必知必会-Hot Patch(热补丁)的两种方案

原创

dalerkd 于 2018-01-04 15:08:47 发布 15683 收藏 9

分类专栏： 深造之旅 有趣-好玩 啊哈-灵光一闪

本文链接：https://blog.csdn.net/dalerkd/article/details/78971171


深造之旅 同时被 3 个专栏收录

103 篇文章 0 订阅

订阅专栏


有趣-好玩

54 篇文章 0 订阅

订阅专栏

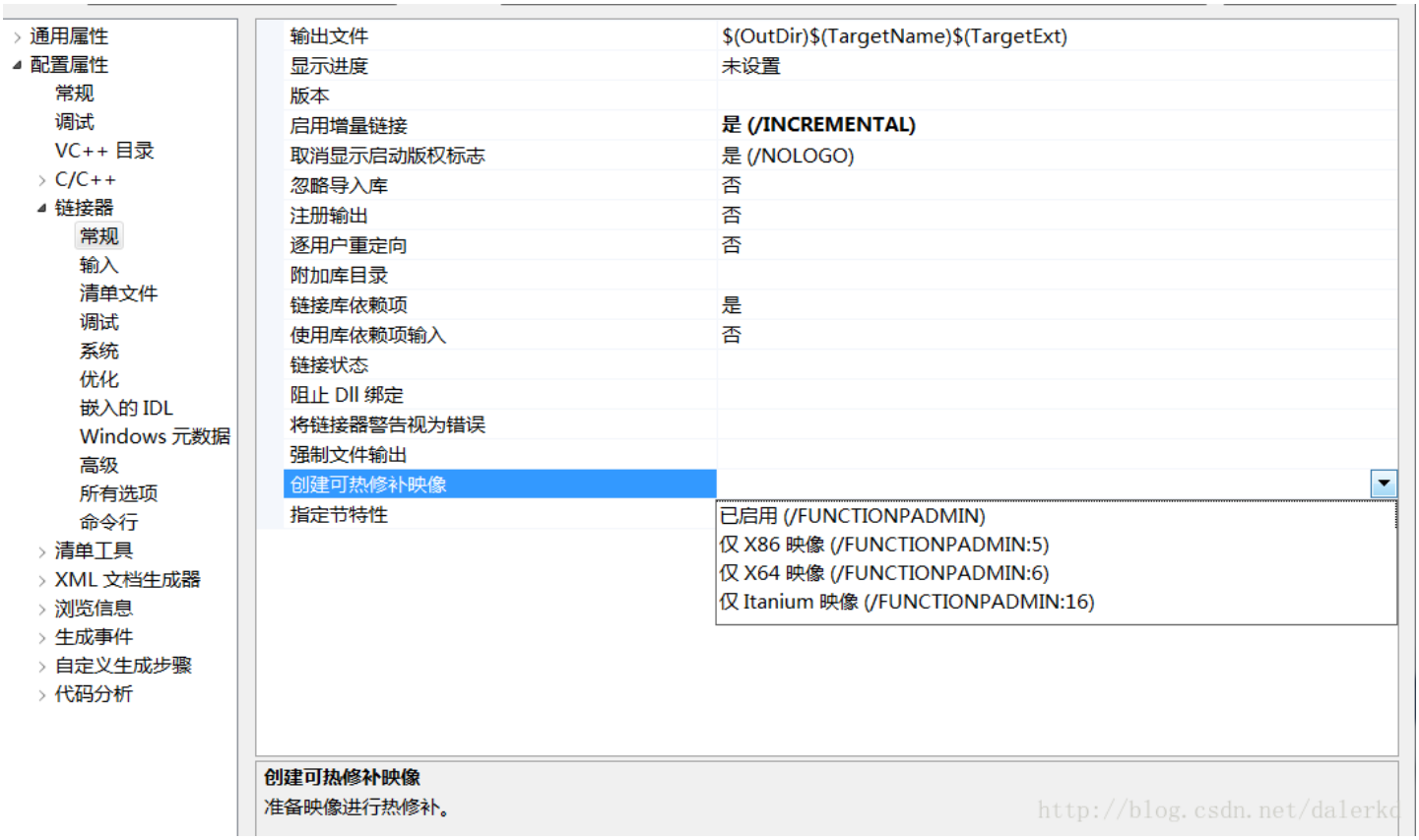

啊哈-灵光一闪

2 篇文章 0 订阅

订阅专栏

素闻hot patch热补丁，它到底是个什么鬼？我之前只听说过，它很给力，实际上我在小学时候已经听说**热补丁**，国防科大教授让我们**给系统打热补丁**。

先看这里：



如何对多线程程序进行补丁正是另人苦恼的事情，前两天因为工作遇到要inline hook的问题：

我想出来的办法+代码是：

- 先修改 `mov edi,edi` 为原地死循环。
- 然后修改余下的`push ebp mov ebp,esp为即将跳转过去的部分长度`
- 再修改死循环的两个字节为：E9 + XX//跳转的一部分

```
        *p = (WORD)0xFEEB;//实为EBFE    JMP $

        /*
        mov edi,edi        0x8b 0xff
        push ebp        0x55
        mov ebp,esp        0x8b 0xec
```

```
    void Work();
    bool Init();
```

```
void Patch();
/*
1. 准备好
2. 先修改造成死循环
8B FF
修改成：
EB FE
3. 半修改
8B FF 55 8B EC
E9 12 34 56 78//jmp 到我们的dll里面
//对参数对比，修改后进行处理，
//
4. 恢复
利用了原子操作
*/
FARPROC pSystemAddress = nullptr;
FARPROC pWorkAddress = nullptr;
DWORD oldProtect;
DWORD lengthChange = 0x5;
char* ByeAddress = nullptr;//回跳地址
bool Init(){
    //HMODULE hModule        = LoadLibrary(L"kernel32.dll");
    bool suc = false;
    HMODULE hModule = GetModuleHandle(L"kernel32.dll");
    if (hModule == 0)
    {
        return false;
    }
    pSystemAddress = GetProcAddress(hModule, "CreateProcessW");//能不能获取?
    if (pSystemAddress == 0)
    {
        return false;
    }
    //hModule = GetModuleHandle(NULL);
    //if (hModule==0)
    //{
    //    return false;
    //}
    pWorkAddress = (FARPROC)Work;
    if (pWorkAddress == 0)
    {
        return false;
    }
    ByeAddress = ((char*)pSystemAddress) + 5;
    return true;
}
//1修改页面属性
//2写地址
//3恢复属性
void Patch(){
    VirtualProtect(pSystemAddress, lengthChange, PAGE_EXECUTE_READWRITE, &oldProtect);
    PWORD p = (PWORD)pSystemAddress;//修改成死循环，防止出问题
    *p = (WORD)0xFEEB;//实为EBFE
    int offset = (DWORD)pWorkAddress - (DWORD)pSystemAddress - 5;
