**软件安全课程设计**

**基于API HOOK的软件行为分析系统**

**一、课程设计题目**

基于API HOOK的软件行为分析系统

**二、课程编码：130681**

课程性质：必修

**三、学时学分**

学时：2周

学分：1学分

**四、先修课程**

C语言、汇编语言、数据结构、软件安全

**五、课程设计任务与要求**

对于无源码情况下分析样本程序的行为，有多种方法。本次课程设计是利用Detours开源项目包提供的接口，完成基本的程序行为分析。具体课程设计任务见表1。任务主要分为API调用截获及分析两大部分。任务可以自己选择Windows（含Win7/win8/win10）平台上实现,编程用语言为C或者C++。其中使用的Windows平台使用微软的Detours开源库，可以在VS2019环境下编译后使用，后面会有详细介绍。

平台：Win7/Win8/Win10,VC++(VS2013/VS2015/VS2019)。

**表1 课程设计任务表**

|  |  |  |
| --- | --- | --- |
| 序号 | 任务 | 要求 |
| 1 | 实现基本的第三方进程WindowsAPI截获框架 | 框架包括1.1编译生成Detours库；1.2完成挂钩框架DLL，实现对MessageBox调用截获，能打印出调用的参数、进程名称以及进程Exe文件信息；1.3自编或者利用已有恶意代码样例（包含弹出对话框动作）；1.4完成注入动作开启和关闭的“注射器”控制程序 |
| 2 | 实现堆操作API截获 | 修改1.2-1.4，实现堆操作（创建，释放）API进行截获，打印出所有参数信息。 |
| 3 | 实现文件操作API截获 | 实现对文件操作（创建，关闭，读写）API进行截获，打印出所有参数信息。 |
| 4 | 注册表操作API截获 | 实现对注册表操作（创建，关闭，读写）API进行截获，打印出所有参数信息。 |
| 5 | 堆操作异常行为分析 | 设计并完成算法，记录并给出提示：  1.检测堆申请与释放是否一致（正常）；  2.是否发生重复的多次释放（异常） |
| 6 | 文件操作异常行为分析 | 设计并完成算法，记录并给出提示：   1. 判断操作范围是否有多个文件夹； 2. 是否存在自我复制的情况； 3. 是否修改了其它可执行代码包括exe，dll，ocx等； 4. 是否将文件内容读取后发送到网络（选做）； |
| 7 | 注册表操作异常行为分析 | 设计并完成算法，记录并给出提示：   1. 判断是否新增注册表项并判断是否为自启动执行文件项； 2. 是否修改了注册表； 3. 输出所有的注册表操作项； |
| 8 | 提供系统界面 | 所设计实现的功能，有图形界面展示 |
| 9 | 行为检测样本库 | 提供5个待检测的可能存在恶意的Exe样本； |
| 10 | 网络通信操作异常行为分析（选做） | 设计并完成算法，记录并给出提示：   1. 实现对网络传输SOCKET操作（连接、发送与接收）API的截获； 2. 打印进程连接端口、协议类型、IP信息 3. HTTP连接协议的解析，判断传输的内容是否为明文 |
| 11 | 内存拷贝监测与关联分析（选做） | 设计并完成算法，记录并给出提示：  能够输出内存拷贝信息，并分析拷贝的内容流向。 |

**六、关键技术原理介绍**

本课程设计采用微软的开源工具库Detours实现Windows API 截获（或称为绕道、挂钩）操作。Detours 是一个在x86平台上截获任意Win32函数调用的工具库。中断代码可以在运行时动态加载，也可以在静态执行文件处理。

## Detours库原理

修改目标函数（一般为Windows API接口函数）：使用一个无条件转移指令来替换该目标函数的首部几条指令，将控制流直接转移到一个用户自己实现的截获函数（即Detour函数）。而原目标函数中被替换的指令被保存在一个被称为“Trampoline”函数中（译注：英文意为蹦床函数）。这些指令包括目标函数中被替换的代码以及一个重新跳转到目标函数正确位置的无条件分支。截获函数可以替换目标函数，或者通过执行“Trampoline”函数时，将目标函数作为子程序来调用的办法，在保留原来目标函数功能基础上，来完成扩展功能。

