关于看雪上那份inline hook代码的小问题

原创

hbprotoss 于 2011-10-02 15:20:09 发布 1125 收藏
分类专栏： 安全相关 C++ 文章标签： hook byte system api null
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/digimon/article/details/6840543
版权


安全相关 同时被 2 个专栏收录
13 篇文章 0 订阅
订阅专栏


C++
2 篇文章 0 订阅
订阅专栏

源代码见后文。

在debug版本中总是出现堆栈平衡检查通不过的情况~~单步跟了一下发现是在detour函数里检查堆栈平衡前没有保存前一栈帧的ESI，导致当前堆栈平衡检查完之后的ESI保存的是当前函数被调用前的ESP，和上一层调用者的ESP不一样，所以__RTC_CheckEsp的时候报错。

解决办法很简单，detour函数里调函数前保存一下ESI，完了之后再恢复，这样就不会在debug版本中报堆栈平衡被破坏的bug了：）

（MyMessageBoxA函数里那两个条件编译~~）

代码

```
#include <windows.h>
#include <iostream>
//using namespace std;

DWORD head;//保存API返回地址
int nRet;
BYTE orig_code[5] = {0x90, 0x90, 0x90, 0x90, 0x90};//存放原始的指令
BYTE hook_code[5] = {0xe9, 0, 0, 0, 0};//存放跳转到MyMessageBoxA的指令
BYTE jmp_org_code[5] = {0xe9, 0, 0, 0, 0};//存放跳转到原起始地址后5字节的指令

int MyMessageBoxA(
 HWND hWnd,          // handle to owner window
 LPCTSTR lpText,     // text in message box
 LPCTSTR lpCaption,  // message box title
 UINT uType          // message box style
 );
int MyMessageBoxAA(
 HWND hWnd,          // handle to owner window
 LPCTSTR lpText,     // text in message box
 LPCTSTR lpCaption,  // message box title
 UINT uType          // message box style
 );
int MyFunc();
```

```
void Hook();
int jmp_back();

ULONG OldFuncAddr;
ULONG MyFuncAddr;
ULONG jmp_backAddr;

//在修改前几个字节时，注意：取出的指令为完整的

int main()
{
 Hook();

 int rt = MessageBoxA(NULL, "Hello World", "Title", MB_OK);
 //  cout << rt << endl;//查看返回值是否已修改成功

 getchar();

 system("pause");

 return 0;
}

void Hook()
{
 DWORD dwOldProtect;
 OldFuncAddr = (ULONG)MessageBoxA;

 // MyFuncAddr = MyMessageBoxA的实际地址
 MyFuncAddr = *(ULONG *)((BYTE *)MyMessageBoxA+1) + (ULONG)MyMessageBoxA + 5;
 // jmp_backAddr = jmp_back的实际地址
 jmp_backAddr = *(ULONG *)((BYTE *)jmp_back+1) + (ULONG)jmp_back + 5;
 //修改内存为PAGE_EXECUTE_READWRITE
 VirtualProtect((LPVOID)jmp_backAddr, 10, PAGE_EXECUTE_READWRITE, &dwOldProtect);

 VirtualProtect((LPVOID)OldFuncAddr, 5, PAGE_EXECUTE_READWRITE, &dwOldProtect);
 //计算跳转地址
 *((ULONG*)(hook_code+1)) = (ULONG)MyFuncAddr - (ULONG)OldFuncAddr - 5;

 memcpy(orig_code,(BYTE *)OldFuncAddr, 5);

 memcpy((BYTE*)OldFuncAddr, hook_code, 5);
 //计算返回地址
 *((ULONG*)(jmp_org_code+1)) = (ULONG)OldFuncAddr - (ULONG)jmp_backAddr - 5;

 memcpy((BYTE *)jmp_backAddr, orig_code, 5);

 memcpy((BYTE *)jmp_backAddr + 5, jmp_org_code, 5);
}

__declspec(naked) int jmp_back()
{
 __asm
 {
  _emit 0x90
  _emit 0x90
  _emit 0x90
  _emit 0x90
  _emit 0x90
```

```
  _emit 0x90
  _emit 0x90
  _emit 0x90
  _emit 0x90
  _emit 0x90
 }
}

//MyMessageBoxA：在函数执行前进行自己的处理
__declspec(naked) int MyMessageBoxA(
 HWND hWnd,          // handle to owner window
 LPCTSTR lpText,     // text in message box
 LPCTSTR lpCaption,  // message box title
 UINT uType          // message box style
 )
{
#ifdef DEBUG
 __asm push esi
#endif

 printf("MyMessageBoxA is called\r\n");

#ifdef DEBUG
 __asm pop esi
#endif
 __asm
 {
  pop head
  pop hWnd
  pop lpText
  pop lpCaption
  pop uType
 }
 MyFunc();可以加入函数过程
 __asm
 {
  //压栈过程
  push uType
  push lpCaption
  push lpText
  push hWnd
  push head
  //跳回MessageBoxA入口点
  jmp jmp_back;
  ret;
 }
}

int MyFunc()
{
 printf("Hello World\r\n");
 return 1;
}
```