

在笔者前一篇文章《驱动开发：内核枚举Registry注册表回调》中实现了对注册表的枚举，本章将实现对注册表的监控，不同于32位系统在64位系统中，微软为我们提供了两个针对注册表的专用内核监控函数，通过这两个函数可以在不劫持内核API的前提下实现对注册表增加，删除，创建等事件的有效监控，注册表监视通常会通过 `CmRegisterCallback` 创建监控事件并传入自己的回调函数，与该创建对应的是 `CmUnRegisterCallback` 当注册表监控结束后可用于注销回调。

- `CmRegisterCallback` 设置注册表回调
- `CmUnRegisterCallback` 注销注册表回调

默认情况下 `CmRegisterCallback` 需传入三个参数，参数一回调函数地址，参数二空余，参数三回调句柄，微软定义如下。

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

// 参数1: 回调函数地址
// 参数2: 无作用
// 参数3: 回调句柄
NTSTATUS CmRegisterCallback(
    [in]          PEX_CALLBACK_FUNCTION Function,
    [in, optional] PVOID                Context,
    [out]         PLARGE_INTEGER         Cookie
);
```

自定义注册表回调函数 `MyLySharkCallback` 需要保留三个参数，`CallbackContext` 回调上下文，`Argument1` 是操作类型，`Argument2` 定义详细结构体指针。

```
NTSTATUS MyLySharkCallback(_In_ PVOID CallbackContext, _In_opt_ PVOID Argument1,
    _In_opt_ PVOID Argument2)
```

在自定义回调函数内 `Argument1` 则可获取到操作类型，类型是一个 `REG_NOTIFY_CLASS` 枚举结构，微软对其的具体定义如下所示。

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

typedef enum _REG_NOTIFY_CLASS {
    RegNtDeleteKey,
    RegNtPreDeleteKey = RegNtDeleteKey,
    RegNtSetValueKey,
    RegNtPreSetValueKey = RegNtSetValueKey,
    RegNtDeleteValueKey,
    RegNtPreDeleteValueKey = RegNtDeleteValueKey,
    RegNtSetInformationKey,
    RegNtPreSetInformationKey = RegNtSetInformationKey,
    RegNtRenameKey,
    RegNtPreRenameKey = RegNtRenameKey,
```

```
RegNtEnumerateKey,  
RegNtPreEnumerateKey = RegNtEnumerateKey,  
RegNtEnumerateValueKey,  
RegNtPreEnumerateValueKey = RegNtEnumerateValueKey,  
RegNtQueryKey,  
RegNtPreQueryKey = RegNtQueryKey,  
RegNtQueryValueKey,  
RegNtPreQueryValueKey = RegNtQueryValueKey,  
RegNtQueryMultipleValueKey,  
RegNtPreQueryMultipleValueKey = RegNtQueryMultipleValueKey,  
RegNtPreCreateKey,  
RegNtPostCreateKey,  
RegNtPreOpenKey,  
RegNtPostOpenKey,  
RegNtKeyHandleClose,  
RegNtPreKeyHandleClose = RegNtKeyHandleClose,  
//  
// .Net only  
//  
RegNtPostDeleteKey,  
RegNtPostSetValueKey,  
RegNtPostDeleteValueKey,  
RegNtPostSetInformationKey,  
RegNtPostRenameKey,  
RegNtPostEnumerateKey,  
RegNtPostEnumerateValueKey,  
RegNtPostQueryKey,  
RegNtPostQueryValueKey,  
RegNtPostQueryMultipleValueKey,  
RegNtPostKeyHandleClose,  
RegNtPreCreateKeyEx,  
RegNtPostCreateKeyEx,  
RegNtPreOpenKeyEx,  
RegNtPostOpenKeyEx,  
//  
// new to windows Vista  
//  
RegNtPreFlushKey,  
RegNtPostFlushKey,  
RegNtPreLoadKey,  
RegNtPostLoadKey,  
RegNtPreUnLoadKey,  
RegNtPostUnLoadKey,  
RegNtPreQueryKeySecurity,  
RegNtPostQueryKeySecurity,  
RegNtPreSetKeySecurity,  
RegNtPostSetKeySecurity,  
//  
// per-object context cleanup  
//  
RegNtCallbackObjectContextCleanup,  
//  
// new in Vista SP2  
//  
RegNtPreRestoreKey,
```

```

    RegNtPostRestoreKey,
    RegNtPreSaveKey,
    RegNtPostSaveKey,
    RegNtPreReplaceKey,
    RegNtPostReplaceKey,

    MaxRegNtNotifyClass //should always be the last enum
} REG_NOTIFY_CLASS;

