[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 11/08 (2006.01)
G06F 12/16 (2006.01)

[12] 发明专利说明书

专利号 ZL 02102067.1

[45] 授权公告日 2007 年 4 月 4 日

[11] 授权公告号 CN 1308832C

[22] 申请日 2002.1.21 [21] 申请号 02102067.1

[30] 优先权

[32] 2001. 1.19 [33] EP [31] 01400161.4

[32] 2001. 5. 7 [33] EP [31] 01401170.4

[73] 专利权人 安泰马维尔有限公司

地址 英属维尔京群岛托托拉岛

[72] 发明人 E·多坦

[74] 专利代理机构 北京纪凯知识产权代理有限公司

代理人 程 伟

[56] 参考文献

EP0472487A2 1992.2.26

US6125447A 2000.9.26

审查员 白雪涛

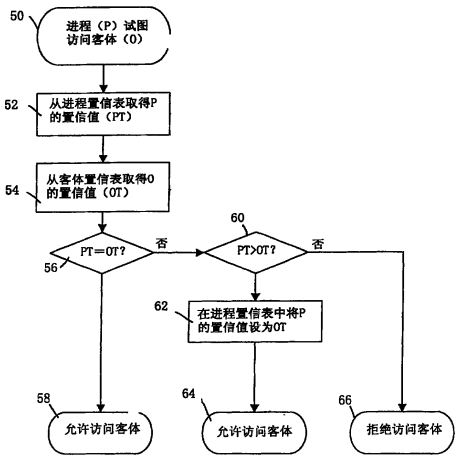
权利要求书 2 页 说明书 18 页 附图 5 页

[54] 发明名称

防御恶意代码而对计算机程序和数据进行保护的方法和装置

[57] 摘要

本发明涉及一种防止不置信程序而对计算机进行保护的方法。 每一个计算机的客体和进程都被分配有置信属性，该置信属性规定了其与系统内的其它客体相互作用的方式。 置信属性是被分层规定的，这样进程即不能访问比其自身具有更高置信水平的客体。 当进程访问具有较低置信水平的客体时，该进程的置信水平将被降低到被访问的客体的置信水平。 具有不同置信水平的进程和客体之间的相互作用是完全可编程控制的。



1. 一种防止恶意代码保护计算机的方法，包括如下步骤：

定义至少两个置信组，每个定义的置信组由一个置信组值表征；

将计算机内的客体和进程分配到所述置信组中的一个内，而不考虑所述计算机的用户的权限；

定义至少两个客体类型；

为每个客体分配一个客体类型；

为每个进程置信组值、客体置信组值和客体类型的结合定义一个动作规则；以及

根据一个请求进程对一个目标客体的访问请求，完成由应用于请求进程的置信组值、目标客体的置信组值和客体类型的动作规则所指明的动作。

2. 如权利要求 1 所述的方法，其中一旦生成了一个进程，该进程即被分配到置信组中，而上述置信组是被分配到产生上述进程的被动代码中去的。

3. 如权利要求 1 所述的方法，进一步包括如果所述进程的置信组值大于所述客体的置信组值，改变所述进程的置信组。

4. 如权利要求 1 所述的方法，进一步包括在完成所述动作之后改变所述客体的置信组。

5. 如权利要求 1 所述的方法，进一步包括一旦一个进程生成了一个客体，即将所述生成的客体分配到所述进程的置信组中。

6. 如权利要求 1 所述的方法，进一步包括定义至少两个操作类型，以及其中所述结合包括至少一个所述操作类型。

7. 如权利要求 1 所述的方法，其中所述置信组是分层排序的，以

及其中所述进程进一步包括：

当在所述分层结构中所述进程的置信组高于或等于所述客体的置信组时，允许所述访问请求。

8. 如权利要求 7 所述的方法，进一步包括如果所述进程的置信组高于所述客体的置信组，则将所述进程分配到所述客体的置信组。

9. 如权利要求 7 所述的方法，其中一旦所述进程重启，所述进程的置信组即恢复到该进程所生成的客体的初始置信组。

10. 如权利要求 1 所述的方法，进一步包括：

定义至少两种进程类型；

将进程分配给所述进程类型之中的一种；以及

其中所述结合包括所述进程类型的至少一个。

11. 如权利要求 1 所述的方法，其中所述客体类型包括被动代码和执行代码。

12. 如权利要求 6 所述的方法，其中所述操作类型包括打开、读取、生成、修改和删除。

防御恶意代码而对计算机程序和数据进行保护的方法和装置

技术领域

本发明涉及一种保护计算机数据和程序的方法，更为概要地说，涉及计算机系统和网络的安全性。本发明有助于防止由恶意代码攻击引起的数据破坏和丢失的发生。

背景技术

在本发明说明书中，计算机是一种包含处理器和存储器的机器，其中处理器能够执行在一个给定的指令集中所选定的指令。由处理器执行的指令序列被称为一个“程序”或者“代码”；当一个程序被存储在计算机的存储器内时，该程序被称为“被动代码（passive code)”。当将其调入处理器加以执行时，它被称为一个“进程”。数据是可由计算机程序以任何方式进行处理或管理的信息；数据也可以被存储在计算机的存储器中。一个网络包括连接在一起的多个计算机。

我们将经过设计或改动、用于有意识地从计算机系统或者网络上破坏或偷窃其上运行的数据或程序的任何代码称为“恶意”或“不友好”代码。由于除了可知晓代码对于用户最终是否有益之外，没有办法在程序上识别正面和负面的程序作用，因此防止不友好代码是一个具有挑战性的问题。例如一个程序可以删除一个文件，因为用户明确地要求该程序如此运行，但是一个恶意程序也可以违反用户的意志而删除一个文件。换句话讲对“恶意”代码或“不友好”代码没有恰当的技术规定——上述“恶意”代码是根据其在计算机上的预期行为而由其合法用户确定的。

虽然有可能利用密码来确认被授权的用户，但置信用户自己也可能无意中运行了包含恶意指令如“病毒”的程序而危及系统和网络的安全性能，上述病毒如为：“特洛伊木马程序”、“恶意宏病毒”、“恶意脚本病毒”、“蠕虫病毒”、“侦探程序”和“后门”。一个计算机病毒是一个通过将其自身附加在其它程序上而进行复制的程序。一个特洛伊

木马程序是一个通常不执行用户期望要进行的程序，而是执行恶意的操作如数据破坏或者系统恶化。宏和脚本是用高级语言编写成的程序，它们可由应用程序如字处理程序进行解释并加以执行，以便能使频繁的任务自动执行。因为许多宏和脚本语言极少需要或者完全不需要用户的交互作用，恶意宏病毒和脚本病毒经常被用来在没有得到用户许可的情况下将病毒或者特洛伊木马程序引入到系统中。蠕虫是一个程序，它像一个病毒一样可以自己扩散。但与病毒不同，蠕虫不会感染其它寄主程序而是通过网络手段如电子邮件将其自己传送给其它用户。侦探程序是特洛伊木马程序的一个子类型，其可被秘密地安装在一个受害的计算机上从而将机密数据和密码由该计算机传送给放置该病毒的人。后门是一个秘密功能，附加到程序上从而允许该程序的作者闯入或对其滥用，或者一般来说为其自身利益而利用该功能。