因为截获函数（Detour函数）是执行时被插入到内存中目标函数的代码里，不是在硬盘上直接修改目标函数，所以，可以在一个很好的粒度上使得截获二进制函数的执行变得更容易。例如，一个应用程序执行时加载的 DLL中的函数过程，可以被插入一段截获代码（detoured），与此同时，这个DLL还可以被其他应用程序按正常情况执行（译注：也就是按照不被截获的方式执行，因为DLL二进制文件没有被修改，所以发生截获时不会影响其他进程空间加载这个DLL）。不同于DLL的重新链接或者静态重定向方式， Detours库中使用的这种中断技术，确保不会影响到应用程序中的方法或者系统代码对目标函数的定位。

如果其他人为了调试或者在内部使用其他系统检测手段而试图修改二进制代码，Detours将是一个可以普遍使用的开发包。Detours是第一个可以在任意平台上将未修改的目标代码作为一个可以通过“trampoline”调用的子程序来保留的开发包。而以前的系统，在逻辑上预先将截获代码放到目标代码中，而不是将原始的目标代码作为一个普通的子程序来调用。独特的“trampoline”设计对于扩展现有的软件的二进制代码是至关重要的。

出于使用基本的函数截获功能的目的， Detours同样提供了编辑任何DLL导入表的功能，达到向已存在的二进制代码中添加任意数据节表的目的，向一个新进程或者一个已经运行着的进程中注入一个DLL。一旦向一个进程注入了DLL，这个动态库就可以截获任何Win32函数，不论它是在应用程序中或者在系统库中。

## (1) WIN32进程的内存管理

WINDOWS 实现了虚拟存储器，每一WIN32进程拥有4GB的虚存空间， 关于WIN32进程的虚存结构及其操作的具体细节请参阅WIN32 API手册， 以下仅指出与Detours相关的几点：

1) 进程要执行的指令也放在虚存空间中；

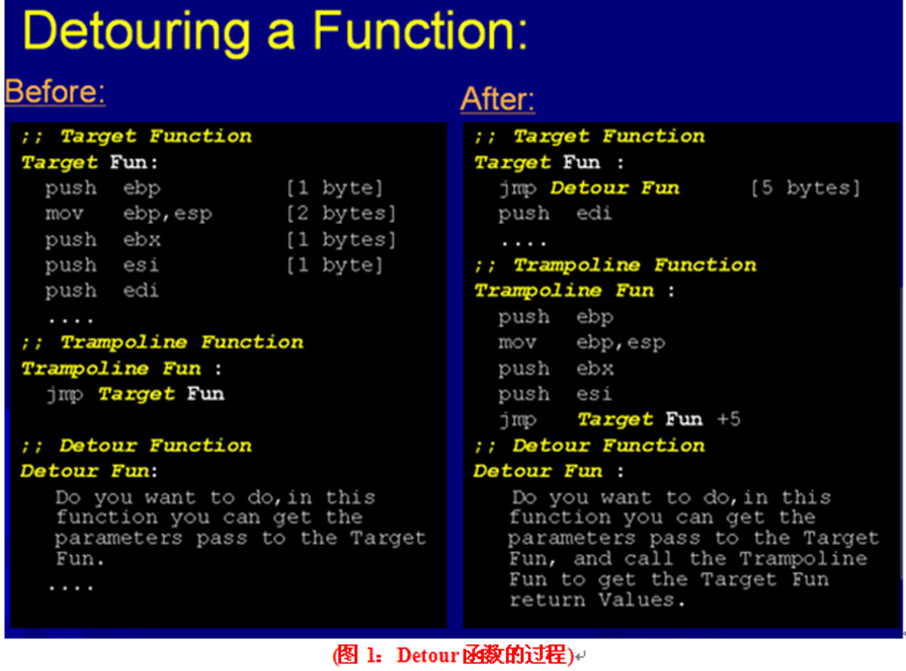
2) 可以使用QueryProtectEx函数把存放指令的页面的权限更改为可读、可写、可执行，再改写其内容，从而修改正在运行的程序

