注册表回调是一个比较监控注册表读写的回调，它的能量非常大，一个回调能实现在 SSDT 上 HOOK 十几个 API 的效果。部分游戏保护还会在注册表回调上做功夫，监控 service 键的子键，实现双层拦截驱动加载（在映像回调那里还有一层）。而在卡巴斯基等 HIPS 类软件里，则用来监控自启动等键值。

注册表回调在 XP 系统上貌似是一个数组，但是从 WINDOWS 2003 开始，就变成了一个链表。这个链表的头称为 CallbackListHead，可在 CmUnRegisterCallback 中找到：

```
lkd> uf CmUnRegisterCallback
nt!CmUnRegisterCallback:
fffff800`01cba790 48894c2408       mov     qword ptr [rsp+8],rcx
fffff800`01cba795 53               push    rbx
fffff800`01cba796 56               push    rsi
fffff800`01cba797 57               push    rdi
fffff800`01cba798 4154             push    r12
fffff800`01cba79a 4155             push    r13
fffff800`01cba79c 4156             push    r14
fffff800`01cba79e 4157             push    r15
fffff800`01cba7a0 4883ec60         sub     rsp,60h
fffff800`01cba7a4 41bc0d0000c0     mov     r12d,0C000000Dh
fffff800`01cba7aa 4489a424b0000000 mov     dword ptr [rsp+0B0h],r12d
fffff800`01cba7b2 33db             xor     ebx,ebx
fffff800`01cba7b4 48895c2448       mov     qword ptr [rsp+48h],rbx
fffff800`01cba7b9 33c0             xor     eax,eax
fffff800`01cba7bb 4889442450       mov     qword ptr [rsp+50h],rax
fffff800`01cba7c0 4889442458       mov     qword ptr [rsp+58h],rax
fffff800`01cba7c5 448d6b01         lea     r13d,[rbx+1]
fffff800`01cba7c9 458afd           mov     r15b,r13b
fffff800`01cba7cc 4488ac24a8000000 mov     byte ptr [rsp+0A8h],r13b
fffff800`01cba7d4 440f20c7         mov     rdi,cr8
fffff800`01cba7d8 450f22c5         mov     cr8,r13
fffff800`01cba7dc  f00fba351b6adcff00  lock  btr  dword  ptr  [nt!CallbackUnregisterLock
(fffff800`01a81200)],0
fffff800`01cba7e5 720c             jb      nt!CmUnRegisterCallback+0x63 (fffff800`01cba7f3)

//省略中间无关代码

nt!CmUnRegisterCallback+0xc6:
fffff800`01cba856 4533c0           xor     r8d,r8d
fffff800`01cba859 488d542420       lea     rdx,[rsp+20h]
fffff800`01cba85e 488d0d6b69dcff   lea     rcx,[nt!CallbackListHead (fffff800`01a811d0)]
fffff800`01cba865 e8a261e5ff       call    nt!CmListGetNextElement (fffff800`01b10a0c)
fffff800`01cba86a 488bf8           mov     rdi,rax
fffff800`01cba86d 4889442428       mov     qword ptr [rsp+28h],rax
fffff800`01cba872 483bc3           cmp     rax,rbx
```

```
fffff800`01cba875    0f84b8000000            je              nt!CmUnRegisterCallback+0x1a3
(fffff800`01cba933)
```

搜索的过程跟寻找进程、线程、映像的数组类似，根据 lea REG,XXX 来定位。不过为了更加精准，这次我采用两次 lea REG,XXX 来定位。：

```
ULONG64 FindCmpCallbackAfterXP()
{
    ULONG64             uiAddress=0;
    PUCHAR              pCheckArea=NULL, i=0, j=0, StartAddress=0, EndAddress=0;
    ULONG64             dwCheckAddr=0;
    UNICODE_STRING   unstrFunc;
    UCHAR b1=0,b2=0,b3=0;
    ULONG templong=0,QuadPart=0xfffff800;
    RtlInitUnicodeString(&unstrFunc, L"CmUnRegisterCallback");
    pCheckArea = (UCHAR*)MmGetSystemRoutineAddress (&unstrFunc) ;
    if (!pCheckArea)
    {
        KdPrint(("MmGetSystemRoutineAddress failed."));
        return 0;
    }
    StartAddress = (PUCHAR)pCheckArea;
    EndAddress = (PUCHAR)pCheckArea + PAGE_SIZE;
    for(i=StartAddress;i<EndAddress;i++)
    {
        if( MmIsAddressValid(i) && MmIsAddressValid(i+1) && MmIsAddressValid(i+2) )
        {
            b1=*i;
            b2=*(i+1);
            b3=*(i+2);
            if( b1==0x48 && b2==0x8d && b3==0x0d ) //488d0d(lea rcx,)
            {
                j=i-5;
                b1=*j;
                b2=*(j+1);
                b3=*(j+2);
                if( b1==0x48 && b2==0x8d && b3==0x54 ) //488d54(lea rdx,)
                {
                    memcpy(&templong,i+3,4);
                    uiAddress = MakeLong64ByLong32(templong) + (ULONGLONG)i + 7;
                    return uiAddress;
                }
            }
        }
    }
    return 0;
}
```

