#define REMOTE\_THREAD 0

#define APC\_INJECOT 1

#define HOOK\_EIP 2

#define SETWINDOWHOOKEX 3

#define REGISTER\_INJECT 4

#define MODIFY\_IMPORT\_TABLE\_INJECT 5

ps:

1.宏定义后的数字是在用cmd注入时作为命令行参数传入的，传入的第一个参数为进程名，第二个参数即为注入方式。

注入一：远程线程注入（CreateRemoteThread）

首先得到目标进程空间中的Kernel32.dll模块基址，然后读取目标进程中Kernel32.dll文件，遍历其导出表找到LoadLibrary函数地址，然后在目标进程空间中申请内存（基地址为base\_address），将我们的hackdll的绝对路径写入，数最后调用CreateRemoteThread函数，将得到的LoadLibrary地址与base\_address作为参数传入，其中base\_address（hackdll绝对路径）是作为LoadLibrary函数的参数使用。

注入二：APC注入

首先在目标进程空间中申请内存（基地址virtual\_address）将hackdll的绝对路径写入，然后调用get\_remote\_proc\_address函数在目标进程空间中的kernel32.dll中找到LoadLibrary函数地址，遍历目标进程得到其所有线程，然后利用循环在目标进程中所有线程的apc队列中都加入一个LoadLibrary的函数执行体，参数为virtual\_address，只要有任意一个线程进入可提醒状态，hackdll将会获得加载，操作目标进程的所有线程只是增加命中率的一个手段。（由于该方法要等到目标进程进入可提醒状态，因此无法具体测出结果）。

注入三：HookEip注入

首先在目标进程空间中申请内存（基址为virtual\_address），然后构建我们自己的shellcode，该shellcode包含hackdll的绝对路径，目标进程空间中LoadLibrary函数的地址，对方进程主线程的原始eip值，以及其他一些必要的汇编指令，构建完成后将shellcode写入到目标进程空间virtual\_address处，然后将目标进程主线程挂起并获得其线程上下文context，将其eip改为virtual\_address，然后将线程上下文设置回去，最后恢复主线程的执行，对方主线程就会调用LoadLibrary函数加载hackdll从而完成注入。

注入四：SetWindowsHookEx注入

同前几种方法一样，依然是获得目标进程的所有线程ID，不同的是这次是自己主动调用LoadLibrary函数加载hackdll得到基地址module\_base，并调用GetProcAddress函数获得hackdll的导出函数Sub\_1(),然后利用for循环将目标进程的所有线程都添加一个Windows鼠标钩子，函数回调传入Sub\_1(),模块基址传入module\_base，这样每次目标进程中出现鼠标事件时，都会调用我们的Sub\_1()函数。

注入五：注册表注入

将hackdll的路径写入AppInit\_DLLs键值中，利用系统加载机制让系统的每个进程都加载hackdll以实现注入目的。

注入六：进程创建期修改导入表

在目标进程创建初期通过其PEB中的ImageBaseAddress成员得到目标进程exe模块的加载基地址，然后ReadProcessMemory将其exe映像读到自己进程中的lpBuffer缓冲区中，pe文件头的大小一般都不会超过0x400，其按照内存粒度对齐后至少会有0xC00的空隙可供我们使用，我们在这一片区域上重建导入表，在原导入表结构末尾添加一个自制的导入表描述，添加的函数是我们hackdll中的导出函数Sub\_1(),并添加上需要的thunkdata，然后修改文件头中关于导入表的相关信息（要把绑定导入的信息清零），最后修改完成后将我们重建的lpBuffer写入目标进程ImageBaseAddress处覆盖掉原来的pe内容，就完成了对目标进程的静态注入。