在 WIN32 平台上，监控注册表的手段通常是 SSDT HOOK。不过用 SSDT HOOK 的方式监控注册表实在是太麻烦了，要 HOOK 一大堆函数，还要处理一些 NT6 系统有而 NT5 系统没有的函数。下面我就来介绍一种完胜 SSDT HOOK 监控注册表的方法，效果跟 SSDT HOOK 一样好。这个方法就是使用微软推荐的注册表监控函数：CmRegisterCallbak。此函数其实在 XP 系统上就有了，不过那时功能不完善，只能简单的禁止或允许，无法获得完整的注册表修改信息（即做不到监控）；在 VISTA 以及之后的系统里，微软对此函数做了相当大的改进，使之能获得完整的注册表修改信息。本文最后实现的效果就是：把“注册表编辑器”（regedit.exe）所有对注册表添加、删除、重命名的操作都通过 DbgView 打印出来，并拒绝访问（只针对 regedit.exe 是因为系统对注册表的操作太频繁了，这么做是为了方便大家实验）。

首先看一下这个函数的原型：

```
NTSTATUS CmRegisterCallback
(
  _In_      PEX_CALLBACK_FUNCTION Function,
  _In_opt_  PVOID Context,
  _Out_     PLARGE_INTEGER Cookie
);
```

MSDN 上的解释太书面化了，看得让人不舒服。用大白话说，这三个参数分别为：回调函数的地址，随便设置的值（直接传入 NULL 即可），回调的句柄。相反还有个函数用于销毁回调，它是 CmUnRegisterCallback，原型如下：

```
NTSTATUS CmUnRegisterCallback( _In_ LARGE_INTEGER Cookie);
```

CmUnRegisterCallback 函数唯一的的参数就是 cookie，也就是我所说的“回调的句柄”。创建和销毁回调的代码如下：

```
LARGE_INTEGER CmHandle;
NTSTATUS CmSt;
CmSt=CmRegisterCallback(RegistryCallback,NULL,&CmHandle);
if(NT_SUCCESS(CmSt))
    DbgPrint("CmRegisterCallback SUCCESS!");
else
    DbgPrint("CmRegisterCallback Failed!");
```

```
CmUnRegisterCallback(CmHandle);
```

接下来看看回调函数的原型：

```
NTSTATUS RegistryCallback
(
  _In_      PVOID CallbackContext,
  _In_opt_  PVOID Argument1,     //操作类型（只是操作编号，不是指针）
  _In_opt_  PVOID Argument2    //操作详细信息的结构体指针
)
```

