计算机二级辅导:如何实现API钩子计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022__E8_AE_A1_E7_AE_97_E6_9C_BA_E4_c97_645078.htm 一、 序言对大多数的Windows开发者来说，如何在Win32系统中对API函数的调用进行拦截一直是项极富挑战性的课题，因为这将是对你所掌握的计算机知识较为全面的考验，尤其是一些在如今使用RAD进行软件开发时并不常用的知识，这包括了操作系统原理、汇编语言甚至是关于机器指令代码的（听上去真是有点恐怖，不过这是事实）。 当前广泛使用的Windows操作系统中，像Win 9x和Win NT/2K，都提供了一种比较稳健的机制来使得各个进程的内存地址空间之间是相互独立，也就是说一个进程中的某个有效的内存地址对另一个进程来说是无意义的，这种内存保护措施大大增加了系统的稳定性。不过，这也使得进行系统级的API拦截的工作的难度也大大加大了。 当然，我这里所指的是比较文雅的拦截方式，通过修改可执行文件在内存中的映像中有关代码，实现对API调用的动态拦截；而不是采用比较暴力的方式，直接对可执行文件的磁盘存储中机器代码进行改写。 二、 API钩子系统一般框架通常，我们把拦截API的调用的这个过程称为是安装一个API钩子（API Hook）。一个API钩子至少有两个模块组成：一个是钩子服务器（Hook Server）模块，一般为EXE的形式；一个是钩子驱动器（Hook Driver）模块，一般为DLL的形式。 服务器主要负责向目标进程注入驱动器，使得驱动器工作在目标进程的地址空间中，这是关键的第一步。驱动器则负责实际的API拦截工作，以便在我们所关心的API函数调用的前后

能做一些我们需要的工作。 一个大家比较常见的API钩子的例子就是一些实时翻译软件（像金山词霸）中必备的的功能：屏幕抓词，它主要是对一些GDI 函数进行了拦截，获取它们的输入参数中的字符串，然后在自己的窗口中显示出来。针对上述的两个部分，有以下两点需要我们重点考虑的： 选用何种DLL注入技术 采用何种API拦截机制 三、 注入技术的选用由于在Win32系统中各个进程的地址是互相独立的，因此我们无法在一个进程中对另一个进程的代码进行有效的修改。而你要完成API钩子的工作就必须进行这种操作。因此，我们必须采取某种独特的手段，使得API钩子（准确的说是钩子驱动器）能够成为目标进程中的一部分，才有较大的可能来对目标进程数据和代码进行有控制的修改。 通常有以下几种注入方式： 1. 利用注册表如果我们准备拦截的进程连接了User32.dll，也就是使用了User32中的API（一般图形界面的应用程序都符合这个条件），那么就可以简单把你的钩子驱动器DLL的名字作为值添加在下面注册表的键下：HKEY_LOCAL_MACHINE\\Software\\Microsoft\\WindowsNT\\CurrentVersion\\Windows\\AppInit_DLLs 值的形式可以为单个DLL的文件名，或者是一组DLL的文件名，相邻的名称之间用逗号或空格间隔。所有由该值标识的DLL将在符合条件的应用程序启动的时候装载。这是一个操作系统内建的机制，相对其他方式来说危险性较小，但它有一些比较明显的缺点： 该方法仅适用于NT/2K操作系统。看看键的名称你就应该明白 为了激活或停止钩子的注入，必须重新启动Windows。这个就似乎太不方便了 不能用此方法向没有使用User32的应用程序注入DLL，例如控制台应用程序 不管需要与否，钩

子DLL将注入每一个GUI应用程序，这将导致整个系统性能的下降 2. 建立系统范围的Windows钩子要向某个进程注入DLL，一个十分普遍也是比较简单的方法就是建立在标准的Windows钩子的基础上。Windows钩子一般是在DLL中实现的，这是一个全局性的Windows钩子的基本要求，这也符合我们的需要。当我们成功地调用SetWindowsHookEx函数之后，便在系统中安装了某种类型的消息钩子，这个钩子可以是针对某个进程，也可以是针对系统中的所有进程。一旦某个进程中产生了该类型的消息，操作系统会自动把该钩子所在的DLL映像到该进程的地址空间中，从而使得消息回调函数（在SetWindowsHookEx的参数中指定）能够对此消息进行适当的处理，在这里，我们所感兴趣的当然不是对消息进行什么处理，因此在消息回调函数中只需把消息钩子向后传递就可以了，但是我们所需的DLL已经成功地注入了目标进程的地址空间，从而可以完成后续工作。 我们知道，不同进程中使用的DLL之间是不能直接共享数据的，因为它们活动在不同的地址空间中。但在Windows钩子DLL中，有一些数据，例如Windows钩子句柄HHook，这是由SetWindowsHookEx函数返回值得到的，并且作为参数将在CallNextHookEx函数和UnhookWindoesHookEx函数中使用，显然使用SetWindowsHookEx函数的进程和使用CallNextHookEx函数的进程一般不会是同一个进程，因此我们必须能够使句柄在所有的地址空间中都是有效的有意义的，也就是说，它的值必须必须在这些钩子DLL所挂钩的进程之间是共享的。为了达到这个目的，我们就应该把它存储在一个共享的数据区域中。 在VC 中我们可以采用预编译指令#pragma data_seg在DLL