3) 可以使用VirtualAllocEx从一个进程为另一正运行的进程分配虚存，再使用 QueryProtectEx函数把页面的权限更改为可读可写可执行，并把要执行的指令以二进制机器码的形式写入，从而为一个正在运行的进程注入任意的代码 。

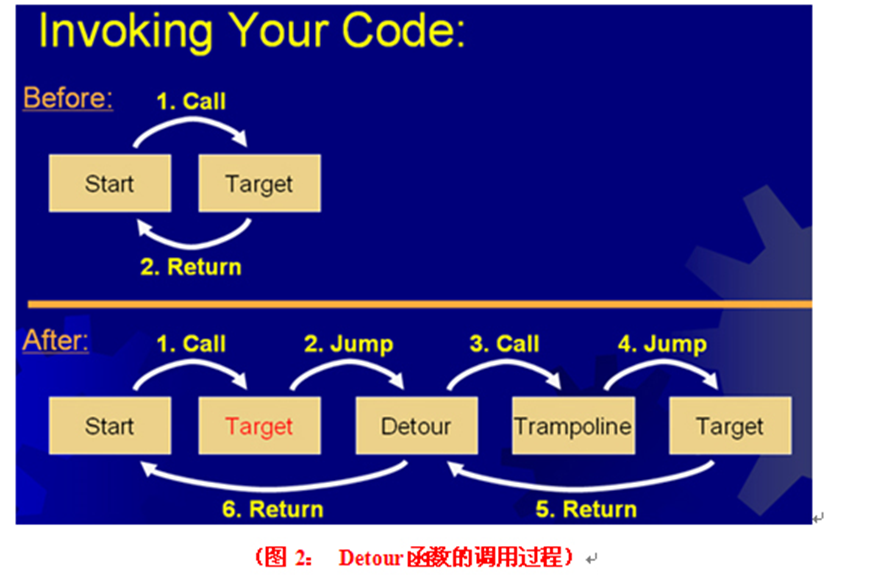
## (2)拦截WIN32 API的原理

Detours定义了三个概念：

1) Target函数（Target Fun）：要拦截的函数，通常为Windows的API；

2) Trampoline函数(Trampoline Fun)：Target函数的部分复制品。因为Detours将会改写Target函数，所以先把Target函数的前5个字节复制保存好，一方面仍然保存Target函数的过程调用语义，另一方面便于以后的恢复。

3) Detour 函数(Detour Fun)：用来替代Target函数的函数。

Detours 在Target函数的开头加入**jmp Address\_of\_ Detour\_ Function**指令（共5个字节）,把对Target函数的调用引导到自己的Detour函数， 把Target函数的开头的5个字节(**push ebp……push esi**)以及**jmp  Address\_of\_ Target \_ Function+5**(共10个字节)作为Trampoline函数的内容。请参考图1和图2。