所有上面的程序都无视用户的意志，可通过破坏数据、从一个文件传播到另一个文件或者将秘密数据传送给非授权人而危及计算机系统和公司的机密性。

多年来，产生了不同的技术来防止恶意程序从而对计算机系统进行保护：

特征扫描程序通过使用一个预规定的“公知病毒”清单来检测病毒。他们对每一个文件进行扫描来寻找列在其公知病毒数据库之内的每一个病毒特征。每当在世界上任何地方发现一个新病毒时，都将该病毒添加到数据库中。然而目前每一天都要产生越来越多的新病毒，为了使公知病毒清单有效要不断地对其进行更新。定期升级反病毒软件对于单用户和网络管理员都是一个沉重的负担，并且在软件升级之间留下了重要的安全空缺。

另一种通常被称为试探式扫描的检测方法包括对程序进行扫描来找出可疑指令，上述可疑指令为恶意程序和特定的病毒的典型，而无须为了在文件中检测到病毒而拥有每一个病毒的精确的特征。然而恶意程序的作者可以通过不同方式书写代码和/或对其进行加密而避免或隐藏这些典型指令，这样很快即可避免恶意代码和病毒为试探式扫描程序所检测到。

Mann 的美国专利 US5408642 和美国专利 US5349655 以及 Kephart

等人的美国专利 US5613002 中都公开了用于恢复受病毒感染的计算机程序的方法。所公开的这些方法包括在受到病毒感染之前生成一个数据指纹并将该指纹进行存储。然后生成第二个数据指纹并将其与在前数据串进行比较，从而确定出该数据是否被病毒破坏并将数据恢复到其初始状态。这些技术不能防止病毒感染，而且它们不能防御其它类型的恶意程序。

本人的美国专利 US6073239 公开了一种方法，其中对文件的 I/O 活动进行过滤。每当一个程序试图感染或者将代码引入到另一个程序文件中时，都将对其加以拒绝。然而该方法仅被设计成适合于防御可执行文件型的病毒。它不对其它类型的病毒如宏病毒以及其它类型的恶意程序如蠕虫程序、特洛伊木马程序、后门或侦探软件起作用，因为这些恶意程序既不引入代码也不修改其它程序，而是直接进行恶意动作如破坏数据。

Jablon 等人的美国专利 US5421006 公开了一种在系统初始化阶段评定计算机系统软件完整性的方法。在允许其执行之前对启动进程加以校验。然而该方法既不能防止受保护的进程在最初即被破坏，它也不能处理数据和程序——除非它们与系统启动相关。

其它安全性方法要点在于确保得到授权的程序运行而阻止所有其它未经授权的程序。然而遗憾的是这些技术并非总是适合于开放系统，在这种开放系统中用户会接收和交换许多文件。

一种通用的安全系统要点在于建立访问控制清单（即 ACL、DACL），上述访问控制清单规定了限制和权限，从而根据这些用户的权限而允许用户访问或者不允许访问某些特定资源。例如通常允许系统管理员修改任何文件而同时一般用户既不能阅读也不能修改某些机密或关键性文件。上述安全系统通常被集成到现代的操作系统中从而可确保单用户基础（per-user basis）上的数据的安全性和机密性。然而重要的是区分并理解，该安全配置是被设计来处理用户信用的问题，而不是代码信用的问题。在其系统之内运行恶意程序的用户会在无意中损害各资源和文件的完整性，而他们被允许访问这些资源和文件并无进一步保护措施。例如我们说用户 X 被赋予了访问共享文件 A、B 和 C 的全权。如果该用户运行了一个受病毒感染的程序，病毒将可以

阅读、感染甚至破坏文件 A、B 和 C。这是由于事实上访问控制清单即如此设计，使得程序和任务运行于启动它们的用户的安全环境(security context)中。这样即使用户实际不想危害文件 A、B、C，然而根据用户的权限，他所运行的程序确实对这些文件造成了危害——无论该用户意愿如何。这就是恶意代码问题的核心。如果一个用户运行了不友好代码，该代码将可以破坏和偷窃其用户访问的系统或网络之内的任何数据。如果系统管理员运行了一个不友好代码，整个系统和网络将会立即遇到危险。除了这些安全性问题外，访问控制清单是为每一个文件和资源静态确定的。在文件每天都被共享和交换的环境中，由于用户通常不会花时间为每一个产生或接收的新文件分配权限安全属性，这样就不会具有防御恶意代码的足够的安全性能。这样的系统公开于以下文献中：欧洲专利 EP-A-0 472 487 或者《计算机通讯》杂志第 13 卷第 9 期第 571-580 页（Computer Communication vol. 13 no.9 pp. 571-580）J. Moffett 等人所著论文《确认分布式系统的任意访问控制的策略》（Specifying Discretionary Access Control Policy for Distributed System）。

“沙箱”技术能对可疑程序进行测试，它是使这些可疑程序安全地运行在一个安全的“沙箱”环境中而不会使被测试的程序对系统或其文件造成危害。然而因为例如这些程序侦测到他们正在被测试或者它们被设计成随机的或在某些日期执行攻击性活动，所以在测试期间这些恶意程序可能不会立即执行攻击性或者期望的活动。因此在测试期间一个程序看起来运行正常，但是一旦它通过测试并允许进行实际的运行，该程序可能会对系统造成危害。而且由于正面程序（positive program）也许基于正当的原因而需要访问系统内的文件和资源，因此正面程序可能在沙箱中表现不正常或完全不能运行。

Chambers 的美国专利 US5398196 公开了一种在虚拟处理器中模仿程序运行的方法，并同时搜索病毒的行为特征。该方法的缺点与上述的各种方法的缺点是一样的。

发明内容

本发明的目的是提供一种防御不友好代码而对计算机上数据和程

序进行保护的方法。它为在计算机运行的任何阶段都提供了保护并且保护计算机不受任何类型的不友好代码的破坏。本发明特别提供了一种防御不友好代码保护计算机内客体的方法，其包括如下步骤：

规定至少两个置信组；

将计算机内的客体和进程分配到所述置信组中的一个中，而不考虑所述计算机的用户的权限。

根据一个进程对一个客体或第二个进程的操作，将该进程的一个置信组与该客体的一个置信组或者与该第二进程的置信组相比较；

根据所述比较步骤的结果确定是否允许进行该项操作。

在一个实施例中，一旦产生了一个进程，该进程即被分配到置信组，而上述置信组是被分配到产生上述进程的被动代码（passive code）中去的。在所述运行之后也提供了一个改变进程和/或客体的置信组的步骤。因此一个解决方案是当允许进行操作时，将进程分配到客体或第二进程的置信组中。

