1 jmp addresss(五字节） jmp后的偏移量=目标地址-原地址-JCC长度

2push address/retn（六字节）

3 mov eax/jmp eax（七字节）

4 call HookProc

5HotPatch Hook

流程

1构造跳转指令

2在内存中寻找需要HOOK函数的地址，并且保存前五个字节

3将构造的跳转指令写入要HOOK函数的位置处

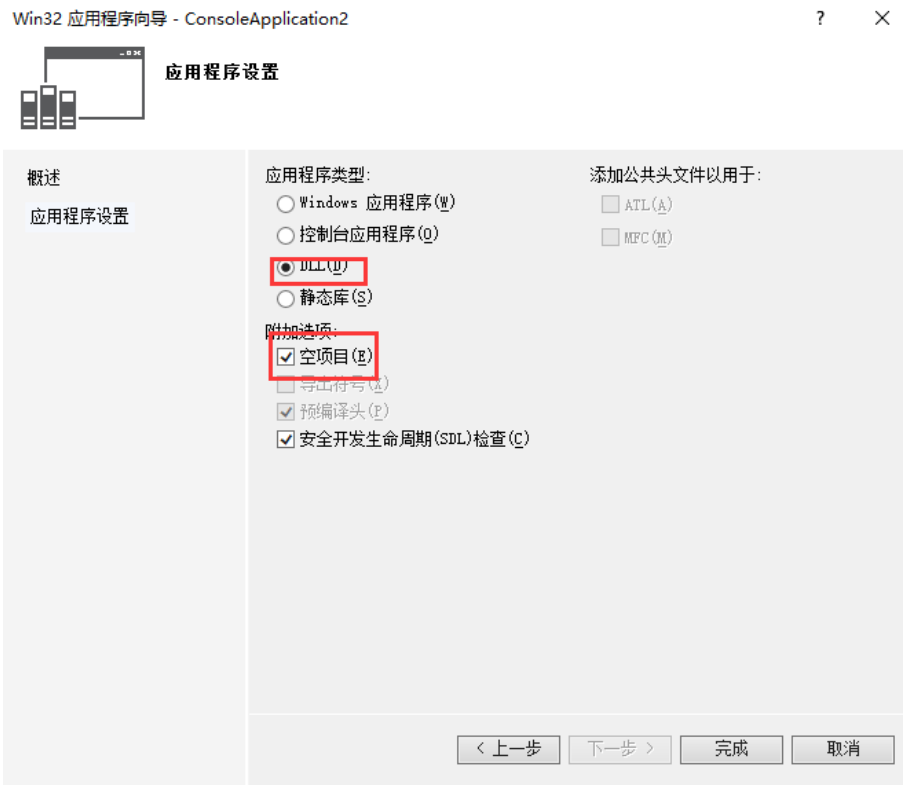
4当被HOOK位置将执行的时候，会劫持到自己的函数

5如果要执行原来的流程，那么取消HOOK

6执行原流程

7重新HOOK

win32

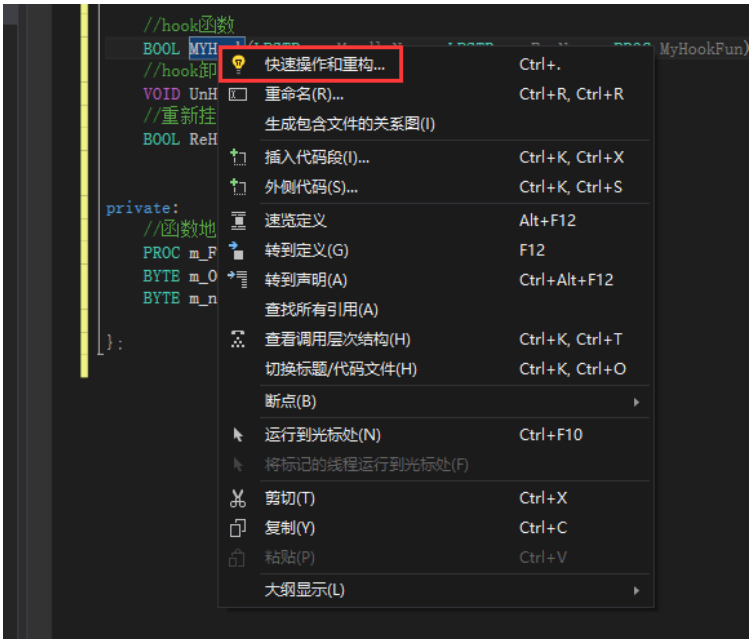


hook.c

```
#pragma once
#include<Windows.h>
class Hook
{
public:
 Hook();
```

```
7   ~Hook();
8
9   //hook函数
10   BOOL MYHook(LPSTR pszMoudleName, LPSTR pszFunName, PROC MyHookFun);
11   //hook卸载函数
12   VOID UnHook();
13   //重新挂钩函数
14   BOOL ReHook();
15
16
17  private:
18   //函数地址
19   PROC m_FunAddress;//原来函数的地址
20   BYTE m_OldFunBytes[5];//原来函数的头五个字节
21   BYTE m_newFunByte[5];//新构造的五个字节
22
23  };
24
```