如果未看懂以上图示，可以再看一下对三个函数的说明：

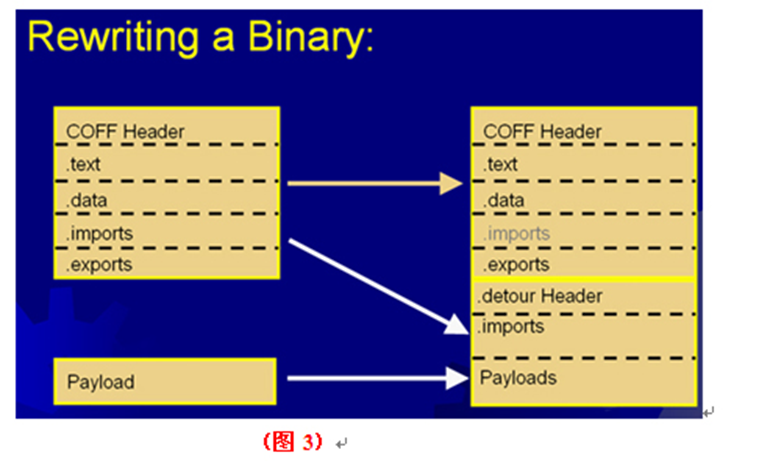
**Target函数：**原来的被调用目标函数的函数体（二进制）需要至少有5个字节以上，才能被改写为Target函数。理由如下：改写后的Target函数，第一条是一个跳转到Detour函数的语句，需要覆盖5个字节，之后才是原始的被调用函数剩余的未被覆盖的内容（例如图1中的push edi是第6个字节的指令，将从Trampoline函数跳回该处）。同时，微软的说明文档中，Trampoline函数的函数体为：从原来被调用函数拷贝被覆盖的这前5个字节，再加一个无条件跳转指令，就完成了蹦床函数。因此，前5个字节必须是完整指令（当第5个字节和第6个字节不能是一条不可分割的指令时，会导致Trampoline函数执行错误，复制前5个字节后，一条完整的指令被硬性分割开来，造成程序崩溃）。

如果第5字节和第6个字节是不可分割指令，需要调整拷贝到Trampoline函数的字节个数（具体拷贝多少，可以查看目标函数的汇编代码得到）。此函数是目标函数的修改版本，不能在Detour函数中直接调用，需要通过对Trampoline函数的调用来达到间接调用。

**Trampoline函数：**此函数默认分配了32个字节，其函数的内容，就是拷贝的目标函数的前5个字节，加上一个JMP Address\_of\_ Target \_ Function+5指令,共10个字节。

此函数仅供Detour函数调用，执行完前5个字节的指令后，再绝对跳转到修改后的目标函数（Target 函数）的第6个字节，继续执行原目标函数功能。

**Detour函数：**（Detour，英文中指绕道，类似于2号食堂二楼对着明德楼的门坏了需要维修，就在此处设置一个detour的指示牌，指引大家从食堂外的楼梯绕道，从食堂一楼正门进去，之后，再从该楼梯返回到二楼）此函数是用户需要的截获API的一个模拟版本，调用方式，参数个数必须和目标函数相一致。如目标函数是\_\_stdcall，则Detour函数声明也必须是 \_\_stdcall，参数个数和类型也必须相同，否则会造成程序崩溃。此函数在程序调用目标函数的第一条指令的时候就会被调用（无条件跳转过来的），如果在此函数中想继续调用原来的目标函数，必须调用Trampoline函数（Trampoline函数在执行完目标函数的前5个字节的指令后会无条件跳转到目标函数的5个字节后继续执行），不能再直接调用目标函数，否则将进入无穷递归（目标函数跳转到Detour函数，Detour函数又跳转到目标函数的递归，因为目标函数在内存中的前5个字节已经被修改成绝对跳转）。通过对Trampoline函数的调用后，可以获取目标函数的执行结果，此特性对分析目标函数非常有用，而且可以将目标函数的输出结果进行修改后再传回给应用程序。

Detour提供了向运行中的应用程序注入Detour函数和在二进制文件基础上注入Detour函数两种方式。本章主要讨论第二种工作方式。通过Detours提供的开发包可以在二进制EXE文件中添加一个名称为 Detour的节表，如下图3所示，主要目的是实现PE加载器加载应用程序的时候会自动加载您编写的Detours DLL，在Detours DLL中的DLLMain中完成对目标函数的Detour。

## (3) Detours提供的截获API的相关接口

Detours提供的API 接口可以作为一个共享DLL给外部程序调用，也可以作为一个静态Lib链接到您的程序内部。

Trampoline函数可以动态或者静态的创建，如果目标函数本身是一个链接符号，使用静态的trampoline函数将非常简单。如果目标函数不能在链接时可见，那么可以使用动态trampoline函数。

要使用静态的trampoline函数来截获目标函数，应用程序生成trampoline的时候必须使用DETOUR\_TRAMPOLINE宏。DETOUR\_TRAMPOLINE有两个输入参数：trampoline的原型和目标函数的名字。

