Hook技术使用SetHook替换IAT表计算机等级考试 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_Hook_E6_8A_80_E6_9C_AF_c97_645648.htm 基本概念 钩子(Hook)，是Windows消息处理机制的一个平台,应用程序可以在上面设置子程以监视指定窗口的某种消息，而且所监视的窗口可以是其他进程所创建的。当消息到达后，在目标窗口处理函数之前处理它。钩子机制允许应用程序截获处理window消息或特定事件。 钩子实际上是一个处理消息的程序段，通过系统调用，把它挂入系统。每当特定的消息发出，在没有到达目的窗口前，钩子程序就先捕获该消息，亦即钩子函数先得到控制权。这时钩子函数即可以加工处理（改变）该消息，也可以不作处理而继续传递该消息，还可以强制结束消息的传递。 常见的Hook应用很多，如现在的木马盗号程序、网络防火墙、以及金山词霸等都使用了这种技术。 下面，我主要谈到我在解决最近的一个项目中用到的几种Hook技术。 Hook手段1———————替换IAT表 要了解IAT表，首先应该了解Windows的PE文件格式，这里不再赘述。简单的说，就是当一个进程加载另一个模块之，建立起来的模块内的函数名与它在模块内相对地址的映射关系。如下图: 函数A 0xb333 函数B 0x1234 Dll temp.dll A getProcAddress(B) IAT表 主 进程 主进程加载temp.dll，就把在自己的IAT表中，映射A，加载的模块名temp.dll，以及在temp.dll的代码偏移量0xb333，我们想实现用函数B来替换函数A，则，只需要把IAT表中的0xb333替换为getProcAddrss(B), 即为getModuleAddress(void) 0x1234。 在何时替换IAT表呢？ 如果这个主进程是我们自己的，那么这就

不是一个问题，我们直接在进程的代码中替换IAT表就好了。但如果这个主进程是另一个应用程序或者所有的所用程序呢？Windows的SetHook机制给我解决了这个问题。 为了说明SetHook的方法，我引用别人的一段文字： 运行机制 1、钩子链表和钩子子程： 每一个Hook都有一个与之相关联的指针列表，称之为钩子链表，由系统来维护。这个列表的指针指向指定的，应用程序定义的，被Hook子程调用的回调函数，也就是该钩子的各个处理子程。当与指定的Hook类型关联的消息发生时，系统就把这个消息传递到Hook子程。一些Hook子程可以只监视消息，或者修改消息，或者停止消息的前进，避免这些消息传递到下一个Hook子程或者目的窗口。最近安装的钩子放在链的开始，而最早安装的钩子放在最后，也就是后加入的先获得控制权。 Windows 并不要求钩子子程的卸载顺序一定得和安装顺序相反。每当有一个钩子被卸载，Windows 便释放其占用的内存，并更新整个Hook链表。如果程序安装了钩子，但是在尚未卸载钩子之前就结束了，那么系统会自动为它做卸载钩子的操作。 钩子子程是一个应用程序定义的回调函数(CALLBACK Function),不能定义成某个类的成员函数，只能定义为普通的C函数。用以监视系统或某一特定类型的事件，这些事件可以是与某一特定线程关联的，也可以是系统中所有线程的事件。 钩子子程必须按照以下的语法： LRESULT CALLBACK HookProc (int nCode, WPARAM wParam, LPARAM lParam). HookProc是应用程序定义的名字。 nCode参数是Hook代码，Hook子程使用这个参数来确定任务。这个参数的值依赖于Hook类型，每一种Hook都有自己的Hook代码特征字符集。 wParam和lParam参数的值

依赖于Hook代码，但是它们的典型值是包含了关于发送或者接收消息的信息。 2、钩子的安装与释放： 使用API函数SetWindowsHookEx()把一个应用程序定义的钩子子程安装到钩子链表中。SetWindowsHookEx函数总是在 Hook链的开头安装Hook子程。当指定类型的Hook监视的事件发生时，系统就调用与这个Hook关联的Hook链的开头的Hook子程。每一个 Hook链中的Hook子程都决定是否把这个事件传递到下一个Hook子程。Hook子程传递事件到下一个Hook子程需要调用 CallNextHookEx函数。 HHOOK SetWindowsHookEx(int idHook, // 钩子的类型，即它处理的消息类型 HOOKPROC lpfn, // 钩子子程的地址指针。如果dwThreadId参数为0 // 或是一个由别的进程创建的线程的标识， // lpfn必须指向DLL中的钩子子程。 // 除此以外，lpfn可以指向当前进程的一段钩子子程代码。 // 钩子函数的入口地址，当钩子钩到任何消息后便调用这个函数。 HINSTANCE hMod, // 应用程序实例的句柄。标识包含lpfn所指的子程的 DLL。 // 如果dwThreadId 标识当前进程创建的一个线程， // 而且子程代码位于当前进程，hMod必须为NULL。 // 可以很简单的设定其为本应用程序的实例句柄。 DWORD dwThreadId // 与安装的钩子子程相关联的线程的标识符。 // 如果为0，钩子子程与所有的线程关联，即为全局钩子。). 函数成功则返回钩子子程的句柄，失败返回NULL。 以上所说的钩子子程与线程相关联是指在一钩子链表中发给该线程的消息同时发送给钩子子程，且被钩子子程先处理。 在钩子子程中调用得到控制权的钩子函数在完成对消息的处理后，如果想要该消息继续传递，那么它必须调用另外一个SDK中的API函数 CallNextHookEx来传递它，以

执行钩子链表所指的下一个钩子子程。这个函数成功时返回钩子链中下一个钩子过程的返回值，返回值的类型依赖于钩子的类型。这个函数的原型如下： LRESULT CallNextHookEx (HHOOK hhk. int nCode. WPARAM wParam. LPARAM lParam.). hhk为当前钩子的句柄，由SetWindowsHookEx()函数返回。NCode为传给钩子过程的事件代码。 wParam和lParam 分别是传给钩子子程的wParam值，其具体含义与钩子类型有关。 钩子函数也可以通过直接返回TRUE来丢弃该消息，并阻止该消息的传递。否则的话，其他安装了钩子的应用程序将不会接收到钩子的通知而且还有可能产生不正确的结果。 钩子在使用完之后需要用UnHookWindowsHookEx()卸载，否则会造成麻烦。释放钩子比较简单，UnHookWindowsHookEx()只有一个参数。函数原型如下： UnHookWindowsHookEx (HHOOK hhk.). 函数成功返回TRUE，否则返回FALSE。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com