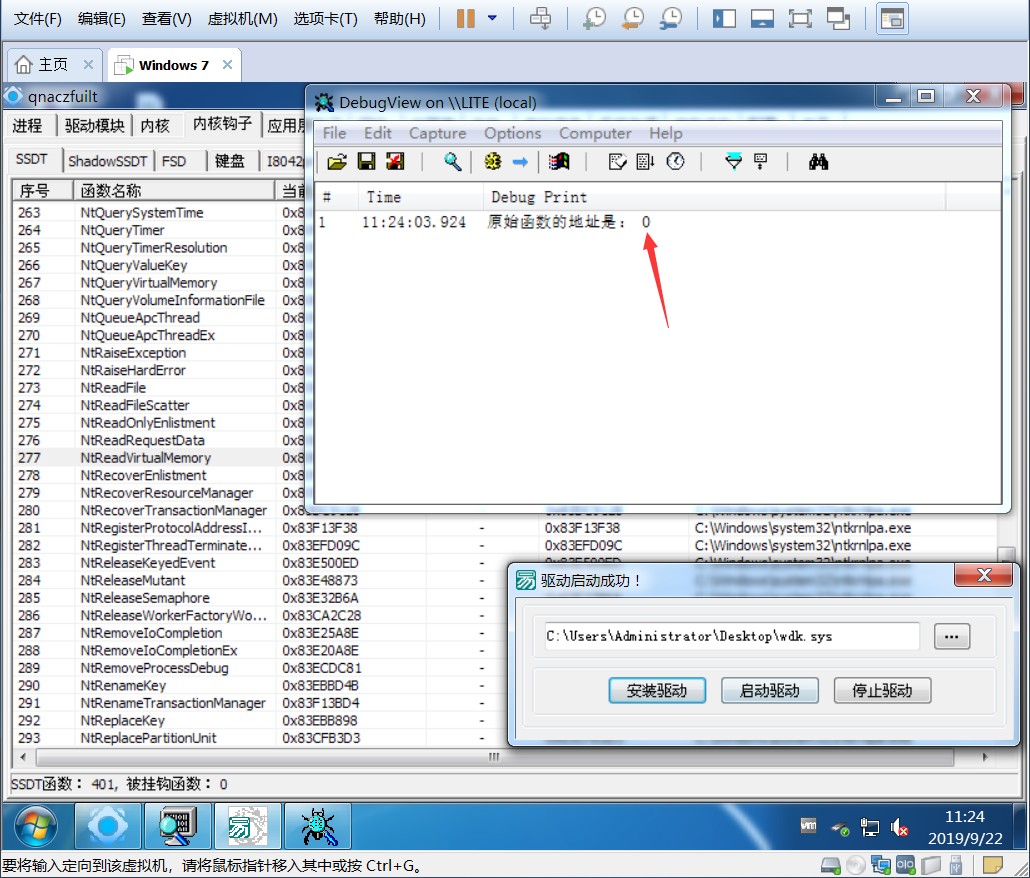
**通过偏移二次读取：** 上面的代码运行后只能获取到一部分函数的原始地址，有些函数的地址是无法获取到

# 的，比如我们想要获取

NtReadVirtualMemory

下获取结果始终显示为0。

这个内核函数的地址时，上方的代码就会显示获取失败，如



既然无法获取到当前函数的地址，那么我们可以尝试获取 NtReadVirtualMemory 函数的前一个函数的内存

# 地址，并通过相加偏移的方式来获取该函数的地址，首先我们通过Xuetr 查询到 函

NtReadVirtualMemory

数的当前地址，然后通过 调试器找到其对应的前一个函数的偏移。

WinDBG

18h

offset nt!NtBuildGUID+0xc9a4 (83c75a28) nt!strchr+0x118 (83c9dba8)

edi,dword ptr fs:[124h] al,byte ptr [edi+13Ah] byte ptr [ebp-1Ch],al esi,dword ptr [ebp+14h]

al,al

lkd> u 83e7f82c

nt!MmCopyVirtualMemory+0x50a: 83e7f82c 6a18 push

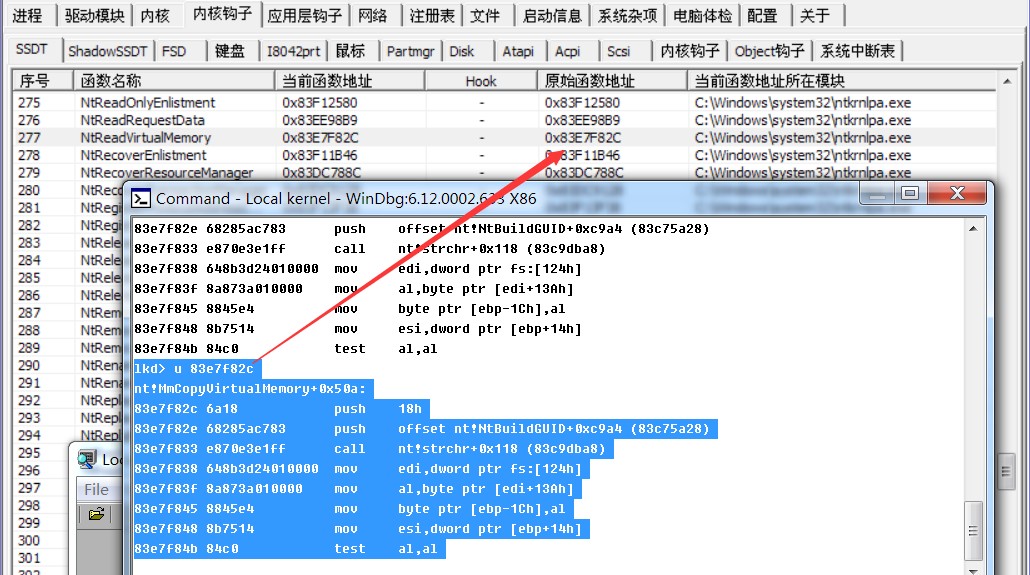
83e7f82e 68285ac783 push

83e7f833 e870e3e1ff call 83e7f838 648b3d24010000 mov

83e7f83f 8a873a010000 mov 83e7f845 8845e4 mov

83e7f848 8b7514 mov

83e7f84b 84c0 test



查询结果中可以发现上一个函数的是

MmCopyVirtualMemory

进上方的程序，即可实现查询，代码如下：

而相对应的偏移地址是 0x50a ，接着直接改

#include <ntddk.h>

extern "C" LONG KerServiceDescriptorTable;

