Windows2000下Api函数的拦截分析 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/245/2021_2022_Windows200_c97_245973.htm 简介： Api拦截并不是一个新的技术，很多商业软件都采用这种技术。对windows的Api函数的拦截，不外乎两种方法，第一种是Mr. Jeffrey Richter 的修改exe文件的模块输入节，种方法，很安全，但很复杂，而且有些exe文件，没有Dll的输入符号的列表，有可能出现拦截不到的情况。第二种方法就是常用的JMP XXX的方法，虽然很古老，却很简单实用。 本文一介绍第二种方法在Win2k下的使用。第二种方法，Win98/me 下因为进入Ring0级的方法很多，有LDT,IDT,Vxd等方法，很容易在内存中动态修改代码，但在Win2k下，这些方法都不能用，写WDM太过复杂，表面上看来很难实现，其实不然。Win2k为我们提供了一个强大的内存Api操作函数---VirtualProtectEx，WriteProcessMemeory,ReadProcessMemeory，有了它们我们就能在内存中动态修改代码了，其原型为： BOOL VirtualProtectEx(HANDLE hProcess, // 要修改内存的进程句柄 LPVOID lpAddress, // 要修改内存的起始地址 DWORD dwSize, // 修改内存的字节 DWORD flNewProtect, // 修改后的内存属性 PDWORD lpflOldProtect // 修改前的内存属性的地址). BOOL WriteProcessMemory(HANDLE hProcess, // 要写进程的句柄 LPVOID lpBaseAddress, // 写内存的起始地址 LPVOID lpBuffer, // 写入数据的地址 DWORD nSize, // 要写的字节数 LPDWORD lpNumberOfBytesWritten // 实际写入的子节数). BOOL ReadProcessMemory(HANDLE hProcess, // 要读进程的句柄

LPCVOID lpBaseAddress, // 读内存的起始地址 LPVOID lpBuffer, // 读入数据的地址 DWORD nSize, // 要读入的字节数 LPDWORD lpNumberOfBytesRead // 实际读入的子节数). 具体的参数请参看MSDN帮助。在Win2k下因为Dll和所属进程在同一地址空间，这点又和Win9x/me存在所有进程存在共享的地址空间不同，因此，必须通过钩子函数和远程注入进程的方法，现以一个简单采用钩子函数对MessageBoxA进行拦截例子来说明： 其中Dll文件为： HHOOK g_hHook. HINSTANCE g_hinstDll. FARPROC pfMessageBoxA. int WINAPI MyMessageBoxA(HWND hWnd, LPCTSTR lpText,LPCTSTR lpCaption,UINT uType). BYTE OldMessageBoxACode[5],NewMessageBoxACode[5]. HMODULE hModule . DWORD dwIdOld,dwIdNew. BOOL bHook=false. void HookOn(). void HookOff(). BOOL init(). LRESULT WINAPI MousHook(int nCode,WPARAM wParam,LPARAM lParam). BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul_reason_for_call, LPVOID lpReserved) { switch (ul_reason_for_call) { case DLL_PROCESS_ATTACH: if(!init()) { MessageBoxA(NULL,"Init","ERROR",MB_OK). return(false). } case DLL_THREAD_ATTACH: case DLL_THREAD_DETACH: case DLL_PROCESS_DETACH: if(bHook) UnintallHook(). break. } return TRUE. } LRESULT WINAPI Hook(int nCode,WPARAM wParam,LPARAM lParam)//空的钩子函数 { return(CallNextHookEx(g_hHook,nCode,wParam,lParam)). } HOOKAPI2_API BOOL InstallHook()//输出安装空的钩子函数

{ g_hinstDll=LoadLibrary("HookApi2.dll"). g_hHook=SetWindowsHookEx(WH_GETMESSAGE,(HOOKPROC)Hook,g_hinstDll,0). if (!g_hHook) { MessageBoxA(NULL,"SET ERROR","ERROR",MB_OK). return(false). } return(true). } HOOKAPI2_API BOOL UninstallHook()//输出御在钩子函数 { return(UnhookWindowsHookEx(g_hHook)). } BOOL init()//初始化得到MessageBoxA的地址，并生成Jmp XXX(MyMessageBoxA)的跳转指令 { hModule=LoadLibrary("user32.dll"). pfMessageBoxA=GetProcAddress(hModule,"MessageBoxA"). if(pfMessageBoxA==NULL) return false. _asm { lea edi,OldMessageBoxACode mov esi,pfMessageBoxA cld movsd movsb } NewMessageBoxACode[0]=0xe9.//jmp MyMessageBoxA 的相对地址的指令 _asm { lea eax,MyMessageBoxA mov ebx,pfMessageBoxA sub eax,ebx sub eax,5 mov dword ptr [NewMessageBoxACode 1],eax } dwIdNew=GetCurrentProcessId(). //得到所属进程的ID dwIdOld=dwIdNew. HookOn().//开始拦截 return(true). } int WINAPI MyMessageBoxA(HWND hWnd, LPCTSTR lpText,LPCTSTR lpCaption, UINT uType)//首先关闭拦截，然后才能调用被拦截的Api 函数 { int nReturn=0. HookOff(). nReturn=MessageBoxA(hWnd,"Hook",lpCaption,uType). HookOn(). return(nReturn). } void HookOn() { HANDLE hProc. dwIdOld=dwIdNew. hProc=OpenProcess(PROCESS_ALL_ACCESS,0,dwIdOld).//得

到所属进程的句柄 VirtualProtectEx(hProc,pfMessageBoxA,5,PAGE_READWRITE,amp.dwIdOld). //修改所属进程中MessageBoxA的前5个字节的属性为原来的属性 bHook=true. } void HookOff()//将所属进程中JMP MyMessageBoxA的代码改为Jmp MessageBoxA { HANDLE hProc. dwIdOld=dwIdNew. hProc=OpenProcess(PROCESS_ALL_ACCESS,0,dwIdOld). VirtualProtectEx(hProc,pfMessageBoxA,5,PAGE_READWRITE,amp.dwIdOld). bHook=false. } //测试文件： int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) { if(!InstallHook()) { MessageBoxA(NULL,"Hook Error!","Hook",MB_OK). return 1. } MessageBoxA(NULL,"TEST","TEST",MB_OK).//可以看见Test变成了Hook,也可以在其他进程中看见 if(!UninstallHook()) { MessageBoxA(NULL,"Uninstall Error!","Hook",MB_OK). return 1. } return 0. }