    *(((char*)p) + 2) = *((char*)(&offset) + 1);
    *(((char*)p) + 3) = *((char*)(&offset) + 2);
    *(((char*)p) + 4) = *((char*)(&offset) + 3);
    *p = (WORD)(0xE9 + (((WORD)(*((char*)(&offset) + 0))) << 8));
    VirtualProtect(pSystemAddress, lengthChange, oldProtect, &oldProtect);
```

```
}
//尽力避免冲突
//1修改页面属性
//2查看bWork;.....
//2写地址
//3恢复属性
void UnPatch(){
    VirtualProtect(pSystemAddress, lengthChange, PAGE_EXECUTE_READWRITE, &oldProtect);
    PWORD p = (PWORD)pSystemAddress;//修改成死循环，防止出问题
    *p = (WORD)0xFEEB;//实为EBFE    JMP $
    /*
    mov edi,edi        0x8b 0xff
    push ebp        0x55
    mov ebp,esp        0x8b 0xec
    */
    *(((unsigned char*)p) + 2) = 0x55;
    *(((unsigned char*)p) + 3) = 0x8b;
    *(((unsigned char*)p) + 4) = 0xec;
    *p = (WORD)(0x8b + (0xff << 8));
    VirtualProtect(pSystemAddress, lengthChange, oldProtect, &oldProtect);
}
/*
防止内容在work()中
看标记哦
4. 企修改
裸函数
//获取字符串指针
mov eax,[ebp+4]//因为是push ebp之前所以+4
pushad
LPVOID str;
mov str,eax
wcsstr()//是否子串，返回位置，为NULL则不是。
je bye
//字符串复制
。。。。。。
对参数对比，修改后进行处理。
pushad
获取字符串对比之
是：
- 增加字符跳回；
否，直接跳回。
: bye
_asm popad
_asm push ebp
_asm mov ebp,esp
jmp 接下来的位置
lstrcmp
*/
bool bWork = false;
WCHAR LockHomePage[] = L"\"C:\\Users\\Administrator\\AppData\\Roaming\\360se6\\Application\\360se.exe\"
WCHAR NeedCmpString[] = L"\"C:\\Users\\Administrator\\AppData\\Roaming\\360se6\\Application\\360se.exe\
/*是否需要释放原来的指针需要跟踪*/
void __declspec(naked) Work(){//需要裸函数
    _asm {
        mov bWork, 1//true
            push ebp
            mov ebp, esp
            mov eax, [esp + 0xC] //得到字符串，对比之
            push eax
            lea eax, NeedCmpString
```

```
            push eax
            call lstrcmpW
            cmp eax, 0
            jne  bye
            lea eax, LockHomePage
            mov[esp + 0xC], eax
        bye :
        mov bWork, 0//false
            mov eax, ByeAddress
            jmp eax
    }
}
```