ULONG Get\_SSDTAddr(){ UNICODE\_STRING NtOpen; ULONG SSDT\_Addr;

RtlInitUnicodeString(&NtOpen, L"MmCopyVirtualMemory"); SSDT\_Addr = (ULONG)MmGetSystemRoutineAddress(&NtOpen);

asm

{

push eax

mov eax,SSDT\_Addr add eax,50ah

mov SSDT\_Addr,eax

}

DbgPrint("原始函数的地址是： %x\n", SSDT\_Addr); return SSDT\_Addr;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

KdPrint(("Uninstall Driver Is OK \n"));

}

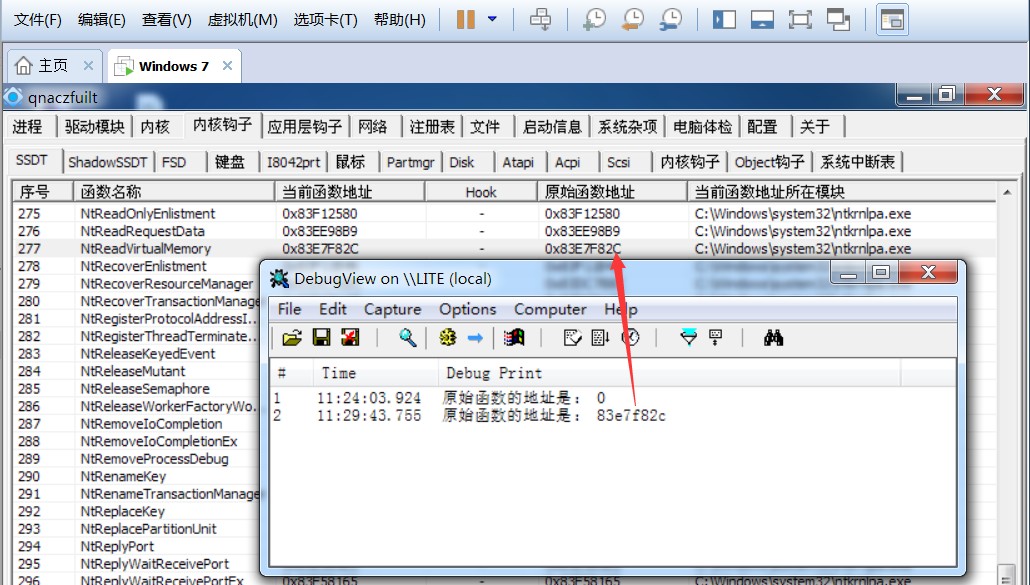
NTSTATUS DriverEntry(IN PDRIVER\_OBJECT DriverObject, IN PUNICODE\_STRING RegistryPath)

{

Get\_SSDTAddr();

DriverObject->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}



**判断函数是否被Hook：** 上方的代码中，我们可以通过使用 MmGetSystemRoutineAddress 函数来获取到函数的原始地址，而在第一部分我们又通过汇编的方式得到了函数的当前地址，通过使用当前地址与原始地址 做比较即可判断出函数是否被Hook。

#include <ntddk.h>

#include <windef.h> //包含windef.h文件byte字节才能使用extern "C" LONG KerServiceDescriptorTable;

extern "C" LONG KeServiceDescriptorTable;

typedef struct \_JMPDATE

{



BYTE E9; // 定义一个字节的e9成员名用来存放一字节数据

ULONG JMPADDR; // 定义JMPADDR成员名用来存放4字节跳转地址数据

}JMPDATE;

ULONG Get\_Origin\_SSDTAddr(){ // 获取到NTOpenProcess的原始地址

UNICODE\_STRING NtOpen; ULONG SSDT\_Addr;

RtlInitUnicodeString(&NtOpen, L"NtOpenProcess"); SSDT\_Addr = (ULONG)MmGetSystemRoutineAddress(&NtOpen); return SSDT\_Addr;

}

ULONG Get\_Now\_SSDTAddr(){ // 获取到NTOpenProcess的当前地址

ULONG SSDT\_Addr;

asm{

push ebx push eax

mov ebx, KeServiceDescriptorTable // 系统描述符号表的地址

mov ebx, [ebx] // 取服务表基址给EBX

mov eax, 0xBE // NtOpenProcess=转成十六进制等于BE

imul eax, eax, 4 // eax=eax 4 -> 7a 4=1e8

add ebx, eax // eax=1e8与服务表基址EBX相加

mov ebx, [ebx] // 取ebx里面的内容给EBX

mov SSDT\_Addr, ebx // 将得到的基址给变量

pop eax pop ebx

}

return SSDT\_Addr;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

KdPrint(("驱动卸载成功 !\n"));

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject,PUNICODE\_STRING RegistryPath)

{

ULONG Get\_Origin\_SSDT, Get\_Now\_SSDT; JMPDATE JmpDate;

Get\_Now\_SSDT = Get\_Now\_SSDTAddr(); // 获取NTOpenProcess的当前地址

Get\_Origin\_SSDT = Get\_Origin\_SSDTAddr(); // 获取原始的NTOpenProcess的地址

if (Get\_Now\_SSDT != Get\_Origin\_SSDT)