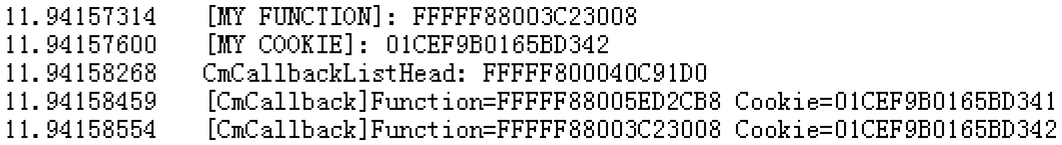
定位完毕之后，就是枚举链表了。注册表回调是一个“结构体链表”，类似于 EPROCESS，它的定义如下：

```
typedef struct _CM_NOTIFY_ENTRY
{
    LIST_ENTRY          ListEntryHead;
    ULONG               UnKnown1;
    ULONG               UnKnown2;
    LARGE_INTEGER       Cookie;
    ULONG64             Context;
    ULONG64             Function;
}CM_NOTIFY_ENTRY, *PCM_NOTIFY_ENTRY;
```

我们只关心两个值，一个是 Cookie，一个是 Function。前者可以理解成注册表回调的“句柄”（用 CmUnRegisterCallback 注销回调传入的就是这个 Cookie），后者是回调函数的地址。代码如下：

```
ULONG CountCmpCallbackAfterXP(ULONG64* pPspLINotifyRoutine)
{
    ULONG                   sum = 0;
    ULONG64                 dwNotifyItemAddr;
    ULONG64*           pNotifyFun;
    ULONG64*           baseNotifyAddr;
    ULONG64                 dwNotifyFun;
    LARGE_INTEGER      cmpCookie;
    PLIST_ENTRY             notifyList;
    PCM_NOTIFY_ENTRY   notify;
    dwNotifyItemAddr = *pPspLINotifyRoutine;
    notifyList = (LIST_ENTRY *)dwNotifyItemAddr;
    do
    {
        notify = (CM_NOTIFY_ENTRY *)notifyList;
        if (MmIsAddressValid(notify))
        {
            if  (MmIsAddressValid((PVOID)(notify->Function))  &&  notify->Function  >
0x8000000000000000)
            {
                DbgPrint("[CmCallback]Function=%p\tCookie=%p",
(PVOID)(notify->Function),(PVOID)(notify->Cookie.QuadPart));
                //notify->Function=(ULONG64)MyRegistryCallback;
                sum ++;
            }
        }
        notifyList = notifyList->Flink;
    }while ( notifyList != ((LIST_ENTRY*)(*pPspLINotifyRoutine)) );
    return sum;
}
```

执行效果类似于：



```
11.94157314    [MY FUNCTION]: FFFFF88003C23008
11.94157600    [MY COOKIE]: 01CEF9B0165BD342
11.94158268    CmCallbackListHead: FFFFF800040C91D0
11.94158459    [CmCallback]Function=FFFFF88005ED2CB8 Cookie=01CEF9B0165BD341
11.94158554    [CmCallback]Function=FFFFF88003C23008 Cookie=01CEF9B0165BD342
```

不过需要注意的是，干净的 WIN7X64 系统是没有注册表回调的。为了体现枚举效果，可以在测试驱动前运行 WIN64AST。对付注册表回调有三种方法（老三套）：1.直接使用 CmUnRegisterCallback 把回调注销；2.把链表中记录的回调地址修改为自定义的空函数的回调地址；3.直接在目标回调地址上写一个 RET，使其不执行任何代码就返回。**第三种方法没有针对性，可以用于对付任何回调函数**。DisableFunctionWithReturnValue 用来对付有返回值的回调函数，DisableFunctionWithoutReturnValue 用于对付无返回值的回调函数。

```
KIRQL WPOFFx64()
{
    KIRQL irql=KeRaiseIrqlToDpcLevel();
    UINT64 cr0=__readcr0();
    cr0 &= 0xfffffffffffeffff;
    __writecr0(cr0);
    _disable();
    return irql;
}

void WPONx64(KIRQL irql)
{
    UINT64 cr0=__readcr0();
    cr0 |= 0x10000;
    _enable();
    __writecr0(cr0);
    KeLowerIrql(irql);
}

VOID DisableFunctionWithReturnValue(PVOID Address)
{
    KIRQL irql;
    CHAR patchCode[] = "\x33\xC0\xC3";    //xor eax,eax + ret
    if(MmIsAddressValid(Address))
    {
        irql=WPOFFx64();
        memcpy(Address,patchCode,3);
        WPONx64(irql);
    }
}

VOID DisableFunctionWithoutReturnValue(PVOID Address)
{
    KIRQL irql;
    if(MmIsAddressValid(Address))
    {
        irql=WPOFFx64();
        RtlFillMemory(Address,1,0xC3);
        WPONx64(irql);
    }
}
```

等大家深入学习之后，会发现以上写两个函数纯属多此一举，在此先卖个关子不解释。