# 内核非公开函数调用编号反推

## 相关背景知识

* 1. SSDT

SSDT（System Service Descriptor Table）简单地说是windows 32位操作系统内核函数的地址数组，或者是64位操作系统中相同内核函数的相对偏移量数组（注意：32位系统中直接可以通过该数组找到内核函数的地址了，而64位系统中则是个内核函数偏移，需要换算下才可定位到内核函数地址）。

SSDT是系统服务描述符表内核内存结构的第一个成员，其定义如下：

typedef struct tagSERVICE\_DESCRIPTOR\_TABLE {

SYSTEM\_SERVICE\_TABLE nt; //指向SSDT本身的指针。

SYSTEM\_SERVICE\_TABLE win32k;

SYSTEM\_SERVICE\_TABLE sst3; //指向内存地址的指针，该地址包含表中定义的内核函数个数。

SYSTEM\_SERVICE\_TABLE sst4;

} SERVICE\_DESCRIPTOR\_TABLE;

（SSDT常被病毒和rootkit钩住进而躲避查杀，rootkit希望隐藏文件、注册表项、网络连接等。微软为x64系统引入了PatchGuard，通过BSOD系统来对抗SSDT的修改。）

* 1. ring3与ring0

即用户态（ring3）和内核态（ring0），在Windows操作系统中普通用户程序直接运行在ring3层，常见的模块有GDI32.dll、user32.dll、kernel32.dll，ring3程序通过ntdll.dll过渡，最终达到ring0内核态模块win32k.sys、ntoskrnl.exe。

其中，win32k.sys主要为应用层提供窗口管理和图形设备接口，win32k.sys向内核注册一组调用函数，介入到内核的进程线程运行，是user32.dll、GDI32.dll等用户态组件的内核实现。ntoskrnl.exe提供Microsoft Windows NT内核空间的内核和执行层，并负责各种系统服务，例如硬件抽象，进程和内存管理，因此使其成为系统的基础部分。它包含高速缓存管理器，执行程序，内核，安全性引用监视器，内存管理器和调度程序。

而SSDT就是通往内核态函数的地址表的相关描述表。

* 1. MSR

MSR（Model Specific Register）特定模型寄存器是x86架构中的概念，指的是在x86架构处理器中，一系列用于控制CPU运行、功能开关、调试、跟踪程序执行、监测CPU性能等方面的寄存器。

MSR顾名思义就是Model Specific，即不同的CPU型号或不同的CPU厂商（Intel和AMD都会做x86架构的处理器），它的MSR寄存器可能是不一样的，它会根据具体的CPU型号的变化而变化，每款新的CPU都有可能引入新的MSR寄存器。

对这些寄存器的读写分别由rdmsr和wrmsr指令处理。

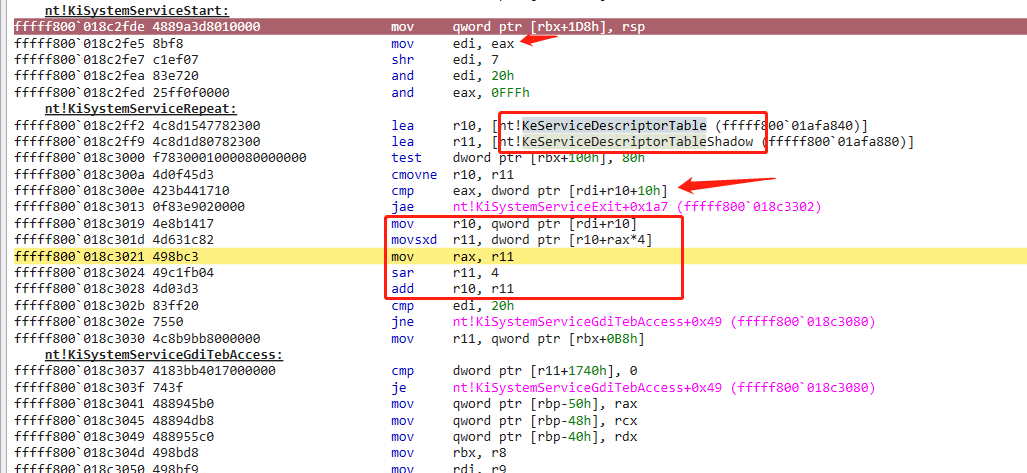
而msr[c0000082]指向系统内核入口nt!KiSystemCall64，即所有用户态程序都要从此进去内核态。在真正进入相关内核函数之前，即nt!KiSystemCall64下方的函数nt!KiSystemServiceRepeat中，会通过查询KeServiceDescriptorTable（SSDT）或KeServiceDescriptorTableShadow（Shadow SSDT）去找到具体内核函数地址。