{

DbgPrint("该函数已经被Hook了! \n");

JmpDate.E9 = 0xe9; // 0xe9 机器码是 jmp指

令

JmpDate.JMPADDR = Get\_Origin\_SSDT - Get\_Now\_SSDT - 5; // 原始地址-当前地址-5 =

需要跳转的机器码数据

DbgPrint("写入了JMP数据=%x \n", JmpDate.JMPADDR);

}else

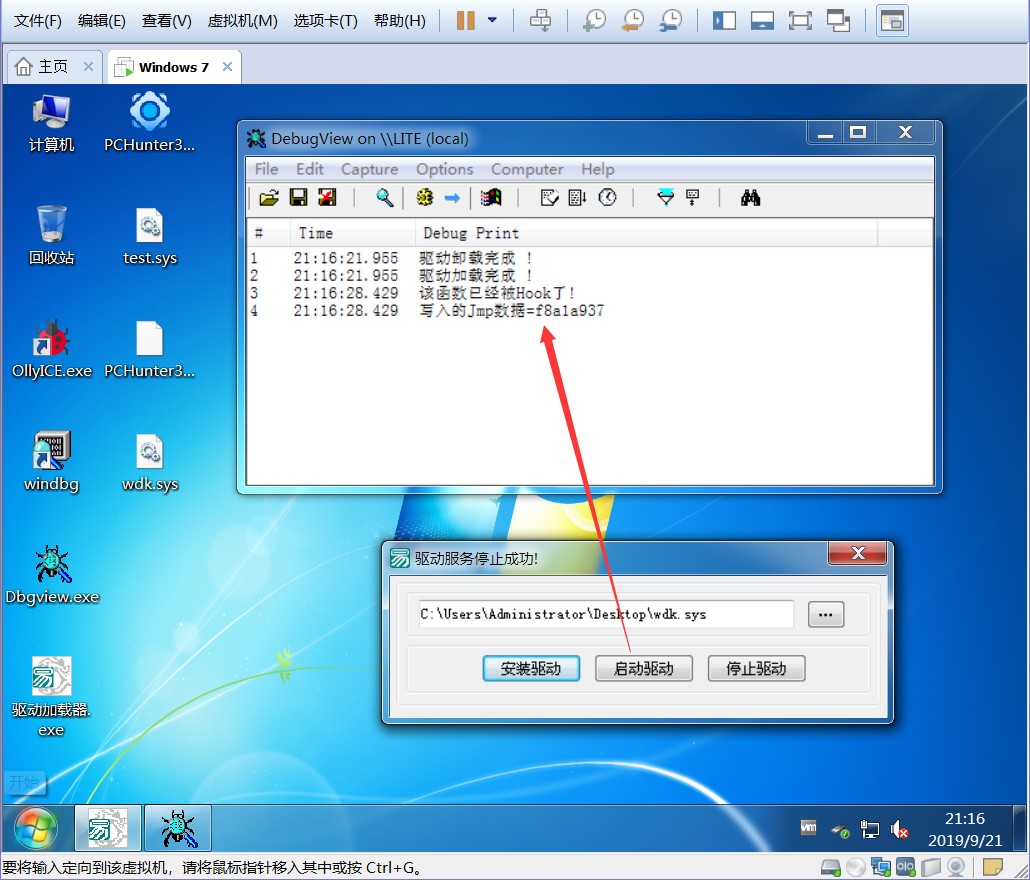
{

DbgPrint("该函数没有被Hook ! \n");

}

DriverObject->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}



**恢复被Hook过的函数：** 接下来我们通过编写驱动程序的方式恢复 NtOpenProcess 内核函数所Hook的地址，恢复Hook的原理非常的简单，只需要在函数头部添加一条 Jmp xxxx 并将其跳转到原始函数地址上面去即可恢复挂钩。

# 在下方的代码中需要注意一条计算公式

JmpDate.JMPADDR = Get\_Origin\_SSDT - Get\_Now\_SSDT - 5;

12345678 - 00401000 - 5

如下使用核心。

即可得到 E9 机器码后面的跳转地址 7346F411 这也是计算代码的

00401000 > - E9 7346F411 jmp 12345678

# 有了计算的公式，我们在前面的代码的基础上继续改进一下即可，最终代码如下:

#include <ntddk.h>

#include <windef.h> //包含windef.h文件extern "C" LONG KerServiceDescriptorTable;

extern "C" LONG KeServiceDescriptorTable;



#pragma pack(1) // 使下面的结构以一字节方式对齐，而不是默认的4字节对齐

typedef struct \_JMPDATE

{

BYTE E9; // 定义一个字节的e9成员名用来存放一字节数据

ULONG JMPADDR; // 定义JMPADDR成员名用来存放4字节跳转地址数据

}JMPDATE, PJMPDATE;

#pragma pack()

JMPDATE Origin\_Data; // 存放原始跳转数据

PJMPDATE pNow\_Data; // 存放当前跳转数据

ULONG Get\_Origin\_SSDTAddr(){ // 获取到NTOpenProcess的原始地址

UNICODE\_STRING NtOpen; ULONG SSDT\_Addr;

RtlInitUnicodeString(&NtOpen, L"NtOpenProcess"); SSDT\_Addr = (ULONG)MmGetSystemRoutineAddress(&NtOpen); return SSDT\_Addr;