CallbackContext 基本可以忽略，重要的就是下面的两个参数 Argument1 和 Argument2。**Argument1 记录的是操作类型（这个参数不是指针，只是操作类型的编号而已），Argument2 记录的是有关操作信息的结构体指针**。接下来举个例子。比如我们已经注册了一个注册表回调，当有删除注册表项的操作发生时，我们注册的回调就会被调用，Argument1 的信息是 RegNtPreDeleteKey（pre 是“操作前”的意思），Argument2 的信息是一个指向 REG_DELETE_KEY_INFORMATION 结构体的指针。当操作完成后，我们的注册表回调又会被调用一次，此时 Argument1 的信息是 RegNtPostDeleteKey（post 是“操作后”的意思），Argument2 的信息是一个指向 REG_POST_OPERATION_INFORMATION 结构体的指针。在所有的结构体里，有一项是肯定有的，就是 Object，它是你操作了那个项或者根项的对象指针（相对于新建项而言，就是根项的对象指针；相对于新建/设置/删除/重命名键值和删除项而言，就是项的对象指针）。通过这个 Object 获得项名称的代码如下：

```
BOOLEAN GetRegistryObjectCompleteName(PUNICODE_STRING pRegistryPath, PUNICODE_STRING
pPartialRegistryPath, PVOID pRegistryObject)
{
    BOOLEAN foundCompleteName = FALSE;
    BOOLEAN partial = FALSE;
    if((!MmIsAddressValid(pRegistryObject)) || (pRegistryObject == NULL))
        return FALSE;
    /* Check to see if the partial name is really the complete name */
    if(pPartialRegistryPath != NULL)
    {
        if( (((pPartialRegistryPath->Buffer[0] == '\\') || (pPartialRegistryPath->Buffer[0] ==
'%')) ||
                ((pPartialRegistryPath->Buffer[0] == 'T') && (pPartialRegistryPath->Buffer[1]
== 'R') &&
                 (pPartialRegistryPath->Buffer[2] == 'Y') && (pPartialRegistryPath->Buffer[3]
== '\\'))) )
        {
            RtlCopyUnicodeString(pRegistryPath, pPartialRegistryPath);
            partial = TRUE;
            foundCompleteName = TRUE;
        }
    }
    if(!foundCompleteName)
    {
        /* Query the object manager in the kernel for the complete name */
        NTSTATUS status;
        ULONG returnedLength;
        PUNICODE_STRING pObjectName = NULL;
        status = ObQueryNameString(pRegistryObject, (POBJECT_NAME_INFORMATION)pObjectName, 0,
&returnedLength );
        if(status == STATUS_INFO_LENGTH_MISMATCH)
```

```
        {
            pObjectName = ExAllocatePoolWithTag(NonPagedPool, returnedLength,
REGISTRY_POOL_TAG);
            status = ObQueryNameString(pRegistryObject, (POBJECT_NAME_INFORMATION)pObjectName,
returnedLength, &returnedLength );
            if(NT_SUCCESS(status))
            {
                RtlCopyUnicodeString(pRegistryPath, pObjectName);
                foundCompleteName = TRUE;
            }
            ExFreePoolWithTag(pObjectName, REGISTRY_POOL_TAG);
        }
    }
    return foundCompleteName;
}
```

一般来说第二个参数可以忽略，只要第一个和第三个参数就行了。第一个参数是一个**已经分配好了缓冲区的 UNICODE_STRING 结构体指针**，第三个参数就是注册表项的对象指针。除了获得项或根项的路径麻烦一点，其它的信息都放在结构体里了，直接就是 UNICODE_STRING，获取非常方便。以设置键值为例子，REG_SET_VALUE_INFORMATION 结构体的定义如下：

```
typedef struct _REG_SET_VALUE_KEY_INFORMATION
{
  PVOID           Object;
  PUNICODE_STRING ValueName;
  ULONG           TitleIndex;
  ULONG           Type;
  PVOID           Data;
  ULONG           DataSize;
  PVOID           CallContext;
  PVOID           ObjectContext;
  PVOID           Reserved;
} REG_SET_VALUE_KEY_INFORMATION, *PREG_SET_VALUE_KEY_INFORMATION;
```

一般来说，我们只关注此结构体的第 1、2、4、5、6 个成员，1 是这个键所属的项，2 是这个键的名称、4 是这个键的类型、5 是这个键的值、6 是这个键的数据长度。其它结构体里具体每个成员的意义和用途，直接查 MSDN 就行了。回调函数的源码如下（看完上面的解释，看下面的代码应该就没难度了）：