注意，对于正确的截获模型，包括目标函数，trampoline函数，以及截获函数都必须是完全一致的调用形式，包括参数格式和调用约定。当通过 trampoline函数调用目标函数的时候拷贝正确参数是截获函数的责任。由于目标函数仅仅是截获函数的一个可调用分支（截获函数可以调用 trampoline函数也可以不调用），这种责任是一种强制性要求。

使用相同的调用约定可以确保寄存器中的值被正确的保存，并且保证调用堆栈在截获函数调用目标函数的时候能正确的建立和销毁。

可以使用DetourFunctionWithTrampoline函数来截获目标函数。这个函数有两个参数：trampoline函数以及截获函数的指针。因为目标函数已经被加到trampoline函数中，所有不需要在参数中特别指定。

我们可以使用DetourFunction函数来创建一个动态的trampoline函数，它包括两个参数：一个指向目标函数的指针和一个截获函数的指针。DetourFunction分配一个新的trampoline函数并将适当的截获代码插入到目标函数中去。

当目标函数不是很容易使用的时候，DetourFindFunction函数可以找到那个函数，不管它是DLL中导出的函数，或者是可以通过二进制目标函数的调试符号找到。

DetourFindFunction 接受两个参数：库的名字和函数的名字。如果DetourFindFunction函数找到了指定的函数，返回该函数的指针，否则将返回一个NULL指针。 DetourFindFunction会首先使用Win32函数LoadLibrary 和GetProcAddress来定位函数，如果函数没有在 DLL的导出表中找到，DetourFindFunction将使用ImageHlp库来搜索有效的调试符号（译注：这里的调试符号是指Windows本身提供的调试符号，需要单独安装，具体信息请参考Windows的用户诊断支持信息）。DetourFindFunction返回的函数指针可以用来传递给DetourFunction以生成一个动态的trampoline函数。

我们可以调用DetourRemoveTrampoline来去掉对一个目标函数的截获。

注意，因为Detours中的函数会修改应用程序的地址空间，请确保当加入截获函数或者去掉截获函数的时候没有其他线程在进程空间中执行，这是程序员的责任。一个简单的方法保证这个时候是单线程执行就是在加载Detours库的时候在DllMain中呼叫函数。

## (4)使用Detours实现对API的截获的两种方法

**1) 静态方法**

建立一个Dll工程，名称为ApiHook，这里以VS开发环境，以截获ASCII版本的MessageBoxA函数来说明。在Dll的工程加入：

DETOUR\_TRAMPOLINE(int WINAPI Real\_Messagebox(

HWND hWnd ,

LPCSTR lpText,

LPCSTR lpCaption,UINT uType), ::MessageBoxA);

生成一个静态的MessageBoxA的Trampoline函数，在Dll工程中加入目标函数的Detour函数：

int WINAPI MessageBox\_Mine(

HWND hWnd ,

LPCSTR lpText,

LPCSTR lpCaption,

UINT uType)  
{  
  CString tmp= lpText;  
  tmp+=” 被Detour截获”;  
  return Real\_Messagebox(hWnd,tmp,lpCaption,uType);  
//  return ::MessageBoxA(hWnd,tmp,lpCaption,uType);  //Error   
}

在Dll入口函数中的加载Dll事件中加入：

DetourFunctionWithTrampoline((PBYTE)Real\_Messagebox, (PBYTE)MessageBox\_Mine);