该进程可进一步包括当一个进程产生了一个客体时，即将所述产生的客体分配到所述进程的置信组中的步骤。

在一个优选实施例中，置信组是分层有序的并且允许步骤包括：

当所述进程的置信组在所述分层结构中高于或等于所述客体或所述第二进程的置信组时，允许进行所述操作，

当所述进程的置信组在所述分层结构中低于所述客体或第二进程的置信组时，拒绝所述操作。

在操作得到允许之后，还可以提供将进程分配到一个客体的置信组或第二进程的置信组中的步骤。

也可以规定至少两种类型的客体，客体被分配到所述类型中的一个中；然后根据所述进程的类型，允许实施在一个客体上进行的操作的步骤。也可以规定至少两种类型的操作；根据所述进程的类型，允许实施在一个进程上进行的操作的步骤。也可以规定至少两种类型的操作；然后根据所述操作的类型，允许实施在一个客体上进行的操作的步骤。进一步而言也可以规定至少两种类型的存储方法并将一个置信组分配到一种类型的存储方法中；然后根据被分配到所述进程的置信组的存储方法，实施一个置信组的进程的存储操作。

本发明进一步提供了一个计算机，包括：

客体和进程；

至少两个置信组的一个表，该计算机内的客体和进程被分配到所述置信组中的一个中而不考虑所述计算机用户的权限；

一个控制器，根据所述进程的置信组与所述客体或所述第二进程的置信组的结果，该控制器可访问所述表并允许进程在一个客体或一个第二进程上进行一个操作。

该计算机进一步包括具有至少两种类型客体的一个类型表，该计算机中的客体被指定为一种类型；在这一实例中，该控制器访问所述表格从而确定是否允许进行一个操作。该置信组的表优选地被存储在一个非易失的存储器中，该类型表也被存储在一个非易失存储器中。

该计算机还可包括一个规则表；于是该控制器访问该规则表格来确定是否允许进行所述操作。非常便利的是该规则表被存储在一个非易失存储器中。

最后本发明提供了一种计算机网络，该计算机网络包括一个服务器和至少一个这样的计算机；然后将置信组表存储在所述服务器之内。本发明还提供了一种计算机网络，该计算机网络包括一个服务器和至少一个这样的计算机，然后将类型表存储在所述服务器之内。本发明还提供了一种计算机网络，该计算机网络包括一个服务器和至少一个这样的计算机；然后将规则表存储在服务器之内。

附图说明

结合下面的附图说明，本发明的其它特点和优点会变得更为清楚，其中：

图 1 示意性的展示了被动代码和一个进程；

图 2 是一个将进程分配给一个置信组的流程图；

图 3 是一个将新客体分配给一个置信组的流程图；

图 4 是一个实施上述进程的计算机的示意图；

图 5 为根据本发明的一个流程图；

图 6 为一个进程在一个客体上进行操作的流程图；

图 7 为一个规则应用的流程图。

具体实施方式

本发明是基于这样的认识：不友好程序违反用户意志的作用是由于程序是利用了与执行该程序的用户相同的访问特权来运行的。这样用户可访问的任何客体对于该用户运行的程序而言也是可访问的—因此在用户有意或无意中运行了这样的程序，该不友好程序即可对上述客体进行访问。

因此本发明建议规定置信组并将计算机中的客体分配到置信组中的一个内。然后根据进程的置信组与该进程在其上进行操作的客体的置信组的比较结果，允许或者不允许一个进程在一个客体上进行的任何操作。

本发明的方法不同于上面讨论的所有的现有技术。本发明不需要对文件或程序进行扫描来检测公知的特征；进程的操作与用户无关：本发明将置信值分配给计算机中的程序和客体，而不是将权限分配给用户并使该用户进行任何操作。

在一个计算机中，我们将计算机处理器可以解释或执行的任何指令集称为“代码”或“程序”。当将代码被动地存储在一个存储介质中而没有被执行时，我们将其称之“被动代码”。当代码在存储器中运行时，这是一个“进程”。将一个程序从存储介质调入到存储器（如 RAM）中的过程被称为“执行”，其中上述程序作为被动代码而被存储在上述存储介质内，上述程序作为一个主动进程（active process）而在上述存储器中运行。图 1 展示了代码执行机制的一个例子：此处被称为“WORD.EXE”的一个文件 2 代表了被动代码；该文件被存储在计算机的存储设备如一个硬盘中。当计算机指示对其进行执行时，操作系统（OS）将该文件的一个映象拷贝到存储器（RAM）中并由此处对其进行运行。在此阶段，由于“WORD”正在运行，所以它是一个进程 4。注意虽然文件“WORD.EXE”和在存储器中的进程“WORD”可具有相似的内容（因为一个是最初由另一个复制而来的），然而它们也是两个不同的实体：可对一个进行修改而对另一个无影响。另外，计算机用户可以多次执行“WORD.EXE”并具有同时运行但相互独立的多个不同的“WORD”进程。此外进程可以执行被动代码并由此产生其它进程。如果一个进程 A 产生了一个进程 B，进程 A 被称为“父进程”，

进程 B 被称为“子进程”。

我们将需要由本发明进行保护或包含了可由操作系统或任何进程解释的代码的任何文件、被动代码、文档、脚本、宏、进程、注册码或其它系统客体称为“客体”。因为包含代码的客体可以被执行并被用于恶意目的，所以优选地对其进行保护。

如此规定进程和客体情况下，进程建议规定置信组并将客体和进程分配到置信组中的一个之内；因此一个置信组就是进程和客体的一个集合；下面讨论将客体和进程分配给置信组的方式；需注意的是置信组是独立于计算机用户的权限的。换句话讲，置信组与现有技术中规定的权限是不同的。

为了便利起见，置信组可以简单地由一个为置信组的所有元素所共有的置信值所代表。该置信值可表示为一个数值或者以任何其它表现方式来表示。在最简单的实例中规定了两个置信组；然后置信组可由 0 和 1 来表示——这些值中的一个表示“置信”，另一个表示“不置信”。如下面举例所示，置信值也可以为 0 到 10 之间的一个数值；这样可以规定 11 个置信组之多。为了将客体和进程确定到置信组中的一个之内，可以使用一个置信表，它代表了分配到置信组中的一个之内的每一个进程或客体的置信组。可提供两个置信表，分别用于存储客体和进程的置信组。实际上可以很便利地将客体的置信表记录在一个非易失存储设备如硬盘或者一个远程服务器之上；另一方面，因为进程基本上仅在 RAM 内是激活的（active），所以进程的一个置信表可被存储在 RAM 之内。

这样客体和进程被分配给置信组，一个进程在一个客体上的任何操作—例如修改或者删除一个现有客体—可导致进程的置信组与客体的置信组进行比较。然后根据比较步骤的结果允许或不允许进行操作。