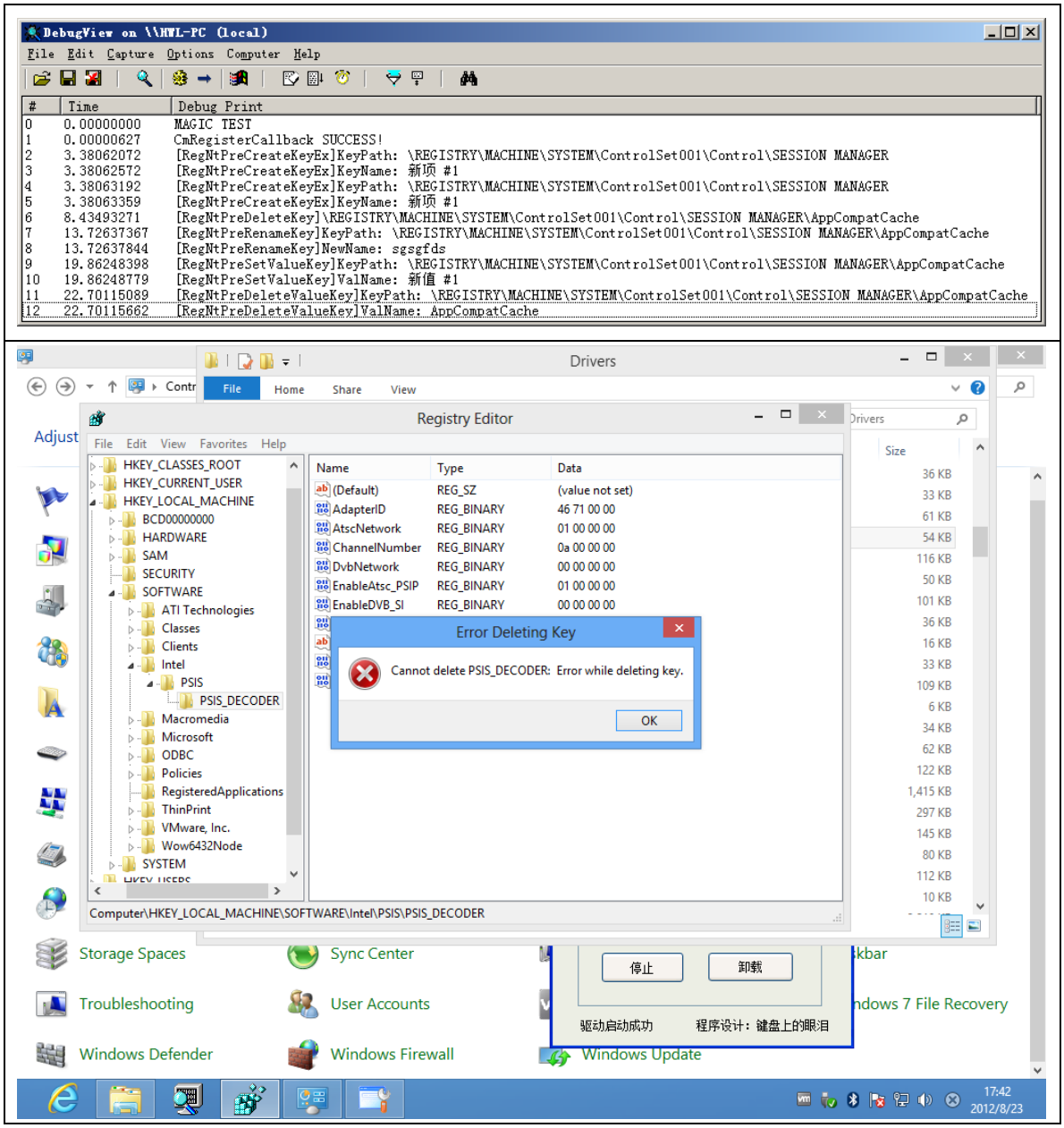
```
NTSTATUS RegistryCallback
(
    IN PVOID CallbackContext,
    IN PVOID Argument1,
    IN PVOID Argument2
```

```
)
{
    long type;
    NTSTATUS CallbackStatus=STATUS_SUCCESS;
    UNICODE_STRING registryPath;
    registryPath.Length = 0;
    registryPath.MaximumLength = 2048 * sizeof(WCHAR);
    registryPath.Buffer = ExAllocatePoolWithTag(NonPagedPool, registryPath.MaximumLength,
REGISTRY_POOL_TAG);
    if(registryPath.Buffer == NULL)
        return STATUS_SUCCESS;
    type = (REG_NOTIFY_CLASS)Argument1;
    switch(type)
    {
        case RegNtPreCreateKeyEx:   //出现两次是因为一次是 OpenKey，一次是 createKey
        {
            if(IsProcessName("regedit.exe",PsGetCurrentProcess()))
            {

    GetRegistryObjectCompleteName(&registryPath,NULL,((PREG_CREATE_KEY_INFORMATION)Argument2)->
RootObject);
                DbgPrint("[RegNtPreCreateKeyEx]KeyPath: %wZ",&registryPath); //新键的路径
                DbgPrint("[RegNtPreCreateKeyEx]KeyName: %wZ",
                        ((PREG_CREATE_KEY_INFORMATION)Argument2)->CompleteName);//新键的名称
                CallbackStatus=STATUS_ACCESS_DENIED;
            }
            break;
        }
        case RegNtPreDeleteKey:
        {
            if(IsProcessName("regedit.exe",PsGetCurrentProcess()))
            {

    GetRegistryObjectCompleteName(&registryPath,NULL,((PREG_DELETE_KEY_INFORMATION)Argument2)->
Object);
                DbgPrint("[RegNtPreDeleteKey]%wZ",&registryPath);                   //新键的路径
                CallbackStatus=STATUS_ACCESS_DENIED;
            }
            break;
        }
        case RegNtPreSetValueKey:
        {
            if(IsProcessName("regedit.exe",PsGetCurrentProcess()))
            {
```

```
    GetRegistryObjectCompleteName(&registryPath,NULL,((PREG_SET_VALUE_KEY_INFORMATION)Argument2
)->Object);