}

ULONG Get\_Now\_SSDTAddr(){ // 获取到NTOpenProcess的当前地址

ULONG SSDT\_Addr;

asm{

push ebx

push eax

mov ebx, KeServiceDescriptorTable // 系统描述符号表的地址

mov ebx, [ebx] // 取服务表基址给EBX

mov eax, 0xBE // NtOpenProcess=转成十六进制等于BE

imul eax, eax, 4 // eax=eax 4 -> 7a 4=1e8

add ebx, eax // eax=1e8与服务表基址EBX相加

mov ebx, [ebx] // 取ebx里面的内容给EBX

mov SSDT\_Addr, ebx // 将得到的基址给变量

pop eax pop ebx

}

return SSDT\_Addr;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

asm //去掉内核页面保护

{

cli

mov eax, cr0

and eax, not 10000h mov cr0, eax

}

// 恢复原始地址

pNow\_Data->E9 = Origin\_Data.E9;

pNow\_Data->JMPADDR = Origin\_Data.JMPADDR;

asm //恢复内核页面保护

{

mov eax, cr0 or eax, 10000h mov cr0, eax sti

}

KdPrint(("驱动卸载成功 !\n"));

}

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)

{

ULONG Get\_Origin\_SSDT, Get\_Now\_SSDT; JMPDATE JmpDate;

Get\_Now\_SSDT = Get\_Now\_SSDTAddr(); // 获取NTOpenProcess的当前地址Get\_Origin\_SSDT = Get\_Origin\_SSDTAddr(); // 获取原始的NTOpenProcess的地址if (Get\_Now\_SSDT != Get\_Origin\_SSDT)

{

DbgPrint("该函数已经被Hook了! \n");

pNow\_Data = (PJMPDATE)(Get\_Now\_SSDT); // 初始化获取NtOpenProcess当前地址的

PJMPDATE结构指针保存在 pNow\_Data

Origin\_Data.E9 = pNow\_Data->E9; // 将pNow\_Data中E9里的内容给

Origin\_Data中的E9,1字节

Origin\_Data.JMPADDR = pNow\_Data->JMPADDR; // NtOpenProcess当前地址后4字节,保存在Origin\_Data.JMPADDR里面

// 取出当前的地址，然后保存到JmpData结构中

JmpDate.E9 = 0xe9; // 0xe9 机器码是 jmp指

令

JmpDate.JMPADDR = Get\_Origin\_SSDT - Get\_Now\_SSDT - 5; // 原始地址-当前地址-5 =

需要跳转的机器码数据

DbgPrint("写入JMP的数据 = %x \n", JmpDate.JMPADDR);

asm //去掉内核页面保护

{

cli

mov eax, cr0

and eax, not 10000h mov cr0, eax

}

pNow\_Data->E9 = JmpDate.E9; // jmp 1字节数据写入pNow\_Data->JMPADDR = JmpDate.JMPADDR; // 写入跳转到目标地址

asm //恢复内核页面保护

{

mov eax, cr0 or eax, 10000h mov cr0, eax sti

}

// 将原始地址的地址给ECX

// 取EBX的第一个字节给AL

// AL内容以字节方式写入ECX=原始地址里

// EBX=当前的地址+1的4字节内容给EAX

// EAX的内容写入ECX=原始地址+1偏移处

}

DriverObject->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 附上汇编版本的Hook恢复代码，如下，自行替换此处不做测试了。

asm{........}

asm

{

//保存写入前的数据,用于驱动卸载恢复。mov ebx, Get\_Now\_SSDT

// 挂钩代码汇编版本，替换到上方完整代码指定字段即可，此处不做演示。

//去掉内核页面保护

// 将当前的地址给EBX

lea ecx, Origin\_Data mov al, byte ptr[ebx] mov byte ptr[ecx], al mov eax, [ebx + 1]

mov[ecx + 1], eax

//写入数据

mov ebx, Get\_Now\_SSDT // 将当前的地址给EBX

lea ecx, JmpDate // 将JmpDate结构的地址给ECX

mov al, byte ptr[ecx] // 取出JmpDate结构地址的第一个字节给AL

mov byte ptr[ebx], al // al=JmpDate.E9 ,将数据写入到EBX=当前的地

址。

mov eax, [ecx + 1]

// [ECX+1]=JmpDate.JMPADDR ,将数据写入到

EAX里保存

mov[ebx + 1], eax

// 将EAX的数据写入到EBX+1=dangqian的地址+1

偏移处。

}

asm{........} //恢复内核页面保护

// 恢复代码汇编版本，替换到上方完整代码指定字段即可，此处不做演示。

asm{........} //去掉内核页面保护

asm

{

mov ebx, pNow\_Data // 将当前结构赋值到ebx中

lea ecx, Origin\_Data // 将原始结构的地址让ECX保存，等候使用

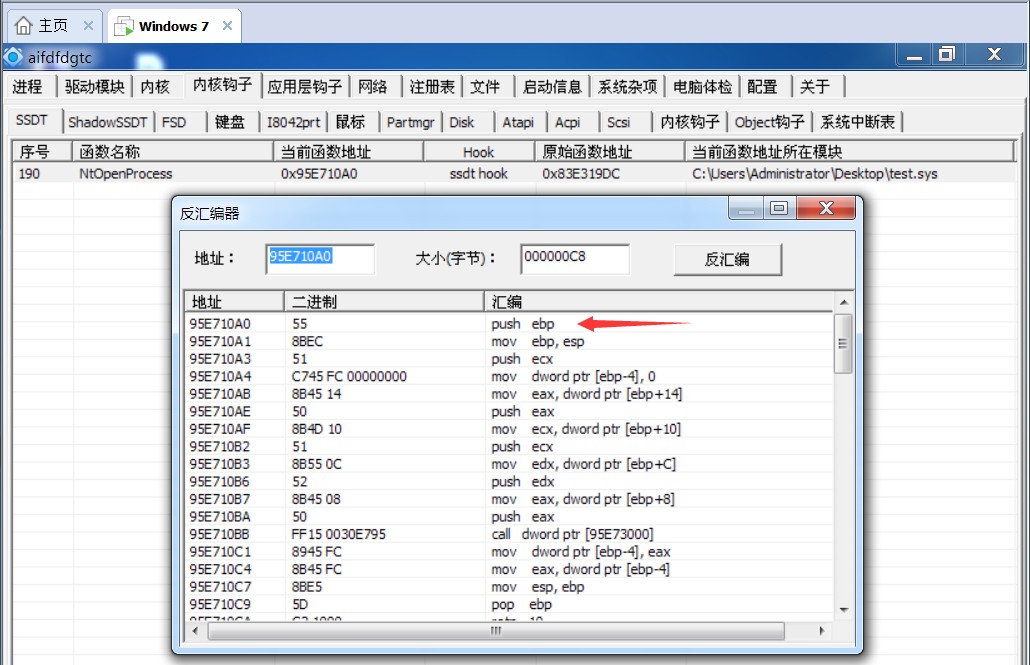
mov al, byte ptr[ecx] // 取出原始结构的地址中第一个字节的数据给AL