vs2015快速构造



hook.cpp

```
1  #include"hook.h"
2
```

```
Hook::Hook()
{
 m_FunAddress=NULL;
 ZeroMemory(m_OldFunBytes, 5);
 ZeroMemory(m_newFunByte, 5);
}

Hook::~Hook()
{
 UnHook();
 m_FunAddress = NULL;
 ZeroMemory(m_OldFunBytes, 5);
 ZeroMemory(m_newFunByte, 5);
}

BOOL Hook::MYHook(LPSTR pszMoudleName, LPSTR pszFunName, PROC MyHookFun)
{
 BOOL ret = FALSE;
 //得到了指定模块中函数地址
 m_FunAddress = (PROC)GetProcAddress(GetModuleHandle(pszMoudleName), pszFunName);
 if (m_FunAddress != NULL)
 {
 //保存这个函数头5个字节
 DWORD Num = 0;
 ReadProcessMemory(GetCurrentProcess(), m_FunAddress, m_OldFunBytes, 5, &Num);
 //构造指令
 //JMP=E9
 m_newFunByte[0] = '\xE9';//opcode jmp address
 *(DWORD *)(m_newFunByte + 1) = (DWORD)MyHookFun - (DWORD)m_FunAddress - 5;
 //把构造好的地址写到函数头5个字节的位置上
 WriteProcessMemory(GetCurrentProcess(), m_FunAddress, m_newFunByte, 5, &Num);
 ret = TRUE;

 }



 return ret;
```

```
}

VOID Hook::UnHook()
{

 if (m_FunAddress != 0)
 {
 DWORD num = 0;
 WriteProcessMemory(GetCurrentProcess(), m_FunAddress, m_OldFunBytes, 5, &num);

 }

}

BOOL Hook::ReHook()
{
 BOOL ret = FALSE;
 if (m_FunAddress != 0)
 {
 DWORD num = 0;
 WriteProcessMemory(GetCurrentProcess(), m_FunAddress, m_newFunByte, 5, &num);
 ret = TRUE;
 }


 return ret;
}

```

选择要hook'的函数，例CreateProcess

HookCreateProcess.cpp

```
#include"hook.h"
Hook CreateProcessHook;
BOOL
WINAPI
//F12查看函数
MyCreateProcessW(
 _In_opt_ LPCWSTR lpApplicationName,
 _Inout_opt_ LPWSTR lpCommandLine,
 _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes,
```

```
  _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_ BOOL bInheritHandles,
  _In_ DWORD dwCreationFlags,
  _In_opt_ LPVOID lpEnvironment,
  _In_opt_ LPCWSTR lpCurrentDirectory,
  _In_ LPSTARTUPINFOW lpStartupInfo,
  _Out_ LPPROCESS_INFORMATION lpProcessInformation
 )
 {

  BOOL ret = FALSE;
  if (MessageBoxW(NULL, L"是否启动该进程", L"提示", MB_YESNO) == IDYES)
  {
  CreateProcessHook.UnHook();
  CreateProcessW(lpApplicationName, lpCommandLine, lpProcessAttributes, lpThreadAttributes, bInheritHandles, dwCreationFlags, lpEnvironment, lpCurrentDirectory, lpStartupInfo, lpProcessInformation);
  CreateProcessHook.ReHook();

  }
  else
  {
  MessageBoxW(NULL, L"启动的进程已经被拦截！", L"提示", MB_OK);

  }
  return ret;
 }
 BOOL APIENTRY DllMain(HANDLE hModle, DWORD ul_reason_for_call, LPVOID lpReserved)
 {
  switch (ul_reason_for_call)
  {
  case DLL_PROCESS_ATTACH:
  CreateProcessHook.MYHook("kernel32.dll", "CreateProcessW", (PROC)MyCreateProcessW);
  break;
  case DLL_PROCESS_DETACH:
  CreateProcessHook.UnHook();
  break;

  }
```

```
47    return TRUE;
48  }
```