                    DbgPrint("[RegNtPreSetValueKey]KeyPath: %wZ",&registryPath);

    DbgPrint("[RegNtPreSetValueKey]ValName: %wZ",((PREG_SET_VALUE_KEY_INFORMATION)Argument2)->V
alueName);
                    CallbackStatus=STATUS_ACCESS_DENIED;
               }
               break;
          }
          case RegNtPreDeleteValueKey:
          {
               if(IsProcessName("regedit.exe",PsGetCurrentProcess()))
               {

    GetRegistryObjectCompleteName(&registryPath,NULL,((PREG_DELETE_VALUE_KEY_INFORMATION)Argume
nt2)->Object);
                    DbgPrint("[RegNtPreDeleteValueKey]KeyPath: %wZ",&registryPath);

    DbgPrint("[RegNtPreDeleteValueKey]ValName: %wZ",((PREG_DELETE_VALUE_KEY_INFORMATION)Argumen
t2)->ValueName);
                    CallbackStatus=STATUS_ACCESS_DENIED;
               }
               break;
          }
          case RegNtPreRenameKey:
          {
               if(IsProcessName("regedit.exe",PsGetCurrentProcess()))
               {

    GetRegistryObjectCompleteName(&registryPath,NULL,((PREG_RENAME_KEY_INFORMATION)Argument2)->
Object);
                    DbgPrint("[RegNtPreRenameKey]KeyPath: %wZ",&registryPath);

    DbgPrint("[RegNtPreRenameKey]NewName: %wZ",((PREG_RENAME_KEY_INFORMATION)Argument2)->NewNam
e);
                    CallbackStatus=STATUS_ACCESS_DENIED;
               }
               break;
          }
          //『注册表编辑器』里的“重命名键值”是没有直接函数的，是先 SetValueKey 再 DeleteValueKey
          default:
               break;
```

```
    }
    if(registryPath.Buffer != NULL)
        ExFreePoolWithTag(registryPath.Buffer, REGISTRY_POOL_TAG);
    return CallbackStatus;
}
```

需要注意的是，此函数如果返回 STATUS_SUCCESS，注册表操作就会继续执行，如果返回 STATUS_ACCESS_DENIED，注册表操作就不会执行执行了。这样子就达到了“监控”的效果。最终效果如下：



本文到此结束，此代码在 WIN7 X64 和 WIN8 X64 上测试通过（特别提醒：WIN8 对权限的管理非常严格，即使你已经是 Administrator 用户了，运行的程序都是以普通用户的权限运行的。所以在运行任何程序时，都务必对程序按右键，选择“以管理员权限运行”）。

课后作业：完善此监控程序，使之能打印出更加详细的信息。