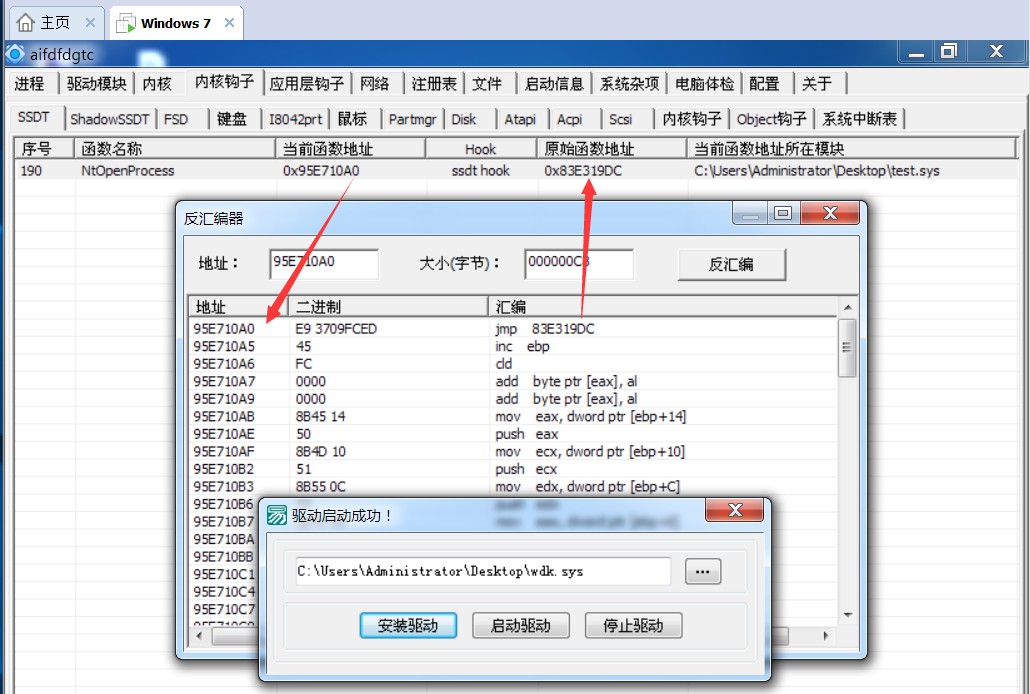
mov byte ptr[ebx], al // 将AL的数据以一个字节的方式写入ebx=当前的地址。mov eax, [ecx + 1] // 取出原始结构的地址+1偏移处开始的4字节数据给EAX。mov[ebx + 1], eax // EAX的数据以4字节的方式写入到当前的地址+1偏移处。

}

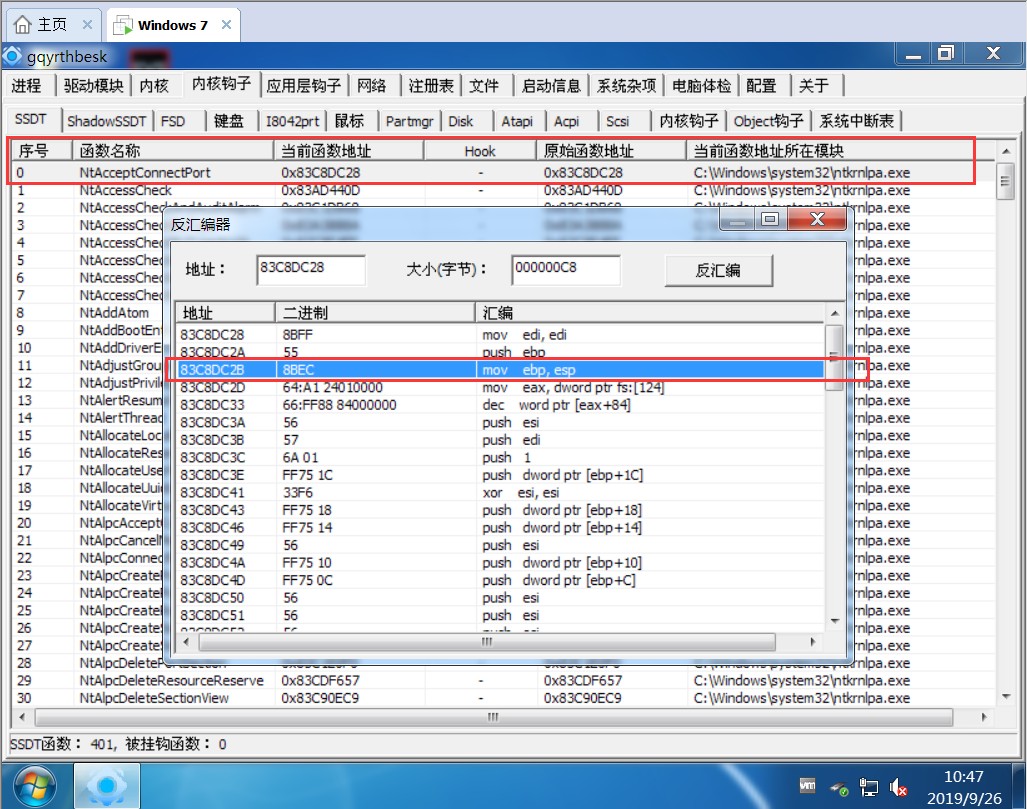
asm{........} //恢复内核页面保护

# 将代码编译，并拖入虚拟机加载驱动，Hook之前如图一所示，Hook之后如图二，发现程序已经跳转到了原始的代码上了，Hook被解除啦。





**在任意位置写入恢复代码：** 上方的代码片段虽然可以恢复浅层的Hook，但如果保护驱动Hook的较深的话需上面的代码将无法恢复，我们需要使用如下代码.





NTSTATUS DriverEntry(IN PDRIVER\_OBJECT DriverObject, IN PUNICODE\_STRING RegistryPath)

