

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

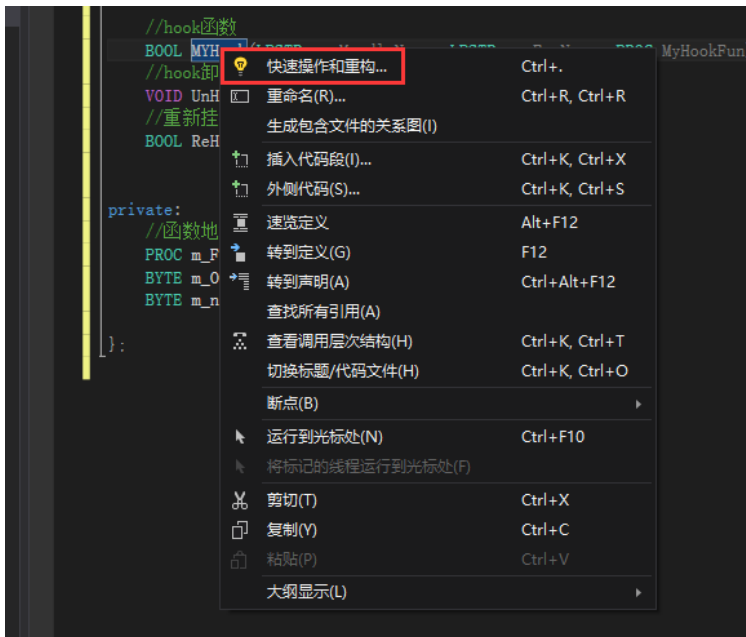
```
1 #pragma once
2 #include<Windows.h>
3 class Hook
4 {
5 public:
6     Hook();
```

```

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

```

```

3 Hook::Hook()
4 {
5     m_FunAddress=NULL;
6     ZeroMemory(m_OldFunBytes, 5);
7     ZeroMemory(m_newFunByte, 5);
8 }
9
10 Hook::~~Hook()
11 {
12     UnHook();
13     m_FunAddress = NULL;
14     ZeroMemory(m_OldFunBytes, 5);
15     ZeroMemory(m_newFunByte, 5);
16 }
17
18 BOOL Hook::MYHook(LPSTR pszMoudleName, LPSTR pszFunName, PROC MyHookFun)
19 {
20     BOOL ret = FALSE;
21     //得到了指定模块中函数地址
22     m_FunAddress = (PROC)GetProcAddress(GetModuleHandle(pszMoudleName), pszFunName);
23     if (m_FunAddress != NULL)
24     {
25         //保存这个函数头5个字节
26         DWORD Num = 0;
27         ReadProcessMemory(GetCurrentProcess(), m_FunAddress, m_OldFunBytes, 5, &Num);
28         //构造指令
29         //JMP=E9
30         m_newFunByte[0] = '\xE9';//opcode jmp address
31         *(DWORD*)(m_newFunByte + 1) = (DWORD)MyHookFun - (DWORD)m_FunAddress - 5;
32         //把构造好的地址写到函数头5个字节的位置上
33         WriteProcessMemory(GetCurrentProcess(), m_FunAddress, m_newFunByte, 5, &Num);
34         ret = TRUE;
35     }
36 }
37
38
39
40 return ret;

```

```

41 }
42
43 VOID Hook::UnHook()
44 {
45
46     if (m_FunAddress != 0)
47     {
48         DWORD num = 0;
49         WriteProcessMemory(GetCurrentProcess(), m_FunAddress, m_OldFunBytes, 5,
&num);
50
51     }
52
53 }
54
55 BOOL Hook::ReHook()
56 {
57     BOOL ret = FALSE;
58     if (m_FunAddress != 0)
59     {
60         DWORD num = 0;
61         WriteProcessMemory(GetCurrentProcess(), m_FunAddress, m_newFunByte, 5,
&num);
62         ret = TRUE;
63     }
64
65
66     return ret;
67 }
68

```

选择要hook'的函数，例CreateProcess

HookCreateProcess.cpp

```

1  #include"hook.h"
2  Hook CreateProcessHook;
3  BOOL
4  WINAPI
5  //F12查看函数
6  MyCreateProcessW(
7      _In_opt_ LPCWSTR lpApplicationName,
8      _Inout_opt_ LPWSTR lpCommandLine,
9      _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes,

```

```

10  _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
11  _In_ BOOL bInheritHandles,
12  _In_ DWORD dwCreationFlags,
13  _In_opt_ LPVOID lpEnvironment,
14  _In_opt_ LPCWSTR lpCurrentDirectory,
15  _In_ LPSTARTUPINFO lpStartupInfo,
16  _Out_ LPPROCESS_INFORMATION lpProcessInformation
17  )
18  {
19
20  BOOL ret = FALSE;
21  if (MessageBoxW(NULL, L"是否启动该进程", L"提示", MB_YESNO) == IDYES)
22  {
23  CreateProcessHook.UnHook();
24  CreateProcessW(lpApplicationName, lpCommandLine, lpProcessAttributes, lpThreadAttributes, bInheritHandles, dwCreationFlags, lpEnvironment, lpCurrentDirectory, lpStartupInfo, lpProcessInformation);
25  CreateProcessHook.ReHook();
26
27  }
28  else
29  {
30  MessageBoxW(NULL, L"启动的进程已经被拦截!", L"提示", MB_OK);
31
32  }
33  return ret;
34  }
35  BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
36  {
37  switch (ul_reason_for_call)
38  {
39  case DLL_PROCESS_ATTACH:
40  CreateProcessHook.MYHook("kernel32.dll", "CreateProcessW", (PROC)MyCreateProcessW);
41  break;
42  case DLL_PROCESS_DETACH:
43  CreateProcessHook.UnHook();
44  break;
45
46  }

```

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

ConsoleApplication2 属性页

? X

配置(C): 活动(Debug) 平台(P): 活动(Win32) 配置管理器(O)...

配置属性	启用字符串池	
常规	启用最小重新生成	是 (/Gm)
调试	启用 C++ 异常	是 (/EHsc)
VC++ 目录	较小类型检查	否
C/C++	基本运行时检查	两者(/RTC1, 等同于 /RTCsu) (/RTC1)
常规	运行库	多线程调试 (/MTd)
优化	结构成员对齐	默认设置
预处理器	安全检查	启用安全检查 (/GS)
代码生成	控制流防护	
语言	启用函数级链接	
预编译头	启用并行代码生成	
输出文件	启用增强指令集	未设置
浏览信息	浮点模型	精度 (/fp:precise)
高级	启用浮点异常	
所有选项	创建可热修补映像	
命令行		
链接器		
清单工具		
XML 文档生成器		
浏览信息		
生成事件		
自定义生成步骤		
代码分析		

运行库  
指定运行库以进行链接。 (/MT, /MTd, /MD, /MDd)

确定 取消 应用(A)