```

其中对于注册表最常用的监控项为以下几种类型，当然为了实现监控则我们必须使用之前，如果使用之后则只能起到监视而无法做到监控的目的。

- RegNtPreCreateKey 创建注册表之前
- RegNtPreOpenKey 打开注册表之前
- RegNtPreDeleteKey 删除注册表之前
- RegNtPreDeleteValueKey 删除键值之前
- RegNtPreSetValueKey 修改注册表之前

如果需要使用监视，首先 `CmRegisterCallback` 注册一个自定义回调，当有消息时则触发

`MyLySharkCallback` 其内部获取到 `lOperateType` 操作类型，并通过 `switch` 选择不同的处理例程，每个处理例程都通过 `GetFullPath` 得到注册表完整路径，并打印出来，这段代码实现如下。

```

// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

#include <ntifs.h>
#include <windef.h>

// 未导出函数声明 pEProcess -> PID
PUCHAR PsGetProcessImageFileName(PEPROCESS pEProcess);

NTSTATUS ObQueryNameString(
    _In_ PVOID Object,
    _Out_writes_bytes_opt_(Length) POBJECT_NAME_INFORMATION ObjectNameInfo,
    _In_ ULONG Length,
    _Out_ PULONG ReturnLength
);

// 注册表回调Cookie
LARGE_INTEGER g_liRegCookie;

// 获取注册表完整路径
BOOLEAN GetFullPath(PUNICODE_STRING pRegistryPath, PVOID pRegistryObject)
{
    // 判断数据地址是否有效
    if ((FALSE == MmIsValidAddress(pRegistryObject)) ||
        (NULL == pRegistryObject))
    {
        return FALSE;
    }

    // 申请内存
    ULONG ulSize = 512;
    PVOID lpObjectNameInfo = ExAllocatePool(NonPagedPool, ulSize);

```

```

    if (NULL == lpObjectNameInfo)
    {
        return FALSE;
    }

    // 获取注册表路径
    ULONG ulRetLen = 0;
    NTSTATUS status = ObQueryNameString(pRegistryObject,
    (POBJECT_NAME_INFORMATION)lpObjectNameInfo, ulSize, &ulRetLen);
    if (!NT_SUCCESS(status))
    {
        ExFreePool(lpObjectNameInfo);
        return FALSE;
    }

    // 复制
    RtlCopyUnicodeString(pRegistryPath, (PUNICODE_STRING)lpObjectNameInfo);
    // 释放内存
    ExFreePool(lpObjectNameInfo);
    return TRUE;
}

// 注册表回调函数
NTSTATUS MyLySharkCallback(_In_ PVOID CallbackContext, _In_opt_ PVOID Argument1,
_In_opt_ PVOID Argument2)
{
    NTSTATUS status = STATUS_SUCCESS;
    UNICODE_STRING ustrRegPath;

    // 获取操作类型
    LONG loperateType = (REG_NOTIFY_CLASS)Argument1;

    // 申请内存
    ustrRegPath.Length = 0;
    ustrRegPath.MaximumLength = 1024 * sizeof(WCHAR);
    ustrRegPath.Buffer = ExAllocatePool(NonPagedPool,
    ustrRegPath.MaximumLength);
    if (NULL == ustrRegPath.Buffer)
    {
        return status;
    }

    RtlZeroMemory(ustrRegPath.Buffer, ustrRegPath.MaximumLength);

    // 判断操作
    switch (loperateType)
    {
        // 创建注册表之前
        case RegNtPreCreateKey:
        {
            // 获取注册表路径
            GetFullPath(&ustrRegPath, ((PREG_CREATE_KEY_INFORMATION)Argument2)-
            >RootObject);
            DbgPrint("[创建注册表] [%wZ] [%wZ]\n", &ustrRegPath,
            ((PREG_CREATE_KEY_INFORMATION)Argument2)->CompleteName);
            break;
        }

        // 打开注册表之前