在Dll入口函数中的卸载Dll事件中加入：

DetourRemove((PBYTE)Real\_Messagebox, (PBYTE)MessageBox\_Mine);  
**2)动态方法**

建立一个Dll工程，名称为ApiHook，这里以Visual C++6.0开发环境，以截获ASCII版本的MessageBoxA函数来说明。在Dll的工程加入：

//声明MessageBoxA一样的函数原型

typedef int  (WINAPI \* MessageBoxSys)(

HWND hWnd ,

LPCSTR lpText,

LPCSTR lpCaption,

UINT uType);  
//目标函数指针  
MessageBoxSys SystemMessageBox=NULL;  
//Trampoline函数指针  
MessageBoxSys Real\_MessageBox=NULL;  
在Dll工程中加入目标函数的Detour函数：  
int WINAPI MessageBox\_Mine( HWND hWnd ,  
    LPCSTR lpText,  
    LPCSTR lpCaption,  
    UINT uType)  
{  
  CString tmp= lpText;  
  tmp+=” 被Detour截获”;  
  return Real\_Messagebox(hWnd,tmp,lpCaption,uType);  
//  return ::MessageBoxA(hWnd,tmp,lpCaption,uType);  //Error   
}  
在Dll入口函数中的**加载Dll事件**中加入：  
  SystemMessageBox=(MessageBoxSys)DetourFindFunction("user32.dll","MessageBoxA");  
  if(SystemMessageBox==NULL)  
  {  
    return FASLE;  
  }  
  Real\_MessageBox=(MessageBoxSys)DetourFunction((PBYTE)SystemMessageBox, (PBYTE)MessageBox\_Mine);  
在Dll入口函数中的**卸载Dll事件**中加入：

DetourRemove((PBYTE)Real\_Messagebox, (PBYTE)MessageBox\_Mine);

## (5)关于截获第三方应用的API

前面所述办法只能截获进程自身的API，如果要截获第三方进程调用API，需要将Dll代码注入到第三方进程空间，往往采用全局消息Hook或者远程线程创建的方式进行。

**七、Win10环境中Detours API截获参考例子**

(1)准备工作：使用VS2019编译Detours

在[Detours-github](https://github.com/microsoft/detours)上下载Detours的源码，解压得到文件夹Detours-master，然后，在开始菜单中找到x64 Native Tools Command Prompt for VS 2019 和 x86 Native Tools Command Prompt for VS 2019，这两个可以分别用来编译64位和32位的Detours。如图4所示。

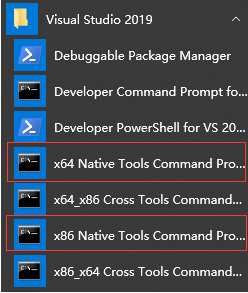
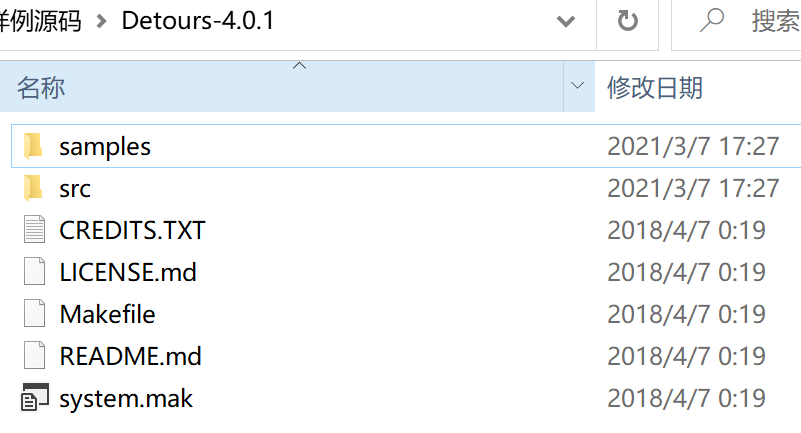


图4 源码及编译Detours的工具

定位路径到解压的文件夹的src目录下，然后编译：

cd Detours-master/src

nmake /f Makefile

编译之后，可以在根目录找到bin.X64、lib.X64、include这三个文件夹，同理如果使用x86 Native Tools Command Prompt for VS 2019的话，生成bin.X86、lib.X86、include这三个文件夹。如图5所示。