文件中创建一个新的段，并且在DEF文件中把该段的属性设置为“ shared ”，这样就建立了一个共享数据段。对于使用Delphi的人来说就没有这么幸运了：没有类似的比较简单的方法（或许是有的，但我没有找到）。不过我们还是可以利用内存映像技术来申请使用一块各进程可以共享的内存区域，主要是利用了CreateFileMapping和MapViewOfFile这两个函数。这倒是一个通用的方法，适合所有的开发语言，只要它能使用Windows的API。 在Borland的BCB中有一个指令#pragma codeseg与VC 中的#pragma data_seg指令有点类似，应该也能起到一样的作用，但我试了一下，没有没有效果，而BCB的联机帮助中对此也提到的不多，不知怎样才能正确的使用。一旦钩子DLL加载进入目标进程的地址空间后，在我们调用UnHookWindowsHookEx函数之前是无法使它停止工作的，除非目标进程关闭。 这种DLL注入方式有两个优点：这种机制在Win 9x/Me和Win NT/2K中都是得到支持的，预计在以后的版本中也将得到支持 钩子DLL可以在不需要的时候，可由我们主动的调用UnHookWindowsHookEx来卸载，比起使用注册表的机制来说方便了许多尽管这是一种相当简洁明了的方法，但它也有一些显而易见的缺点：首先值得我们注意的是，Windows钩子将会降低整个系统的性能，因为它额外增加了系统在消息处理方面的时间 其次，只有当目标进程准备接受某种消息时，钩子所在的DLL才会被系统映射到该进程的地址空间中，钩子才能真正开始发挥作用。因此如果我们要对某些进程的整个生命周期内的API调用情况进行监控，用这种方法显然会遗漏某些API的调用 3. 使用CreateRemoteThread函数在我看来这是一个相当棒的方法，然

而不幸的是，CreateRemoteThread这个函数只能在Win NT/2K系统中才得到支持，虽然在Win 9x中这个API也能被安全的调用而不出错，但它除了返回一个空值之外什么也不做。整个DLL注入过程十分简单。我们知道，任何一个进程都可以使用LoadLibrary来动态地加载一个DLL。但问题是，我们如何让目标进程在我们的控制下来加载我们的钩子DLL（也就是钩子驱动器）呢？这里有一个API函数CreateRemoteThread，通过它可在一个进程中可建立并运行一个远程的线程。 调用该API需要指定一个线程函数指针作为参数，该线程函数的原型如下： Function ThreadProc(lpParam: Pointer): DWORD；我们再来看一下LoadLibrary的函数原型： Function LoadLibrary(lpFileName: PChar): HModule；可以看出，这两个函数原型实质上是完全相同的（其实返回值是否相同关系不大，因为我们是无法得到远程线程函数的返回值的），只是叫法不同而已，这种相同使得我们可以把直接把LoadLibrary当做线程函数来使用，从而在目标进程中加载钩子DLL。 类似的，当我们需要卸载钩子DLL时，也可以FreeLibrary作为线程函数来使用，在目标进程中移去钩子DLL。一切看来是十分的简洁方便。通过调用GetProcAddress函数，我们可以得到LoadLibrary函数的地址。由于LoadLibrary是Kernel32中的函数，而这个系统DLL的映射地址对每一个进程来说都是相同的，因此LoadLibrary函数的地址也是如此。这点将确保我们能把该函数的地址作为一个有效的参数传递给CreateRemoteThread使用。 AddrOfLoadLibrary := GetProcAddress(GetModuleHandle(‘ Kernel32.dll ’), ‘ LoadLibrary ’). HremoteThread :=

CreateRemoteThread(HTargetProcess, nil, 0, AddrOfLoadLibrary, HookDllName, 0, nil). 要使用CreateRemoteThread，我们需要目标进程的句柄作为参数。当我们用OpenProcess函数来得到进程的句柄时，通常是希望对此进程有全权的存取操作，也就是以PROCESS_ALL_ACCESS为标志打开进程。但对于一些系统级的进程，直接这样显然是不行的，只能返回一个的空句柄（值为零）。为此，我们必须把自己设置为拥有调试级的特权，这样将具有最大的存取权限，从而使得我们能对这些系统级的进程也可以进行一些必要的操作。 4． 通过BHO来注入DLL 有时，我们想要注入DLL的对象仅仅是Internet Explorer。幸运的是，Windows操作系统为我们提供了一个简单的归档的方法（这保证了它的可靠性）—— 利用Browser Helper Objects（BHO）。一个BHO是一个在 DLL中实现的COM对象，它主要实现了一个IObjectWithSite接口，而每当IE运行时，它会自动加载所有实现了该接口的COM对象。