```

```

case RegNtPreOpenKey:
{
    // 获取注册表路径
    GetFullPath(&ustrRegPath, ((PREG_CREATE_KEY_INFORMATION)Argument2)-
>RootObject);
    DbgPrint("[打开注册表] [%wZ] [%wZ] \n", &ustrRegPath,
((PREG_CREATE_KEY_INFORMATION)Argument2)->CompleteName);
    break;
}
// 删除键之前
case RegNtPreDeleteKey:
{
    // 获取注册表路径
    GetFullPath(&ustrRegPath, ((PREG_DELETE_KEY_INFORMATION)Argument2)-
>Object);
    DbgPrint("[删除键] [%wZ] \n", &ustrRegPath);
    break;
}
// 删除键值之前
case RegNtPreDeleteValueKey:
{
    // 获取注册表路径
    GetFullPath(&ustrRegPath,
((PREG_DELETE_VALUE_KEY_INFORMATION)Argument2)->Object);
    DbgPrint("[删除键值] [%wZ] [%wZ] \n", &ustrRegPath,
((PREG_DELETE_VALUE_KEY_INFORMATION)Argument2)->ValueName);

    // 获取当前进程，即操作注册表的进程
    PEPROCESS pEProcess = PsGetCurrentProcess();
    if (NULL != pEProcess)
    {
        UCHAR *lpszProcessName = PsGetProcessImageFileName(pEProcess);
        if (NULL != lpszProcessName)
        {
            DbgPrint("进程 [%s] 删除了键值对 \n", lpszProcessName);
        }
    }
    break;
}
// 修改键值之前
case RegNtPreSetValueKey:
{
    // 获取注册表路径
    GetFullPath(&ustrRegPath, ((PREG_SET_VALUE_KEY_INFORMATION)Argument2)-
>Object);
    DbgPrint("[修改键值] [%wZ] [%wZ] \n", &ustrRegPath,
((PREG_SET_VALUE_KEY_INFORMATION)Argument2)->ValueName);
    break;
}
default:
    break;
}

// 释放内存
if (NULL != ustrRegPath.Buffer)

```

```

    {
        ExFreePool(ustrRegPath.Buffer);
        ustrRegPath.Buffer = NULL;
    }

    return status;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("Uninstall Driver Is OK \n"));

    // 注销当前注册表回调
    if (0 < g_liRegCookie.QuadPart)
    {
        CmUnRegisterCallback(g_liRegCookie);
    }
}

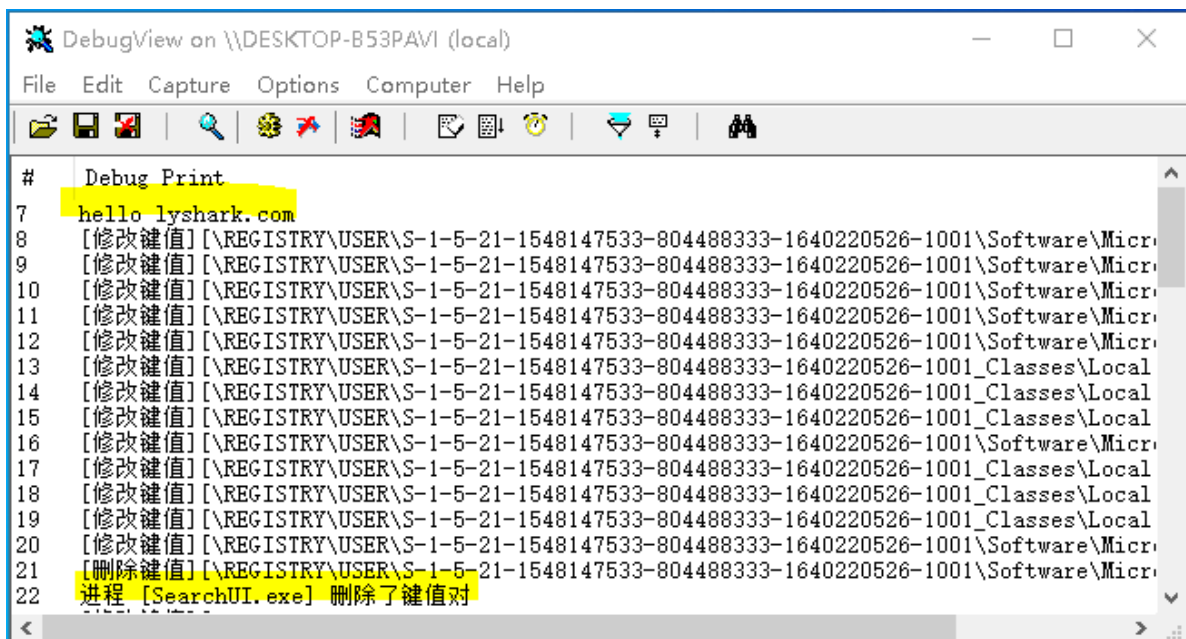
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint(("hello lyshark.com \n"));

    // 设置注册表回调
    NTSTATUS status = CmRegisterCallback(MyLySharkCallback, NULL,
    &g_liRegCookie);
    if (!NT_SUCCESS(status))
    {
        g_liRegCookie.QuadPart = 0;
        return status;
    }

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

运行驱动程序，则会输出当前系统中所有针对注册表的操作，如下图所示。



The screenshot shows a DebugView window titled "DebugView on \\DESKTOP-B53PAVI (local)". The window displays a list of registry operations. The first operation is a "Debug Print" message: "hello lyshark.com". This is followed by a series of "修改键值" (Modify Value) operations for various registry paths, including paths under "Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced" and "Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced". The final operation is a "删除键值" (Delete Value) operation for the path "Registry\CurrentUser\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced". The window also shows a "进程" (Process) column with the value "SearchUI.exe" and a "删除了键值对" (Deleted Value Pair) message.