上面公开的进程防御了恶意代码而对计算机内的客体进行了保护。假定规定了两个置信组，一个包含“置信”客体和进程，另一个包含“不置信”客体和进程。如下所释，置信客体和进程可以被认为是友好的例如商业代码或者其它代码。可以根据置信组而允许或不允许一个进程在一个客体上的操作。例如在最简单的实施例中，唯一被禁止的操作即是一个“不置信”进程在一个“置信”客体上的操作。

该实施例是粗略的；仍有可能对计算机的操作进行限制，这样就不会有恶意代码对计算机进行作用的危险。实际上这种代码应被分配到“不置信”组——因为其起始源可能没有被确认。与现有技术的权限系统相比，该进程还可以防御用户的操作而对计算机进行保护；因为进程和客体独立于用户被分配到置信组中，即使用户拥有权限，他也不会危及计算机的安全。

如上简述，可通过将客体分配到各种置信组中而开始操作。根据赋予这些客体的置信度而将客体分配到置信组中。例如从计算机销售商处购买的软件通常比从不知名的互联网 Web 和 FTP 站点处下载的软件要具有较高的置信度。在安装计算机时复制的软件也可以被认为是可置信的并可相应地被分配到置信组中；后来安装的客体应被分配到表示较低置信度的置信组中。可由一个计算机管理员来实施将客体分配到置信组的工作；该管理员负责计算机的安全。在网络环境下，他通常是网络管理员。在单用户环境下，他可以是机器用户。

也可以采用相同的方法将进程分配到置信组中；然而很明显这样很耗时间并且不是非常有效。因此图 2 展示了一个将进程分配到一个置信组的流程。如上所述，进程由被动代码产生；因此将一个进程分配到一个置信组的一个解决方法是将该进程分配到置信组，而上述置信组是被分配到由此产生进程的被动代码中的。如图 2 所示的实施例，在一个客体置信表和一个进程置信表中规定了置信组。在步骤 8 中执行被动代码，这样在 RAM 中产生一个进程。在步骤 10 中，对客体置信表进行查询从而确认出将被动代码分配到的置信组。在步骤 12 中，该进程被分配到相同的置信组中；在进程置信表中对此做出标记。图 2 中的进程可使得将进程简单且快速地分配给一个给定置信组。

图 3 示出了将新客体分配给一个置信组的流程图：简单而言，一个新客体被分配给产生该新客体的进程的置信组中。在步骤 20 中，一个被分配给一个给定置信组的现有进程产生了一个新客体。在步骤 22 中，对进程置信表进行查询从而确认出将该进程分配到的置信组。在步骤 24 中，新客体被分配给相同的置信组；在客体置信表中对此做出标记。图 3 中的方法可使得将新客体简单且快速地分配给一个给定置信组。也可能不在每一个实例中都进行上述进程：这样将在正常使用

计算机的条件下对该方法进行实施；当将新客体安装在计算机上时实施另一个方法。也可仅允许管理程序使用不同于图 3 中的方法来将客体分配给置信组。而且这不同于现有技术中的方法，因为无论计算机的用户是谁，客体都将被分配给一个置信组。换句话讲，一旦将一个客体分配给一个置信组，该客体的分配将应用于任何将来的用户，可以是程序管理员或其它用户。

在本发明的一个优选实施例中，可以改进允许进行操作的规则；如上所述，将置信组的比较结果作为唯一的根据允许一个进程在一个客体上进行操作。也可以规定不同类型的客体、操作或进程；考虑到客体、操作、进程或存储方法的类型而允许或实施一个操作。对于客体而言例如可以区分被动代码和其它客体，并根据客体的类型允许进行不同的操作。这样将会对某种类型的客体提供进一步的保护。对于进程而言根据用户规定的进程类型可以对进程进行区分。对于操作而言可以区分打开、读、产生、修改或删除。而且这样可使得一些操作——例如那些被认为更具有冒险性的操作——与其它操作相比更难以实施。还可以规定不同的存储方法——例如使用加密和/或压缩，并将一个置信组分配给一个给定的存储方法；在此情况下，置信组中的进程将使用该置信组被分配到的存储方法。这样有可能根据授予进程的置信度等级使用各种存储方法。这样通过加密即可对置信信息进行保护——该信息由一个置信进程产生。

至此上面描述了防御恶意代码保护计算机的方法。然而它对计算机的操作干预太多——可以理解的是任何防御恶意代码的保护措施都会干预计算机的操作，并且将会限制被允许进行的操作的范围。现在讨论一个更为灵活的例子。在该例子中，如果允许进行一个操作，则进程和/或客体的置信组是可以改变的。这样有可能扩展操作的范围，并且同时仍能对计算机进行保护；换句话讲，它可以改善对计算机的保护。在该例子中也具有两个置信组。允许进行操作的规则可包括如下规则：

允许一个被分配到“置信”组的进程在该“置信”组的一个客体上进行操作；

允许一个被分配到“置信”组的进程在“不置信”组的一个客体

上进行操作，然后将该进程分配给“不置信”组。

也规定了用于其它例子下的其它规则。例如假定进程是上面讨论的 WORD 进程；还假定该进程位于“置信”组内。WORD 进程可以访问一个置信客体例如一个置信文件。当 WORD 进程访问一些不置信客体如从互联网服务器上下载的新文件时，允许进行操作。然而该进程由置信组转到了不置信组。这样可以确保可能已经被不置信客体破坏的进程不会对置信组内的客体造成损害。这样增加了可能进行操作的数量同时仍对计算机进行了保护；保护措施也得到了改善。

为了清楚起见，上面讨论的所有特性是分别进行论述的；人们应该理解的是这些特性或者它们中的一些是可以进行组合的。

图 4 为实施本发明的一个计算机的示意图。计算机 30 包括非易失存储器 32 如一个硬盘、一个软盘或其它。该计算机具有一个与 RAM 35 相联系的处理器 34；处理器 34 与非易失存储器 32 和 RAM 35 相连接并可对其进行访问。在存储器 32 之内还具有一个客体类型的表 40 以及规则表 42。RAM 35 包含多个激活的进程 44、一个进程置信组表 46 以及一个进程类型表 47。另外在 RAM 35 之内具有当一个进程在一个客体上进行操作时可访问置信组表的一个控制器 48。控制器根据进程的置信组和客体的置信组的比较结果来允许进行操作。该控制器可独立于操作系统的类型被设计成在操作系统级进行工作且以一个驱动程序（Driver）、VxD 或 TSR 方式加以实施。

现在我们对本发明的另一个实施例进行描述，其中置信组是分层有序的。这样的次序使得根据置信组的比较结果，可更为简单地设计出允许或拒绝进行操作的规则；然后置信组可简单地被称为与该置信组相联系的置信值。在本说明书的剩余部分中术语“置信组”和“置信值”是相当的。例如置信组的秩序可如下进行选定：客体的置信值越高，可访问性越低并且受保护程度越高，反之亦然。在该例子中，置信值是 0 到 10 中的一个数值。一个新机器的客体具有高的置信值；一个计算机的置信值随着时间会降低，因为要执行新的软件和文件并且要在其上进行打开。这样在安装新机器时计算机程序管理员典型地将所有客体设定为具有高的置信值。任何后来安装或引入计算机的客体将被默认为具有一个低的置信值，除非程序管理员明确的指定为其

