 一般来说，编译连接之后的代码只能在固定的位置（这里的位置是指偏移地址）上执行，如果直接将其拷贝到其他位置（偏移地址跟编译时的地址不同）上运行时会发生不可预料的错误。 这是因为在汇编语言中对静态变量的寻址通常是用直接寻址方式，这种方式直接使用变量的绝对偏移地址，如果被使用的变量也随代码一起被移动到目标地址，那对该变量的访问将会是对一个无效数据的访问。比如下面这段代码： Org 100H Add SI,SI Mov AX,Var1[SI] Ret Var1 DW 0,1,2,3,4,5,6,7,8,9 它的作用是从字数组Var1中取出SI所指的那个节点的数据给AX并返回。这段程序编译后的代码如下： 0F05:0100 03F6 ADD SI,SI 0F05:0102 2E8B840801 MOV AX,CS:[SI 0108] 0F05:0107 C3 RET 0F05:0100 -00 00 01 00 02 00 03 00 ..... 0F05:0110 04 00 05 00 06 00 07 00-08 00 09 00 ..... 注意红色的数字，它就是数组Var1的绝对偏移地址。此时如果我们把子程序GetData拷贝到偏移地址200H处，可以得到如下代码： 0F05:0200 03F6 ADD SI,SI 0F05:0202 2E8B840801 MOV AX,CS:[SI 0108] 0F05:0207 C3 RET 0F05:0200 -00 00 01 00 02 00 03 00 ..... 0F05:0210 04 00 05 00 06 00 07 00-08 00 09 00 ..... 再次注意红色的数字！此时数组Var1已经被移动到偏移地址208H处，而代码中对数组的访问仍然使用的是编译时的偏移地址。如果该处的数据正好被另的模块更改过的话，后果就 ..... 但是加密软件生成的保护外壳和病毒代码在加到宿主程序中时可以放在任何偏移地址。这就

说明编写能够在任何偏移地址运行的代码并不是不可能，只是对代码有一定的要求。要想写出能在任何偏移地址运行的代码必须要满足几个条件中的任意一条：1.不引用变量，不使用NEAR/FAR跳转或调用 2.只使用动态变量 2.将对静态变量的访问变为相对寻址 不使用变量和NEAR/FAR跳转自然不会生成使用直接寻址的代码，当然可以拷贝到任何地址运行了，不过不能用变量限制太大了吧。 只使用动态变量是很好的方法，所有的变量都存放在堆栈中，不管代码被移动到什么地址都很方便，而且它还带来一个额外的好处：减小代码尺寸。但它也有缺点：就象C语言的子程序头所做的一样，你需要手工计算所用到的变量的总尺寸，运行时从堆栈中为它们留出同样大小的一块，然后为每一个变量设置一个地址（比如：[ESP 4],{ESP 6]之类）。这些工作在高级语言中由编译器代劳，但在汇编中得自己做，实在是太麻烦了。