```

#   Debug Print
7   hello lyshark.com
8   [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Software\Micro
9   [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Software\Micro
10  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Software\Micro
11  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Software\Micro
12  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Software\Micro
13  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Classes\Local
14  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Classes\Local
15  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Classes\Local
16  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Software\Micro
17  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Classes\Local
18  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Classes\Local
19  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Classes\Local
20  [修改键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Software\Micro
21  [删除键值] [\REGISTRY\USER\S-1-5-21-1548147533-804488333-1640220526-1001\Software\Micro
22  进程 [SearchUI.exe] 删除了键值对

```

如上的代码只能实现注册表项的监视，而如果需要监控则需要在回调函数 `MyLySharkCallback` 判断，如果指定注册表项是需要保护的则直接返回 `status = STATUS_ACCESS_DENIED`；从而达到保护注册表的目的，核心代码如下所示。

```
// 反注册表删除回调
NTSTATUS MyLySharkCallback(_In_ PVOID CallbackContext, _In_opt_ PVOID Argument1,
_In_opt_ PVOID Argument2)
{
    NTSTATUS status = STATUS_SUCCESS;
    UNICODE_STRING ustrRegPath;

    // 获取操作类型
    LONG lOperateType = (REG_NOTIFY_CLASS)Argument1;
    ustrRegPath.Length = 0;
    ustrRegPath.MaximumLength = 1024 * sizeof(WCHAR);
    ustrRegPath.Buffer = ExAllocatePool(NonPagedPool,
ustrRegPath.MaximumLength);
    if (NULL == ustrRegPath.Buffer)
    {
        return status;
    }

    RtlZeroMemory(ustrRegPath.Buffer, ustrRegPath.MaximumLength);
    // 判断操作
    switch (lOperateType)
    {
        // 删除键值之前
        case RegNtPreDeleteValueKey:
        {
            // 获取注册表路径
            GetFullPath(&ustrRegPath,
((PREG_DELETE_VALUE_KEY_INFORMATION)Argument2)->Object);
            DbgPrint("[删除键值] [%wZ] [%wZ] \n", &ustrRegPath,
((PREG_DELETE_VALUE_KEY_INFORMATION)Argument2)->ValueName);