{

BYTE Jmp\_OEP[5] = { 0xEB,0x28,0xDC,0xC8,0x83}; // jmp 83C8DC28 硬编码

BYTE NtOpen = (BYTE )0x83C8DC28; // 此处为了方便演示直接写地址

asm //去掉内核页面保护。

{

cli

mov eax, cr0

and eax, not 10000h mov cr0, eax

}

// 拷贝内存,在NtOpen的地址基础上加3，填充为 jmp 指令填充4字节

RtlCopyMemory(NtOpen+3, Jmp\_OEP, 5);

asm

{

mov eax, cr0 or eax, 10000h mov cr0, eax sti

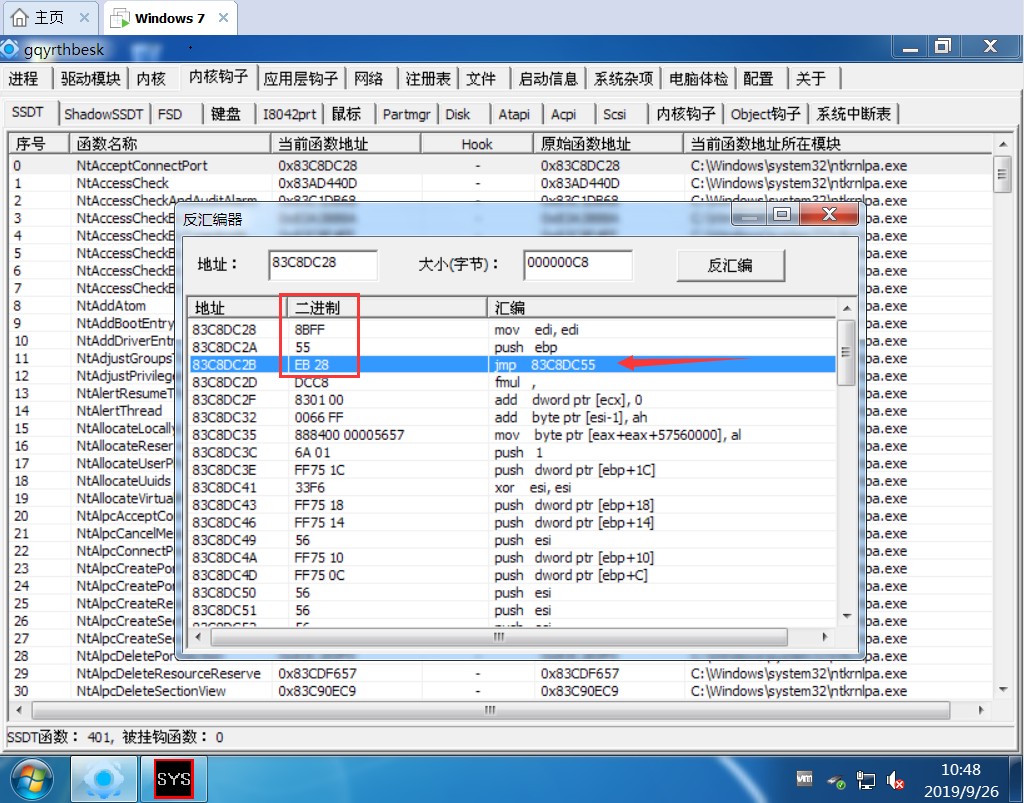
}

DriverObject->DriverUnload = UnDriver; return STATUS\_SUCCESS;