“热补丁”Hook，多线程下InlineHook解决方法-文章作者的做法:

https://www.cnblogs.com/Toring/p/6664481.html

我的分析+截图

黄色区域为CreateProcessW的入口地址

```
75861038 90                nop
75861039 90                nop
7586103A 90                nop
7586103B 90                nop
7586103C 90                nop
7586103D 8B FF             mov         edi,edi
7586103F 55                push        ebp
75861040 8B EC             mov         ebp,esp
75861042 6A 00             push        0
75861044 FF 75 2C          push        dword ptr [ebp+2Ch]
75861047 FF 75 28          push        dword ptr [ebp+28h]
7586104A FF 75 24          push        dword ptr [ebp+24h]
7586104D FF 75 20          push        dword ptr [ebp+20h]
75861050 FF 75 1C          push        dword ptr [ebp+1Ch]
75861053 FF 75 18          push        dword ptr [ebp+18h]
75861056 FF 75 14          push        dword ptr [ebp+14h]
75861059 FF 75 10          push        dword ptr [ebp+10h]
7586105C FF 75 0C          push        dword ptr [ebp+0Ch]
```

先布置好劫持到目标的跳转在没有影响的区域:0x75861038

`E9 XX XX XX XX`替换原来的`NOP NOP NOP NOP NOP`

```
75861038 E9 53 01 79 8B       jmp         NewCreateProcessW (OFF1190h)
7586103D 8B FF                mov         edi,edi
7586103F 55                   push        ebp
75861040 8B EC                mov         ebp,esp
75861042 6A 00                push        0
75861044 FF 75 2C             push        dword ptr [ebp+2Ch]
75861047 FF 75 28             push        dword ptr [ebp+28h]
7586104A FF 75 24             push        dword ptr [ebp+24h]
7586104D FF 75 20             push        dword ptr [ebp+20h]
75861050 FF 75 1C             push        dword ptr [ebp+1Ch]
75861053 FF 75 18             push        dword ptr [ebp+18h]
75861056 FF 75 14             push        dword ptr [ebp+14h]
75861059 FF 75 10             push        dword ptr [ebp+10h]
7586105C FF 75 0C             push        dword ptr [ebp+0Ch]
7586105F FF 75 08             push        dword ptr [ebp+8]
75861062 6A 00                push        0
75861064 E8 52 2B 01 00       call        75873BBB
75861069 5D                   pop         ebp
7586106A C2 28 00             ret         28h
7586106D 90                   nop
7586106E 90                   nop
```