            // 如果要删除指定注册表项则拒绝
            PWCH pszRegister = L"\\REGISTRY\\MACHINE\\SOFTWARE\\lyshark.com";
            if (wcscmp(ustrRegPath.Buffer, pszRegister) == 0)
            {
                DbgPrint("[lyshark] 注册表项删除操作已被拦截! \n");
                // 拒绝操作
                status = STATUS_ACCESS_DENIED;
            }
            break;
        }
        default:
            break;
    }

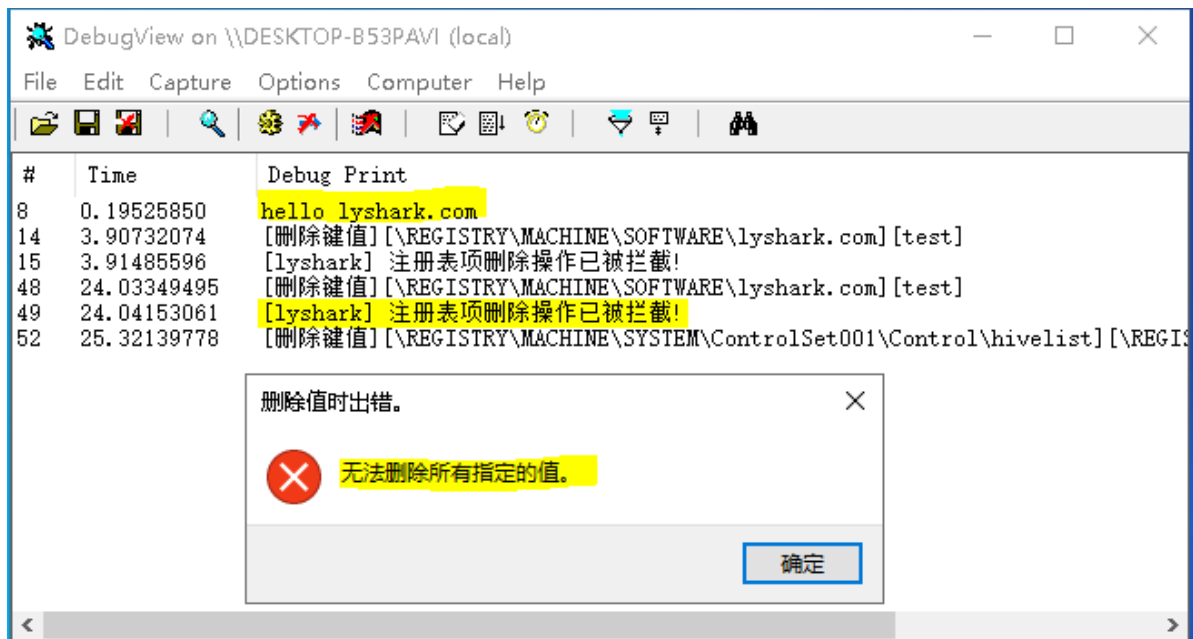
    // 释放内存
    if (NULL != ustrRegPath.Buffer)
    {
        ExFreePool(ustrRegPath.Buffer);
        ustrRegPath.Buffer = NULL;
    }
}
```

```

    return status;
}

```

运行驱动程序，然后我们尝试删除 \\LyShark\\HKEY\_LOCAL\_MACHINE\\SOFTWARE\\lyshark.com 里面的子项，则会提示如下信息。



当然这里的 `RegNtPreDeleteValueKey` 是指的删除操作，如果将其替换成 `RegNtPreSetValueKey`，那么只有当注册表被创建才会拦截，此时就会变成拦截创建。

```

// 拦截创建操作
NTSTATUS MyLySharkCallback(_In_ PVOID CallbackContext, _In_opt_ PVOID Argument1,
_In_opt_ PVOID Argument2)
{
    NTSTATUS status = STATUS_SUCCESS;
    UNICODE_STRING ustrRegPath;

    // 获取操作类型
    LONG loperateType = (REG_NOTIFY_CLASS)Argument1;

    // 申请内存
    ustrRegPath.Length = 0;
    ustrRegPath.MaximumLength = 1024 * sizeof(WCHAR);
    ustrRegPath.Buffer = ExAllocatePool(NonPagedPool,
    ustrRegPath.MaximumLength);
    if (NULL == ustrRegPath.Buffer)
    {
        return status;
    }
    RtlZeroMemory(ustrRegPath.Buffer, ustrRegPath.MaximumLength);

