序的企图都会被在程序运行的时候检测到，因为实际的行为和被签过名的预期行为不一致。

很不幸的是，一个聪明的攻击者可能发动一种叫做模仿攻击（mimicry attack）的攻击，在这种攻击中插入的代码会有和该程序同样的系统调用序列（Wagner和Soto，2002），所以我们需要更复杂的模型，不能仅仅依靠跟踪系统调用。然而，作为深层防御的一部分，IDS还是扮演着重要的角色。

无论如何，基于模型的IDS不仅仅是以一种。许多IDS利用了一个叫做蜜罐（honeypot）的概念，这是一个吸引和捕捉攻击者和恶意软件的陷阱。通常蜜罐会是一个孤立的机器，几乎没有防御，表面看起来令人感兴趣并且有些有价值的内容，像一个成熟等待采摘的果实一样。设置蜜罐的人会小心翼翼地监视它上面的任何攻击并尽量去了解攻击的特征。一些IDS会把蜜罐放在虚拟机上防止对下层实际系统的破坏。所以很自然地，恶意软件也会像之前提到的努力地检查自己是否运行在一个虚拟机上。

9.8.6 封装移动代码

病毒和蠕虫不需要制造者有多大学问，而且往往会与用户意愿相反地侵入到计算机中。但有时人们也会不经意地在自己的机器上放入并执行外来代码。情况通常是这样发生的：在遥远的过去（在Internet世界里，代表去年），大多数网页是含有少量相关图片的静态文件，而现在越来越多的网页包含了叫做Applet的小程序。当人们下载包含Applet的网页时，Applet就会被调用并运行。例如，某个Applet也许包含了需要填充的表格以及交互式的帮助信息。当表格填好后会被送到网上的某处进行处理。税单、客户产品定单以及许多种类的表格都可以使用这种方法。

另一个让程序从一台计算机到另一台计算机上运行的例子是*代理程序*（agent）。代理程序指用户让程序在目标计算机上执行任务后再返回报告。例如，要求某个代理程序查看旅游网站，查找从阿姆斯特丹到旧金山的最便宜航线。代理程序会登录到每个站点上运行，找到所需的信息后，再前进到下一个站点。当所有的站点查询完毕后，它返回原处并报告结果。

第三个移动代码的例子是PostScript文件中的移动代码，这个文件将在PostScript打印机上打印出来。一个PostScript文件实际上是用PostScript语言编写，它可在打印机里执行的程序。它通常告诉打印机如何画某些特定的曲线并加以填充，它也可以做其他任何想做的事。Applet、代理和PostScript是*移动代码*（mobile code）的三个例子，当然还有许多其他的例子。

在前面大篇幅讨论了病毒和蠕虫之后，我们很清楚地意识到让外来代码运行在自己的计算机上多少有点冒险。然而，有些人的确想要运行外来代码，所以就会产生问题："移动代码可以安全运行吗？"简而言之：可以，但并不容易。最基本的问题在于当进程把Applet或其他的移动代码插入地址空间并运行后，这些代码就成了合法的用户进程的一部分，并且掌握了用户所拥有的权限，包括对用户的磁盘文件进行读、写、删除或加密，把数据用E-mail发送到其他国家等。

很久以前，操作系统推出了进程的概念，为的是在用户之间建立隔离墙。在这一概念中，每个进程都有自己的保护地址空间和UID，允许获取自己的文件和资源，而不能获取他人的。而对于保护进程的一部分（指Applet）或者其他资源来说，进程概念也无能为力。线程允许在一个进程中控制多个线程，但是单个线程与其他线程之间却没有提供保护。

从理论上来说，将每个Applet作为独立的进程运行只能帮上一点忙，但缺乏可操作性。例如，某个Web网页包含了相互之间互相影响的两个或多个Applet，而数据在Web页里。Web浏览器也需要与Applet交互，启动或停止它们，为它们输入数据等。如果每个Applet被放在自己的进程里，就无法进行任何操作。而且，把每个Applet放在自己的地址空间里并不能保证Applet不窃取或损害数据。如果有Applet想这样做是很容易的，因为没有人在一旁监视。

人们还使用了许多新方法来对付Applet（通常是移动代码）。下面我们将看看其中的两种方法：沙盒法和解释法。另外，代码签名同样能够用于验证Applet代码。每一种方法都有自己的长处和短处。

1. 沙盒法

第一种方法叫做*沙盒法*（sandboxing），这种方法将每个运行的Applet限制在一定范围的有效地址中（Wahbe等人，1993）。它的工作原理是把虚拟地址空间划分为相同大小的区域，每个区域叫做沙盒。每个沙盒必须保证所有的地址共享高位字节。对32位的地址来说，我们可以把它划分为256个沙盒，每个沙盒有16MB空间并共享相同的高8位。同样，我们也可以划分为512个8MB空间的沙盒，每个沙盒共享9