将入口`mov edi,edi`改成`EB F9`，构成一个死循环。

patch后：

```
75861038 E9 53 01 79 8B       jmp         NewCreateProcessW (OFF1190h)
7586103D EB F9                jmp         75861038
7586103F 55                   push        ebp
75861040 8B EC                mov         ebp,esp
75861042 6A 00                push        0
75861044 FF 75 2C             push        dword ptr [ebp+2Ch]
75861047 FF 75 28             push        dword ptr [ebp+28h]
7586104A FF 75 24             push        dword ptr [ebp+24h]
7586104D FF 75 20             push        dword ptr [ebp+20h]
75861050 FF 75 1C             push        dword ptr [ebp+1Ch]
75861053 FF 75 18             push        dword ptr [ebp+18h]
75861056 FF 75 14             push        dword ptr [ebp+14h]
75861059 FF 75 10             push        dword ptr [ebp+10h]
7586105C FF 75 0C             push        dword ptr [ebp+0Ch]
7586105F FF 75 08             push        dword ptr [ebp+8]
75861062 6A 00                push        0
75861064 E8 52 2B 01 00       call        75873BBB
75861069 5D                   pop         ebp
7586106A C2 28 00             ret         28h
7586106D 90                   nop
7586106E 90                   nop
7586106F 90                   nop
75861070 90                   nop
```


只要双字节修改没有冲突即可，这个CPU和内存应当可以保证。

以下是

原文：

“热补丁”（hot patch）是微软提出的一种安全Hook的机制，也是为了方便开发者对某些API函数进行下钩子。这种方法不同于普通的Inline hook更改首部的五个字节，而是更改首部的七个字节。为什么是七个字节呢？下边我们来讲一下这个的原理。

我们可以看到CreateProcessW函数的首字节为 mov edi,edi（88 FF），这句汇编意思就是将edi的值放入edi，实际上并没有什么用，我们还看到在这个API上边有大段无用的字节。这就给了我们一种新的Hook思路，即将前两个字节改为短跳转指令（EB E9），使其跳到函数上边五字节处，然后再将这五个字节改为长跳转指令（E9 xxxxxxxx）。这样，即使Hook失败，也不影响函数的继续执行。