    // 判断操作
    switch (loperateType)
    {
        // 修改键值之前
        case RegNtPreSetValueKey:
        {

```



```

// 获取注册表路径
GetFullPath(&ustrRegPath,
((PREG_DELETE_VALUE_KEY_INFORMATION)Argument2)->Object);

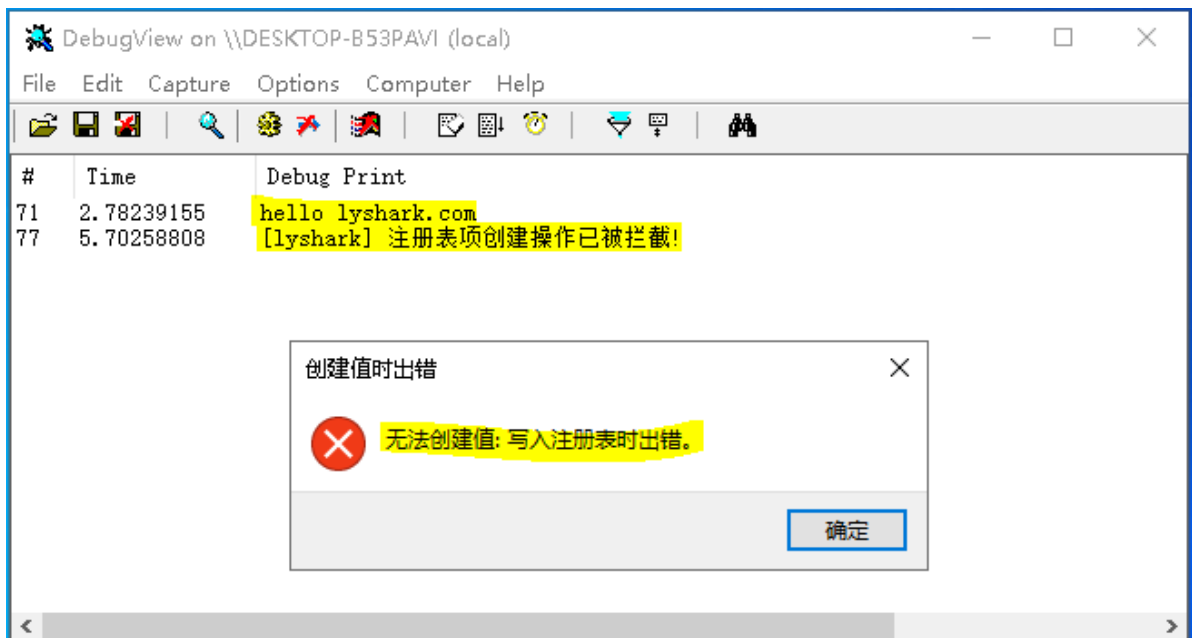
// 拦截创建
PWCH pszRegister = L"\\REGISTRY\\MACHINE\\SOFTWARE\\lyshark.com";
if (wcscmp(ustrRegPath.Buffer, pszRegister) == 0)
{
    DbgPrint("[lyshark] 注册表项创建操作已被拦截! \n");
    status = STATUS_ACCESS_DENIED;
}
break;
default:
    break;
}

// 释放内存
if (NULL != ustrRegPath.Buffer)
{
    ExFreePool(ustrRegPath.Buffer);
    ustrRegPath.Buffer = NULL;
}

return status;
}

```

加载驱动保护，然后我们尝试在 \\LyShark\\HKEY\_LOCAL\_MACHINE\\SOFTWARE\\lyshark.com 里面创建一个子项，则会提示创建失败。



作者：王瑞 (LyShark)

作者邮箱：[me@lyshark.com](mailto:me@lyshark.com)

版权声明：本博客文章与代码均为学习时整理的笔记，文章 [均为原创] 作品，转载文章请遵守《中华人民共和国著作权法》相关法律规定或遵守《署名CC BY-ND 4.0国际》规范，合理合规携带原创出处转载，如果不携带文章出处，并恶意转载多篇原创文章被本人发现，本人保留起诉权！