它值（例如当安装一个新的商业软件时）。

客体置信表

这样每一个系统客体最初具有一个程序管理员规定的置信值。所有的客体置信值被记录在一个“客体置信表”中，该客体置信表被存储在一个存储介质如一个硬盘或者数据库服务器上。任何被添加到系统内的客体将明显的没有列在最初设定的客体置信表之内，这样所有没有存储在客体置信表内的客体将自动被默认为具有一个低的置信值。例如：当机器处于其原始清洁状态（original clean state）时，程序管理员将置信客体标记为一个高的置信值；在该时刻之后从外部接收的任何客体将自动被分配一个低的置信值，除非程序管理员明确指定为其它值。

进程置信表

与客体相似，进程也被分配有置信值。如上所述，进程置信值也随着时间而降低。实际上调入或解释代码的被置信的进程本身可以被操纵来执行恶意操作。例如若一个最初为置信状态的进程打开了一个带有宏的文件而该宏指示上述文件进行侦探或破坏数据，该进程会变得不友好。进程置信值将被存储在称为“进程置信表”的一个第二个表中。该表被存储在一个存储介质如 RAM（随机存取存储器）中，并且可以在每次加载或重起操作系统时对该表进行清空，因为在该点处没有运行进程。当在执行阶段期间产生一个进程时，该进程得到与加载该进程的被动代码相同的置信值。例如当被动代码“WORD.EXE”被标记为一个置信值 10，然而无论何时执行 WORD.EXE，由此得到的新进程 WORD 也将被初始化为具有置信值 10。图 2 展示了一个进程如何根据加载该进程的客体而得到一个置信值。

客体的生成/修改

如上参考图 3 所述，当一个进程生成一个新客体时，该客体得到与该进程相同的置信属性。当一个进程对一个现有的客体的内容进行修改时——允许该进程在该现有客体上进行操作——客体的置信组被修改为进程的置信组；这样就能将一个进程对一个客体可能造成的损害纳入考虑范围。

进程和客体之间的相互作用

图 5 为本发明的例子的进程的流程图。进程可以访问具有与其本身相同的置信值的客体。换句话讲，置信值为 5 的一个进程可以访问一个置信值为 5 的客体。然而进程不能访问比其自身的置信值高的客体。例如一个置信值为 5 的进程不能访问置信值为 10 的客体。

当一个置信进程访问一个具有较低置信值的客体时，在进程置信表中的该进程本身的置信值会立即改变并变为与被访问的客体的置信值一样低，这样一直保持下去直到该进程被中止。这样在下一次当进程试图访问一个客体时，该进程与其访问不置信客体之前相比具有了较低的权限。例如一个置信值为 10 的进程被允许访问置信值为 10 的客体。但是如果该进程打开了一个置信值为 9 的客体，该进程的置信值将降低为 9 并且从那时起将不允许该进程访问置信值大于 9 的客体，直到该进程被中止或者重新启动。这样就考虑了如下事实：一个访问客体的进程可能被该客体所损害；降低进程的置信值将会避免置信客体遭到损害。

图 5 展示了进程和客体之间的相互作用的方式。在步骤 50 中，一个进程（P）试图去访问一个客体（O）。在步骤 52 中，我们对进程置信表进行查阅，寻找匹配进程 P 的表项。我们将结果存储在 PT 之内。在步骤 54 中，我们对客体置信表进行查阅，寻找匹配客体 O 的表项。我们将结果存储在 OT 之内。在步骤 56 中，对 PT 和 OT 进行比较。如果这两个置信值相等，则在步骤 58 中允许进行访问。返回到步骤 56 中，如果 PT 和 OT 不相等，我们在步骤 60 中进行测试看进程的置信值是否高于客体的置信值。如果在步骤 62 中是这样，进程置信表中的进程的表项被设为 OT，然后我们继续进行步骤 64，在步骤 64 中可以对客体进行访问。注意在下一次进程 P 试图访问一个客体时，除了该进程的置信值不同外，也将经历相同的决策进程，因为在步骤 62 中该进程的置信值已经被改变。返回到步骤 60 中，如果进程的置信值低于客体的置信值，则我们继续步骤 66，在步骤 66 中不允许访问客体。

本发明的具体示例

下面的表提供了在不同情况下进程的具体示例。

表 1

顺序	系统完成的操作	控制器执行的操作	进程置信值
1	执行置信值为 10 的 WORD.EXE	将新产生的进程标记成与 WORD.EXE 相同的置信值：10	10
2	WORD 进程打开置信值为 10 的客体 INFO.DOC	允许	10
3	WORD 进程产生一个新客体：NEWFILE.DOC	将新产生的客体标记为与进程相同的置信值：10	10
4	WORD 进程打开置信值为 5 的客体 MACRO.DOC	将进程置信表内的 WORD 进程置信值变为 5 并允许对该客体进行访问	5
5	WORD 进程试图打开置信值为 10 的客体 INFO.DOC	拒绝访问该客体	5
6	WORD 进程产生一个新客体：NEWFILE2.DOC	将新产生的客体标记为与进程相同的置信值：10	5

如参考图 5 所述，表 1 中的例子展示了允许进程在客体上进行操作的规则。

表 2

顺序	系统完成的作用	由本发明执行的操作	进程置信值
1	执行其置信值为 1 的一个客体：VIRUS.EXE	将新产生的进程标记成与 VIRUS.EXE 相同的置信值：1	1
2	VIRUS 进程试图访问置信值为 10 的客体 C:\WINDOWS.EXE	拒绝访问该客体	1
3	VIRUS 进程产生一个新客体:PAYLOAD.EXE	将新产生的客体标记为与进程相同的置信值：1	1

这样利用该方法我们可以确保计算机的安全，同时仍允许进行许多操作。现在我们描述该方法的更为灵活和快捷的另一个实施例。

置信组

除了如通过规定置信组为每一个客体提供一个置信值之外，也为置信组内的客体和进程规定了额外的信息和性能。利用一个置信组标识符将每一个客体和进程与一个“置信组”联系在一起。下面的表是进程置信表的一个例子，其中利用一个置信组标识符将由操作系统提供的进程 ID（PID）标识的每一个进程与一个置信组联系在一起。

表 3

系统规定的 PID	100	110	120	130	140
置信组标识符	1	1	2	1	3

置信组规定可被记录在一个被称为“置信组表”的中心表中，该置信组表被存储在一个中心存储介质中（例如文件服务器、硬盘或者非易失存储器中）。每一个置信组可具有如下表所示的数据结构。