* 1. KiServiceTable与W32pServiceTable

在KeServiceDescriptorTable和KeServiceDescriptorTableShadow的一定偏移位置存放着系统服务地址表(KiServiceTableh和W32pServiceTable)。其中KeServiceDescriptorTable只有已导出的内核函数KiServiceTable，而KeServiceDescriptorTableShadow中包含未导出内核函数服务地址表KiServiceTableh和W32pServiceTable。因此，我们要调用未导出内核函数时就需要通过KeServiceDescriptorTableShadow来查找。

在漏洞利用领域，攻击者通过直接调用非导出内核函数进行相关漏洞利用。

那具体是查找KiServiceTableh（对应ntoskrnl.exe）还是W32pServiceTable（win32k.sys）呢，我们看下反汇编代码：



我们看到有一个判断来判断传进来的eax即系统调用编号的大小，进而决定最终选择是查找KiServiceTableh还是W32pServiceTable。

决定最终查找哪个表后，接下来就是具体定位操作了。

## 那如何调用非导出内核函数：

**正向调用分析：**

（index 为系统调用的调用号）：

自定义一个函数：

EXTERNDEF g\_NtUserMessageCall\_syscall:dword

g\_NtUserMessageCall\_syscall = 0x1007

UBLIC NtUserMessageCall

NtUserMessageCall PROC

mov r10, rcx ;//rcx储存的是下一条指令的地址，所以windows在Syscall之前会将rcx储存到r10中.

mov eax, g\_NtUserMessageCall\_syscall ;syscall参数

syscall ;syscall根据eax参数的值不同进而执行不同的系统调用。

ret

NtUserMessageCall ENDP

断点运行一阵子后断下：

bp nt!KiSystemServiceStart ".if @eax==0x1007 {} .else {gc}"

内核函数调用绝对地址base = （[(W32pServiceTable地址+4\*(index&0x0FFF;即保留后三位))-->获得内核函数地址偏量） 用最高位（即符号位）补齐至系统机器长度（eg:x64为8字节，而该偏移量是一个dword 4字节的，需要将其用最高位补齐至8自己。这样进行算数位移后才可得到正确结果）>>> 4] +W32pServiceTable地址

//\*4是为index是一个数组编号偏移量，而W32pServiceTable或KiServiceTable都是4个自己长度的，所以\*4转化为地址偏移长度。

具体过程：

（该举例调用的是win32k.sys中的函数，所以使用的是win32k!W32pServiceTable 而非nt!KiServiceTable，实际上而KeServiceDescriptorTableShadow里面这两个表都有，具体使用哪个，是根据传入的eax值(即本例中的 index)进行判断后决定的），具体操作步骤：

步骤1：index & 0x0FFF

0x1007&0x0FFF

步骤2：4\*步骤1结果

步骤3：W32pServiceTable\KiServiceTable + 步骤2结果

步骤4：dd /c 1 步骤三结果 -->获得内核函数地址偏量。

步骤5：内核函数地址偏量 最高位（即符号位）补齐至系统机器长度

步骤6：步骤5结果 >>> 4 即算术右移4位。

步骤7：u W32pServiceTable\KiServiceTable + 步骤6结果 -->即可看到内核函数调用绝对地址即函数开头反汇编代码。

WinDbg中如下:

kd> dd /c 1 W32pServiceTable+4\*(0x1007&0x0FFF) L1

fffff960`00141c1c ffddfec3

ffddfec3 最高位（即符号位）补齐至系统机器长度--》ffffffffffddfec3

1: kd> ? ffffffffffddfec3>>>4

Evaluate expression: -139284 = ffffffff`fffddfec

kd>u fffffffffffddfec+W32pServiceTable L6

win32k!NtUserMessageCall:

fffff960`0011fbec 48895c2408 mov qword ptr [rsp+8],rbx

fffff960`0011fbf1 48896c2410 mov qword ptr [rsp+10h],rbp

fffff960`0011fbf6 4889742418 mov qword ptr [rsp+18h],rsi

fffff960`0011fbfb 57 push rdi

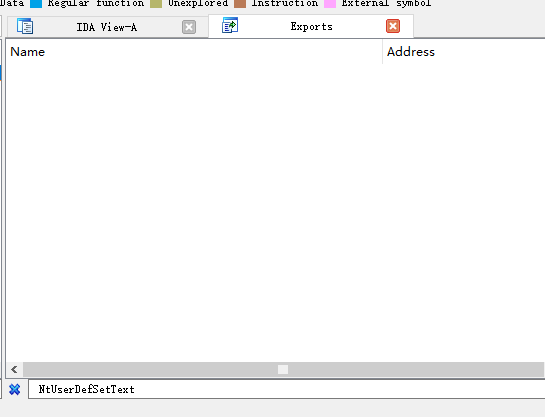
fffff960`0011fbfc 4154 push r12

fffff960`0011fbfe 4155 push r13

注：System系统进程是没有加载ShadowSSDT表的.所以我们必须切换到调用GUI的进程空间(即用户空间进程)中查看.

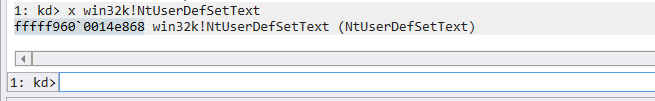
**反推调用编号：**

假如你要调用某个非导出内核函数，那如何确定index值呢，思路跟上边的正向举例相反，比如这个函数：win32k!NtUserDefSetText：



首先我们看到该函数非导出函数。

我们通过windbg看到该函数的入口地址为：fffff960`0014e868



那接下来我们反向操作：

步骤1：win32k!NtUserDefSetText地址-win32k!W32pServiceTable地址

1: kd> ? win32k!NtUserDefSetText-win32k!W32pServiceTable

Evaluate expression: 52328 = 00000000`0000cc68

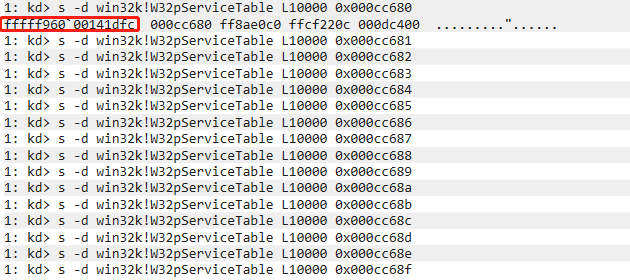
步骤2：步骤1结果左移4位

1: kd> ? 00000000`0000cc68<<4

Evaluate expression: 837248 = 00000000`000cc680

步骤3：步骤2取最后4个字节，即000cc680

步骤4：查找W32pServiceTable中值为fff8dc0\*的地址：



得到该函数偏移量所在的地址为：fffff960`00141dfc

步骤5：步骤4结果-win32k!W32pServiceTable，得到相对偏移地址：

1: kd> ? fffff960`00141dfc-win32k!W32pServiceTable

Evaluate expression: 508 = 00000000`000001fc

步骤6：步骤5结果/4：

1: kd> ? 00000000`000001fc/4

Evaluate expression: 127 = 00000000`0000007f

步骤7：步骤6结果与0x1000进行或运算：

1: kd> ? 00000000`0000007f|0x1000

Evaluate expression: 4223 = 00000000`0000107f

所以最终得到要调用内核未公开函数时需要的调用编号index值为0x107f，即：

Mov eax,0x107f

syscall

## 参考：

海洋太大我太小，欢迎和感谢各位指正错误与不足。

http://www.alonemonkey.com/get-original-ssdt.html

http://www.alonemonkey.com/shadowssdt-explain-in-detail.html

https://www.cnblogs.com/iBinary/p/10990673.html

https://www.ired.team/miscellaneous-reversing-forensics/windows-kernel-internals/glimpse-into-ssdt-in-windows-x64-kernel

https://www.atelierweb.com/the-quest-for-the-ssdts/