//恢复内核页面保护

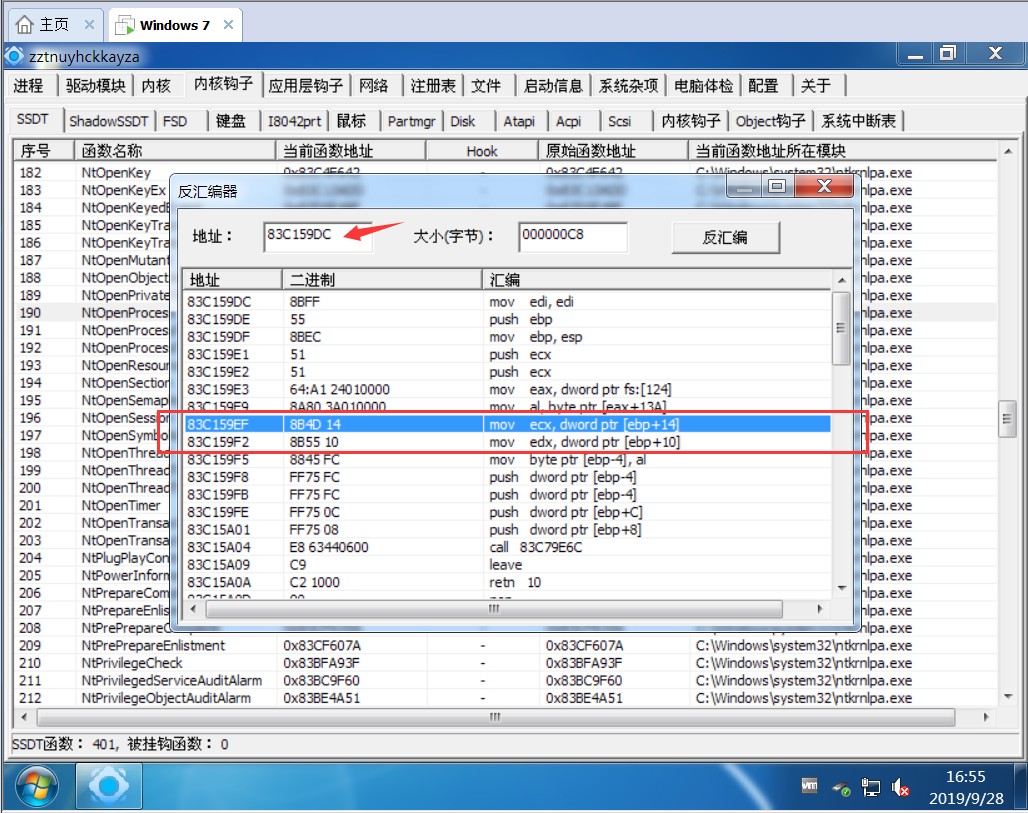
}

编译生成好代码以后，拖入虚拟机并加载驱动，观察内存变化，发现已经写入地址成功，我们可以使用该方 法写入任意位置，注意堆栈平衡，否则会直接蓝屏。



**给系统函数添加额外功能:** 通过使用Jmp跳转指令，我们可以给相应的系统函数添加新功能，以

# NtOpenProcess为例



核心汇编伪代码如下，这里并没有写全，可以自行完善:



#include <ntddk.h> #include <windef.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

KdPrint(("Uninstall Driver Is OK \n"));

}

BYTE RetAddr = NULL;

BYTE MyHook = NULL;

// 保存函数的返回地址

// Hook要修改的地址

// 对于jmp类型hook 如果没有使用\_declspec(naked)修饰，会破破坏我们的堆栈平衡

// 对于call类型的hook，如果使用\_declspec(naked)修饰的话，要注意自己恢复堆栈平衡

declspec(naked) VOID inline\_NtOpenProcess()

{

asm

{

mov ecx, dword ptr[ebp + 14] // 这两条原始代码必须写在这里

mov edx, dword ptr[ebp + 10] mov eax, RetAddr

jmp eax

}



}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath)

{

BYTE NtOpenProcess = (BYTE )0x83C159DC; // 函数的基址: 为方便演示这里直接填上

了

BYTE Jmp\_Addr[6] = { 0xE9, 0, 0, 0, 0,0x90}; // JMP跳转字节码 0x0=等待添加跳转地址 0x90 为填充字节(凑够6字节)

asm

{

push eax

mov eax, NtOpenProcess add eax, 0x13

(NtOpenProcess起始位置) = 相对偏移

mov MyHook, eax add eax, 0x06

间隔

mov RetAddr, eax pop eax

}

// 取出NtOpenProcess函数的基地址,原始地址

// 我们要写入数据的地方 (写入数据的位置)-

// 相加后获取到需要Hook的位置

// 获取到函数的返回地址，相加后得到，0x6h就是指令的

// 获取返回的地址

// Jmp\_Addr+1=指向E9后面的字节

// inline\_NtOpenProcess - MyHook + 5 得到需要跳转到的

位置

(ULONG )(Jmp\_Addr + 1) = (ULONG)inline\_NtOpenProcess - ((ULONG)MyHook + 5);

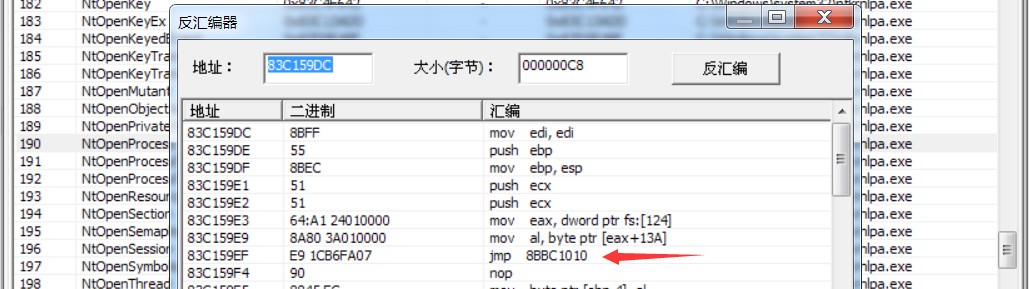
CloseProtect(); //去掉内核页面保护

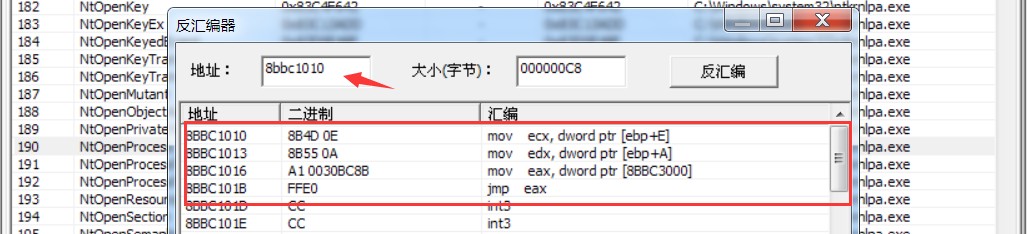
RtlCopyMemory(MyHook, Jmp\_Addr, 6); //将Jmp\_Addr的6字节写入到 MyHook要HOOK的位置StartProtect(); //恢复内核页面保护