表 4

置信组（TG）
ID：置信组标识符
TV：置信值
FromLower：规则清单
ToLower：规则清单

ID 是一个被记录在客体置信表和进程置信表内的索引值，它标识出了客体和进程所属的置信组。TV 是组的置信值。FromLower 是一个规则清单，当其置信组的置信值低于 TV 的进程做出指向属于 TG 的客体的请求时，应用上述规则。ToLower 是一个规则清单，当属于组 TG 的一个进程对其置信值低于 TV 的客体做出请求时，应用上述规则。下面的表是具有三个不同置信组的置信组规定的一个例子。

表 5

置信组 1
ID： 1
TV：8
FromLower 规则：A
ToLower 规则： B

表 6

置信组 2
ID： 1
TV：8
FromLower 规则：C
ToLower 规则： D

表 7

置信组 3
ID： 1
TV：8
FromLower 规则：E
ToLower 规则： F

一个“规则清单”将多个规则节点连接在一起。每一个规则节点如表 8 所示。

表 8

规则（R）
OP：操作
OT：目标客体类型
A： 要执行的操作

OP 指示出该规则应用于何种类型的 I/O 操作。例如 OP 可以为：打开、读、写。OT 为要对其应用该规则的客体的类型。例如 OT 可以为：执行文件、文档文件和注册码。客体也可以被应用于对某些特定置信组需要被限制到进程中的操作，例如指示“联网 API”为一种客体类型就使得禁止对不置信进程使用联网操作。当 OP 和 OT 请求相匹配时，A 是要执行的操作。例如 A 可以为：“拒绝”、“允许”、“将请求进程的置信值设为目标客体的置信值”和/或“将目标客体的置信值设为进程的置信值”。根据它与其它进程和客体的相互作用，这些最后两个作用可以动态的改变一个进程或一个客体的置信组。表 9 示出了多个规则清单的一个例子。

表 9

	规则 A	规则 B	规则 C	规则 D	规则 E	规则 F
	规则#1	规则#1	规则#1	规则#1	规则#1	规则#1
OP:	读、写	读	打开	打开	写	打开
OT:	注册码	文档文件	注册码	图像文件	任何文件	文本文件
A:	拒绝	拒绝	拒绝	允许	拒绝	允许

	规则#2	规则#2	规则#2	规则#2	规则#2	规则#2
OP:	打开	打开	读	打开	读	打开
OT:	注册码	可执行文件	可执行文件	任何文件	文档文件	任何文件
A:	拒绝	拒绝	允许	拒绝	拒绝	允许

	…	…	…	…	…	…

现在我们来看这个新的实施方案是如何工作的。图 6 展示出了一个流程图，其中如步骤 70 所示一个进程（P）试图访问一个客体（O）。在步骤 72 中我们对进程置信表进行查阅，寻找匹配进程 P 的表项。在大多数操作系统中，通过其自己的进程 ID（PID）可对进程进行识别。与进程相联系的置信组 ID 被存储在 PTID 中。在步骤 74 中我们对置信表进行查阅，寻找其 ID 与 PTID 相匹配的置信组。在步骤 76 和 78 中，我们重复进程正试图访问客体的相同的操作。在步骤 76 中，我们查询客体置信表内的客体置信组并将其存储在 OTID 之内。在步骤 78 中，我们查询其 ID 与 OTID 相匹配的置信组的置信组表，我们将进程的置信组的置信值字段与客体的置信组的置信值字段相比较。如果进程的置信组具有较高的置信值，我们继续进行步骤 82，在步骤 82 中我们应用了来自进程的置信组 ToLower 字段的规则。返回步骤 80，如果进程的置信组不具有比客体较高的置信值，我们继续步骤 84，在步骤 84 中我们检查进程置信组是否具有一个较低的置信值。如果是这样，我们继续步骤 86，在步骤 86 中我们应用了来自目标客体的 FromLower 字段的规则。返回步骤 84，如果进程的置信组和客体的置信组具有相等的置信值字段，则在步骤 88 中允许对客体进行访问。

图 7 展示了如何应用置信组的规则清单字段内的一个规则。在步骤 90 中，我们参考需要被应用的规则清单。在 IOOP 中，我们有请求施加于目标客体（O）上的操作。对每一个规则清单内的规则节点，执行循环中的步骤 92 到步骤 98。在步骤 96 中，我们检查当前规则的 OP 字段是否包含被请求的操作 IOOP。例如可以利用一个位组合来完成这样一个检查，其中每一个可能的操作由一个位表示，整个数字表示整个操作组。如果检查结果不是肯定的，我们继续步骤 92 中的下一个规则。相类似，在步骤 98 中，我们检查当前规则的 OT 字段是否包含客体 O 的类型。客体的类型例如可以为：可执行文件、文档文件、注册码等。如果不包含，我们继续步骤 92 中的下一个规则。如果两个条件都满足，在步骤 100 中该进程结束，在步骤 100 中被应用的进程得自规则节点的 Action 字段。如果没有发现与所需的条件相匹配的规则，在步骤 102 中结束该进程，其中默认允许进行该操作。

为了进一步提高灵活性，规则节点也可具有一个“优先规则权”

字段。它允许在两个具有相同置信值的不同置信组之间进行分解检查（resolving test）：服从于具有较高规则优先权的组。如图 7 所示，这样也有利于特定的实例：需要应用具有最低置信值的置信组的规则，而不是需要应用具有最高置信值的置信组的规则。

因此现在有可能建立起来一个安全的环境，其中多个置信组对具有不同安全属性的不同的客体进行保护。本发明用于在操作系统级工作，过滤 I/O 操作。它通常独立于操作系统的类型且以一个驱动程序、VxD 或 TSR 方式来实现。