至此，编译结束。



图5 编译后生成的目录结构

(2) 准备工作：在VS上配置Detours库（使用Detours，采用dll注入的方式进行Hook）

在VS创建DLL项目，调试---调试属性。如图6所示。

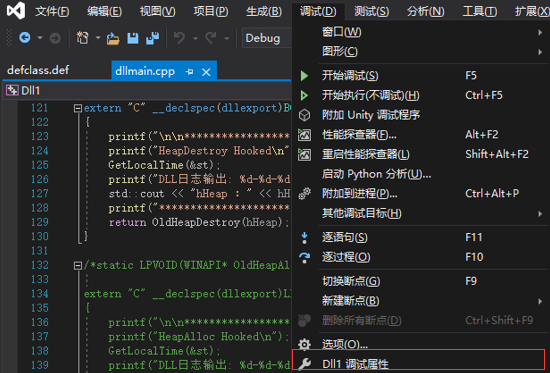


图6 VS 2019中配置包含目录及库目录-1

配置属性中选择VC++目录，在**包含目录**中加上刚才编译出的**include文件夹路径**，在**库目录**上加上**lib.X86**（64位系统则对应lib.X64）。如图7所示。

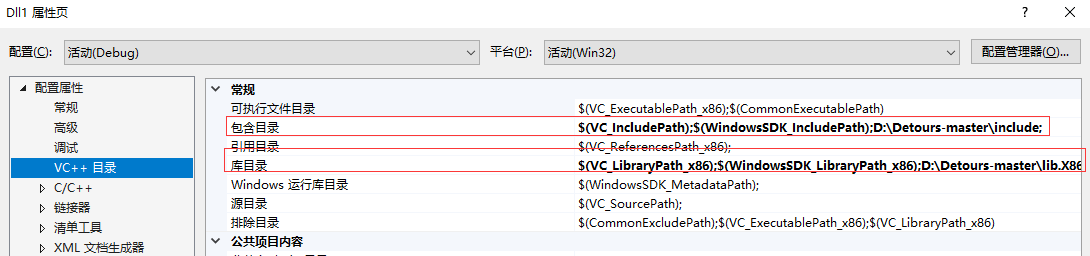


图7 VS 2019中配置包含目录及库目录-2

**(3)准备DLL**

新建DLL项目（若需要Hook的API为 VOID WINDOWSAPI(VOID p1,VOID p2,VOID p3);

**1）定义和引入需要挂钩（Hook）的函数和替换的函数**

static VOID (WINAPI\* OLD\_WINDOWSAPI)(VOID p1,VOID p2,VOID p3) = WINDOWSAPI;

extern "C" \_\_declspec(dllexport) VOID NEW\_WINDOWSAPI(VOID p1,VOID p2,VOID p3)

{

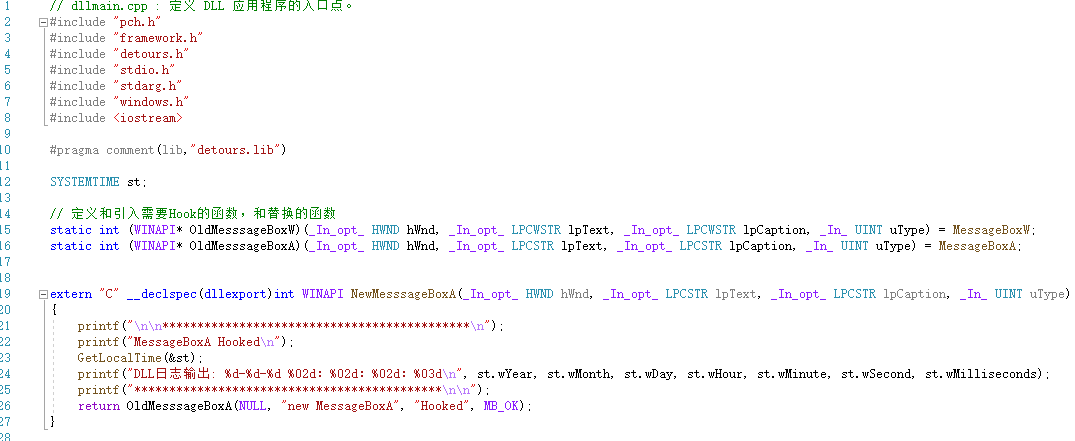
//TODO:

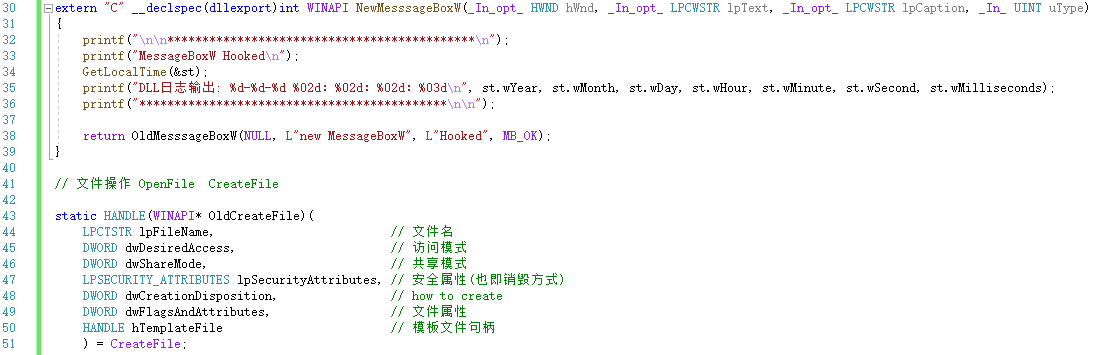
return OLD\_WINDOWSAPI(p1,p2,p3);

}

**2）DLL主体实现参考代码**

参考代码如图，阅读可以发现哪些函数被挂钩，请结合课程设计的要求，挂钩对应的函数。





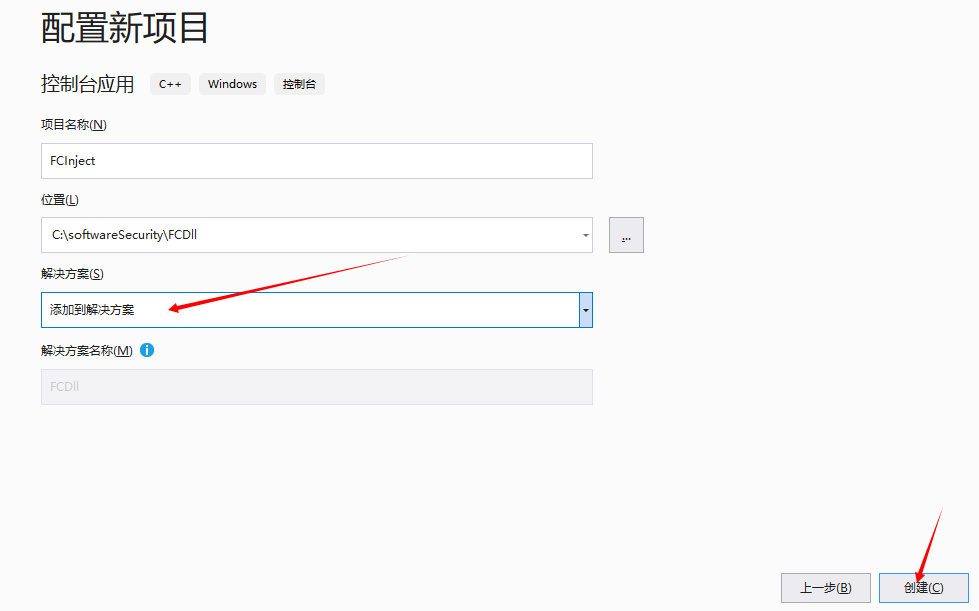






(4)创建待注入的客户端程序

为了测试前面的挂钩是否生效，可以创建一个简单的WPF程序，程序名为app.exe（该程序名会在后面的“注射器“中使用），并增加一个按钮，弹出一个对话框（这样就调用了前面DLL中挂钩的函数MessageBox）。代码略去。



**（5）完成开启和关闭注入的工具（注射器）**

新建一个控制台程序，使用DetourCreateProcessWithDllEx将DLL注入目标客户端程序，代码如下。样例代码只会检测给定的目标程序，如果需要监测所有应用，则可以采用全局钩子的形式SetWindowsHookEx。本指导书给出截获基本的API实例代码，其它API可以参照样例，由同学们自己实现。

主文件实现：