DriverObject->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

Hook以后截图如下，可以看到我们能够在自己的函数中为NtOpenProcess函数增加额外的功能，也可以利 用该方法过掉某些游戏的驱动保护，点到为止不说了。





**拓展：还原 Shadow SSDT 中被Hook的函数**

# Shadow SSDT的全称是 Shadow System Services Descriptor Table 影子系统服务描述符表，该表中存放的是一些与系统图形回调队列以及键盘鼠标事件相关的信息。

ServiceDescriptor中只有指向KiServiceTable的SST，是ServiceDescriptorTable是被系统所导出的表结构， 而ServiceDescriptorTableShadow是未导出的，但我们依然可以通过相加偏移的方式得到其当前地址。

在网络游戏中通常会Hook挂钩 NtUserSendInput 这个内核函数，从而实现拦截用户使用能够模拟合成鼠标键盘事件操作的软件 脚本精灵 ，那么该怎末过保护？来直接上车。

通过WinDBG附加内核调试，然后输入以下命令，记得加载符号链接。

// 获取到ShadowSSDT的基址

lkd> dd KeServiceDescriptorTableShadow 8055d6c0 80505570 00000000 0000011c 805059e4

8055d6d0 bf9a1500 00000000 0000029b bf9a2210

// 获取到SSDT表基址

lkd> dd KeServiceDescriptorTable

8055d700 80505570 00000000 0000011c 805059e4

8055d710 00000000 00000000 00000000 00000000

KeServiceDescriptorTable - KeServiceDescriptorTableShadow 相减得到SSDT相对SSSDT的偏移地址此处的便宜地址是0x40，然后直接 dd poi(KeServiceDescriptorTable-0x40) 此处的poi命令为取出后面的内存地址。

lkd> dd KeServiceDescriptorTable - KeServiceDescriptorTableShadow

00000040 ???????? ???????? ???????? ????????

00000050 ???????? ???????? ???????? ????????

lkd> dd poi(KeServiceDescriptorTable-0x40) 80505570 805a5664 805f23ea 805f5c20 805f241c

80505580 805f5c5a 805f2452 805f5c9e 805f5ce2

|  |  |  |
| --- | --- | --- |
| lkd> u 805a5664  nt!NtAcceptConnectPort: |  | // 得到SSDT表中第一个函数的地址 |
| 805a5664 689c000000 | push | 9Ch |
| 805a5669 6850ab4d80 | push | offset nt!\_real+0x118 (804dab50) |
| 805a566e e8cd76f9ff | call | nt!\_SEH\_prolog (8053cd40) |

lkd> Dd poi(KeServiceDescriptorTable-0x40+0x10)

NtGdiAbortDoc

// +0x10得到SSSDT表地址中第一个

bf9a1500 bf93b025 bf94c876 bf88e421 bf9442da

bf9a1510 bf94df11 bf93b2b9 bf93b35e bf839eba

# 上方结果显示

是第一个函数 NtGdiAbortDoc 的地址，加上

# 的序号十进制的

529 转为十六进制是 0x211 ，然后乘以4字节即可获取到

bf93b025

NtUserSendInput

NtUserSendInput

# 家Hook了所以显示的地址是a1ea5e9e 如果管家关闭的话这里就是了。

函数的基址，这里由于电脑管



lkd> dd poi[KeServiceDescriptorTable-0x40+0x10]+0x211 4 bf9a1d44 a1ea5e9e bf86b7d8 bf82938b bf914622

bf9a1d54 bf80e6cb bf8921d4 bf914ae8 bf915076

既然流程都已经清楚了，还原就很简单了，附上汇编代码。



}

DbgPrint("KeServiceDescriptorTable地址为：%x", Shadow\_Address);

DbgPrint("NtUserSendInput地址为：%x", NtUserSendInput);

driver->DriverUnload = DriverUnload; return STATUS\_SUCCESS;

// 系统服务描述表地址给EAX

// EAX-0x40

// eax+0x10

// 取[EAX] eax里面的数据给EAX

// 将取出的数据给变量Shadow\_Address

// 取出的数据给了ECX

// 0x211给EAX，NtUserSendInput的序号

// EAX 4

// ecx+eax

// 取ecx里面的数据给EBX

// EBX给局部变量NtUserSendInput

// 存储ShadowSSDT地址

// 保存NtUserSendInput当前地址

ULONG Shadow\_Address; ULONG NtUserSendInput;

asm

{

push eax push ebx push ecx

mov eax, KeServiceDescriptorTable sub eax, 0x40

add eax, 0x10

mov eax, [eax]

mov Shadow\_Address, eax mov ecx, eax

mov eax, 0x211 imul eax, eax, 4 add ecx, eax mov ebx, [ecx]

mov NtUserSendInput, ebx pop ecx

pop ebx

pop eax

}

extern "C" LONG KeServiceDescriptorTable;

NTSTATUS DriverEntry(PDRIVER\_OBJECT driver, PUNICODE\_STRING reg\_path)

{

恢复代码如下，只附上关键代码吧，其他的和上方基本一致。

ULONG NtUserSendInput\_Now; // 局部变量存放当前地址

ULONG NtUserSendInput\_Ord = 0xFFFFFFFF; // 此处的地址可以用Xuetr直接获取到起源地址

asm //去掉内核页面保护。

{

cli

}

mov eax, cr0

and eax, not 10000h mov cr0, eax

asm

{

}

push eax push ebx push ecx push edx

mov eax, KeServiceDescriptorTable sub eax, 0x40

add eax, 0x10

mov eax, [eax] mov ecx, eax mov eax, 0x211 imul eax, eax, 4 add ecx, eax

mov edx, NtUserSendInput\_Ord // 将函数的原始地址给EDX

mov[ecx], edx // EDX写入到ECX里

mov ebx, [ecx] // 取出NtUserSendInput的地址mov NtUserSendInput\_Now, ebx // 取出NtUserSendInput的起源地址 pop edx

pop ecx pop ebx pop eax

asm //恢复内核页面保护

{

mov eax, cr0

or eax, 10000h mov cr0, eax sti

}

}