可以改变上面给定的例子：置信组可以被预规定和/或是可配置的。该进程可应用到危及系统安全的 I/O 操作以及 API 函数调用或者任何触发程序的作用中。

图1

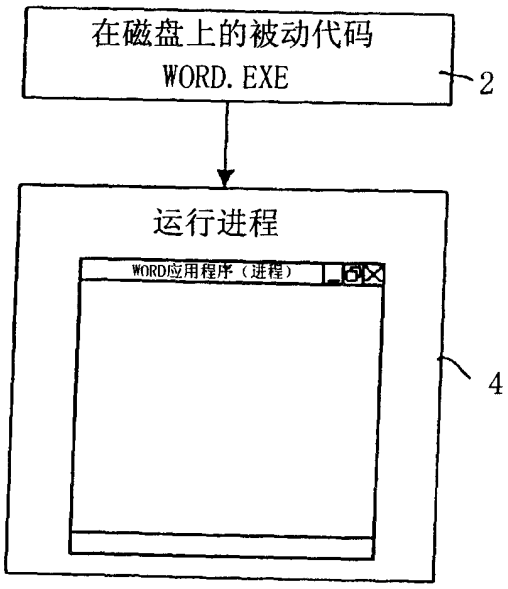

在磁盘上的被动代码
WORD.EXE
2
运行进程
WORD应用程序（进程）
4

图2

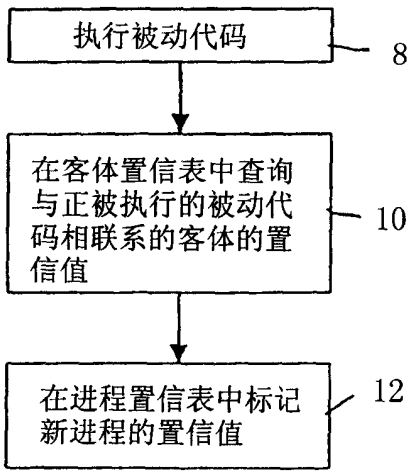

执行被动代码
8
在客体置信表中查询与正被执行的被动代码相联系的客体的置信值
10
在进程置信表中标记新进程的置信值
12

图3

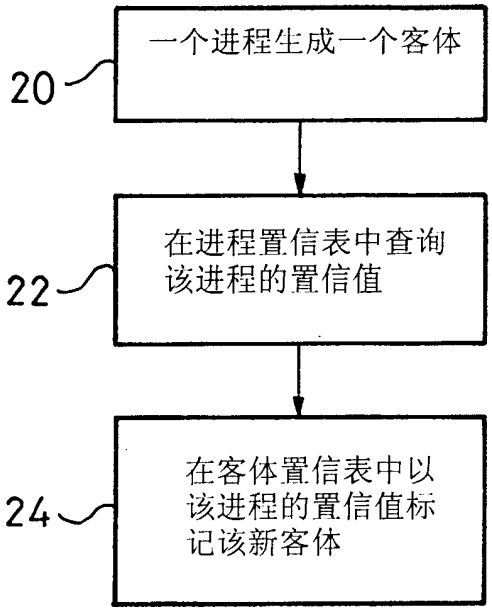
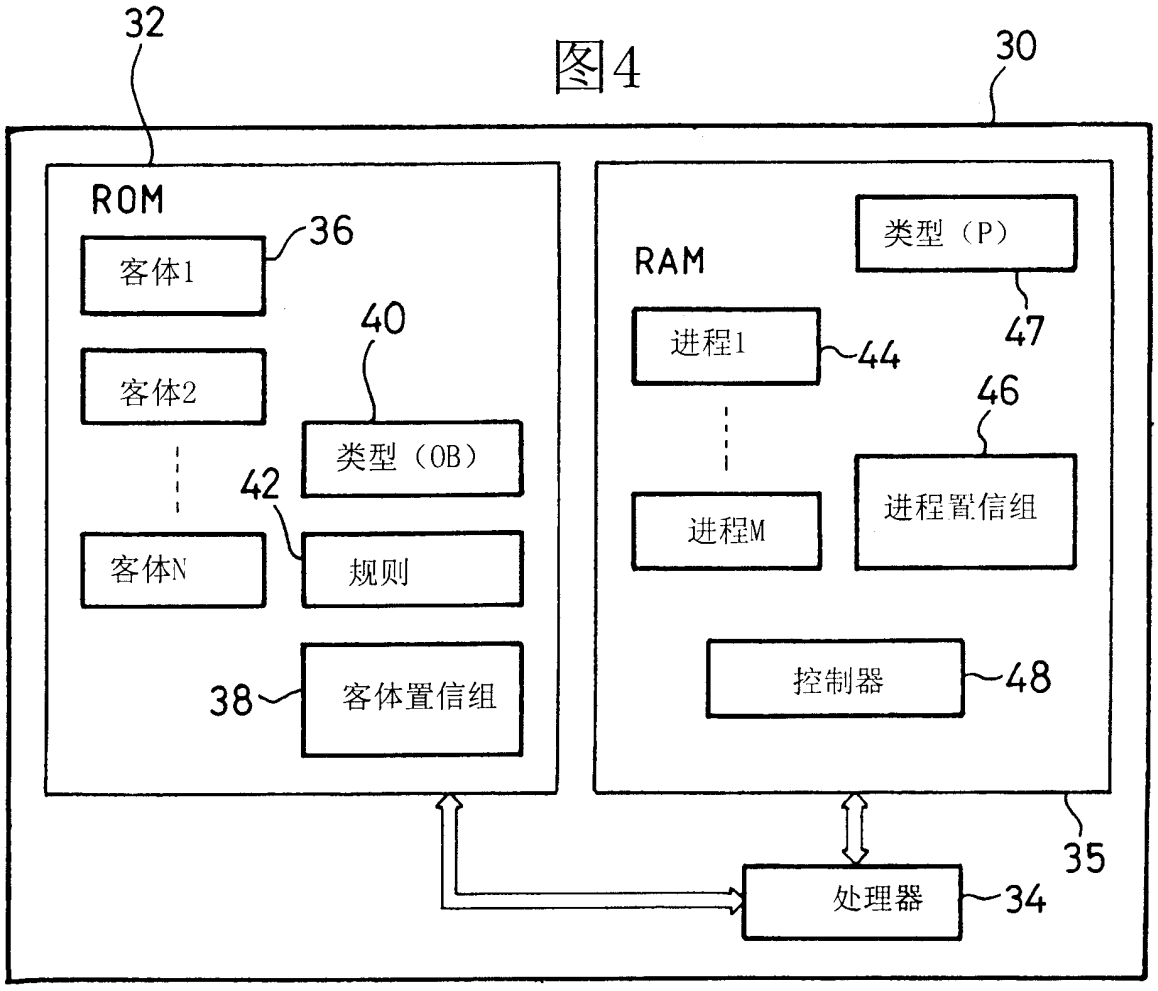

一个进程生成一个客体
20
在进程置信表中查询
该进程的置信值
22
在客体置信表中以
该进程的置信值标
记该新客体
24

图4


32
30
ROM
36
客体1
客体2
客体N
40
类型（OB）
42
规则
38
客体置信组
RAM
类型（P）
47
进程1
44
进程M
46
进程置信组
控制器
48
35
处理器
34