接下来我们用代码来实现这一功能：

```
// HotPatch.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#include <Windows.h>

typedef BOOL(WINAPI *pfnCreateprocessW)(
    LPCTSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
    );

BOOL SetPrivilege(LPCTSTR lpszPrivilege, BOOL bEnablePrivilege);
BOOL HOOKByHotpatch(LPWSTR wzDllName, LPCSTR szFuncName, PROC pfnNewFunc);
BOOL UnhookByHotpatch(LPWSTR wzDllName, LPCSTR szFuncName);
BOOL WINAPI NewCreateProcessW(
    LPCTSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);
int main()
{
    SetPrivilege(SE_DEBUG_NAME, TRUE);

    //hook
    HOOKByHotpatch(L"kernel32.dll", "CreateProcessW",
        (PROC)NewCreateProcessW);

    Sleep(1000);

    STARTUPINFO si = { 0 };
    si.cb = sizeof(si);
    si.dwFlags = STARTF_USESHOWWINDOW;
    si.wShowWindow = SW_SHOW;
    PROCESS_INFORMATION pi;
```

```
    //创建进程
    TCHAR cmdLine[MAXBYTE] = L"notepad.exe";
    BOOL bOk = CreateProcess(NULL, cmdLine,
        NULL, NULL, FALSE, NULL,
        NULL, NULL, &si, &pi);

    Sleep(1000);

    UnhookByHotpatch(L"kernel32.dll", "CreateProcessW");
    return 0;
}


//设置权限
BOOL SetPrivilege(LPCTSTR lpszPrivilege, BOOL bEnablePrivilege)
{
    TOKEN_PRIVILEGES TokenPrivileges;//权限令牌
    HANDLE TokenHandle = NULL;      //权限令牌句柄
    LUID  Luid;

    if (!OpenProcessToken(GetCurrentProcess(),
        TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
        &TokenHandle))
    {
        printf("OpenProcessToken error: %u\n", GetLastError());
        return FALSE;
    }

    if (!LookupPrivilegeValue(NULL,            // lookup privilege on local system
        lpszPrivilege,    // privilege to lookup
        &Luid))          // receives LUID of privilege
    {
        printf("LookupPrivilegeValue error: %u\n", GetLastError());
        return FALSE;
    }

    TokenPrivileges.PrivilegeCount = 1;
    TokenPrivileges.Privileges[0].Luid = Luid;
    if (bEnablePrivilege)
        TokenPrivileges.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
    else
        TokenPrivileges.Privileges[0].Attributes = 0;

    // Enable the privilege or disable all privileges.
    //调整权限
    if (!AdjustTokenPrivileges(TokenHandle,
        FALSE,
        &TokenPrivileges,
        sizeof(TOKEN_PRIVILEGES),
        (PTOKEN_PRIVILEGES)NULL,
        (PDWORD)NULL))
    {
        printf("AdjustTokenPrivileges error: %u\n", GetLastError());
        return FALSE;
    }

    if (GetLastError() == ERROR_NOT_ALL_ASSIGNED)
    {
        printf("The token does not have the specified privilege. \n");
```

```
        return FALSE;
    }

    return TRUE;
}

//Hootpatch，将函数首字节改为 short jmp（EB F9）
BOOL HOOKByHotpatch(LPWSTR wzDllName, LPCSTR szFuncName, PROC pfnNewFunc)
{
    FARPROC pOrgFuncAddr = NULL;
    DWORD  dwOldProtect, dwAddress;
    BYTE    pBuf[5] = { 0xE9, 0, };
    BYTE    pBuf2[2] = { 0xEB, 0xF9 };
    PBYTE  pByte;

    pOrgFuncAddr = (FARPROC)GetProcAddress(GetModuleHandle(wzDllName), szFuncName);
    pByte = (PBYTE)pOrgFuncAddr;
    //判断是否被勾
    if (pByte[0] == 0xEB)
        return FALSE;

    //将前五字节代码改为可读可写
    VirtualProtect((LPVOID)((DWORD)pOrgFuncAddr - 5), 7, PAGE_EXECUTE_READWRITE, &dwOldProtect);

    // 1. NOP (0x90)
    //将前五字节改为E8 xxxxxxxx
    dwAddress = (DWORD)pfnNewFunc - (DWORD)pOrgFuncAddr;
    memcpy(&pBuf[1], &dwAddress, 4);
    memcpy((LPVOID)((DWORD)pOrgFuncAddr - 5), pBuf, 5);

    // 2. MOV EDI, EDI (0x8BFF)
    //将函数前两个字节改为EB F9
    memcpy(pOrgFuncAddr, pBuf2, 2);

    VirtualProtect((LPVOID)((DWORD)pOrgFuncAddr - 5), 7, dwOldProtect, &dwOldProtect);

    return TRUE;
}

BOOL UnhookByHotpatch(LPWSTR wzDllName, LPCSTR szFuncName)
{
    FARPROC pHookFunc = NULL;
    DWORD  dwOldProtect;
    PBYTE  pByte;
    BYTE    pBuf[5]  = { 0x90, 0x90, 0x90, 0x90, 0x90 };
    BYTE    pBuf2[2] = { 0x8B, 0xFF };


    pHookFunc = (FARPROC)GetProcAddress(GetModuleHandle(wzDllName), szFuncName);
    pByte = (PBYTE)pHookFunc;
    if (pByte[0] != 0xEB)
        return FALSE;

    VirtualProtect((LPVOID)pHookFunc, 5, PAGE_EXECUTE_READWRITE, &dwOldProtect);

    // 1. NOP (0x90)
    memcpy((LPVOID)((DWORD)pHookFunc - 5), pBuf, 5);

    // 2. MOV EDI, EDI (0x8BFF)
```

```
        memcpy(pHookFunc, pBuf2, 2);

        VirtualProtect((LPVOID)pHookFunc, 5, dwOldProtect, &dwOldProtect);

        return TRUE;
    }

    BOOL WINAPI NewCreateProcessW(
        LPCTSTR lpApplicationName,
        LPTSTR lpCommandLine,
        LPSECURITY_ATTRIBUTES lpProcessAttributes,
        LPSECURITY_ATTRIBUTES lpThreadAttributes,
        BOOL bInheritHandles,
        DWORD dwCreationFlags,
        LPVOID lpEnvironment,
        LPCTSTR lpCurrentDirectory,
        LPSTARTUPINFO lpStartupInfo,
        LPPROCESS_INFORMATION lpProcessInformation
    )
    {
        MessageBox(NULL, L"Hook", L"", 0);
        return TRUE;
    }
```

需要注意的是使用这一方法钩取的适用条件（NOP*5指令+MOV ESI,ESI），使用的时候一定要反汇编看一下目标函数是否满足条件。

优缺点分析

两种方案都能达到目的，都是基于单个字节操作是线程安全的。
我的方案使用了两次该操作，而另一种方案是使用了一次该操作。
我推荐第二种方案，直观。