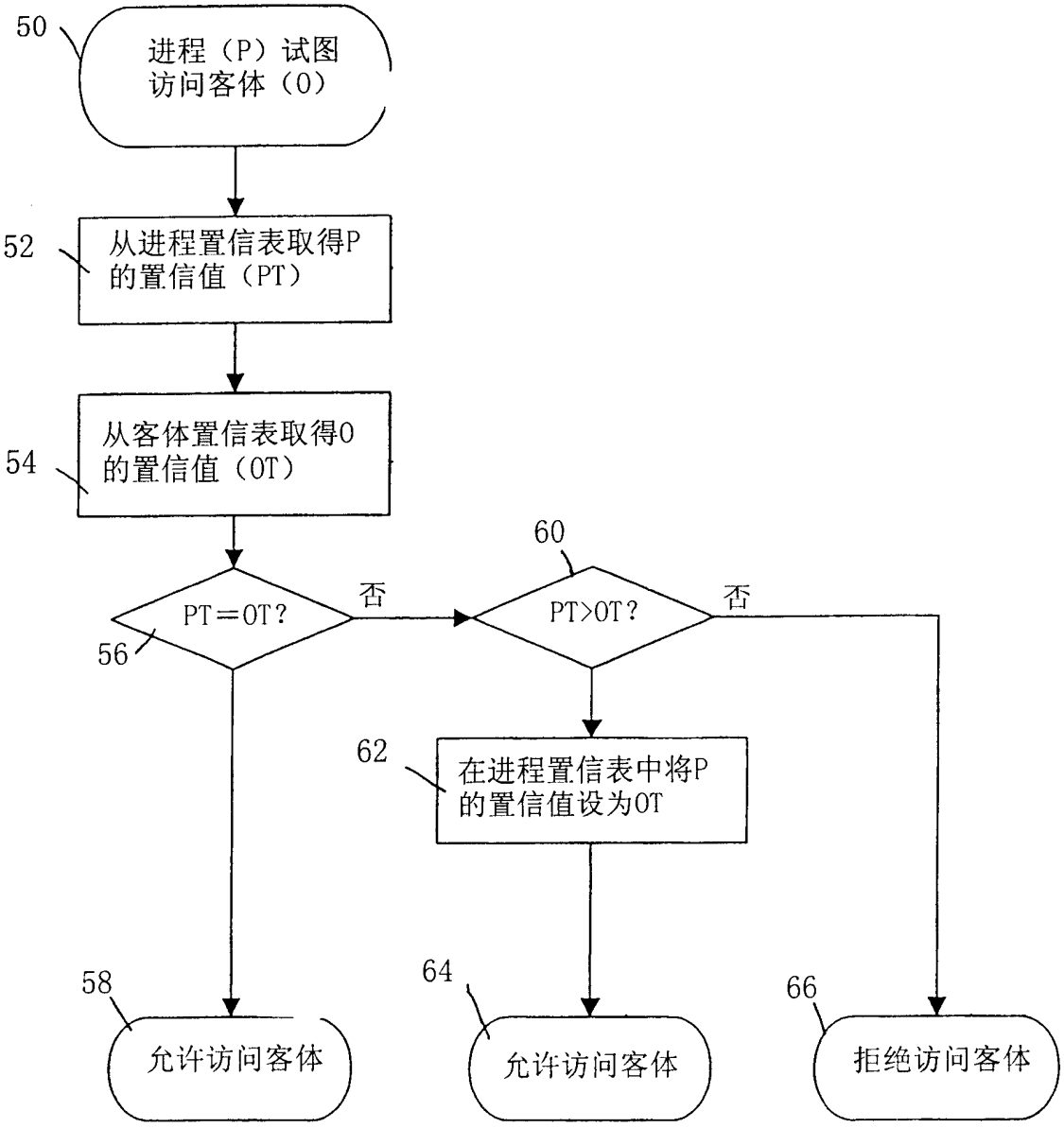

50
进程（P）试图访问客体（O）
52
从进程置信表取得P的置信值（PT）
54
从客体置信表取得O的置信值（OT）
56
PT＝OT?
否
60
PT>OT?
否
62
在进程置信表中将P的置信值设为OT
58
允许访问客体
64
允许访问客体
66
拒绝访问客体

图5

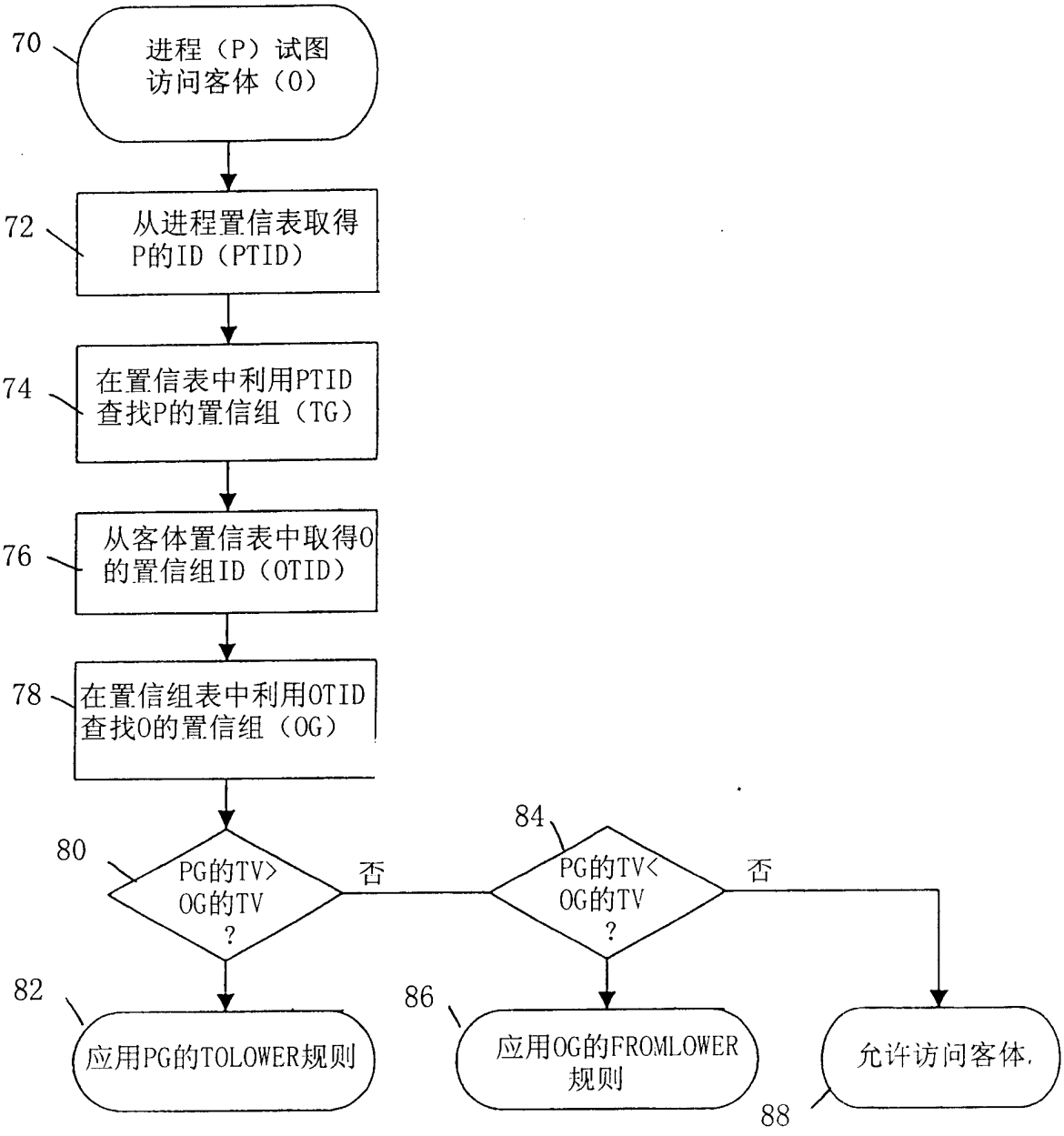

70
进程（P）试图访问客体（O）
72
从进程置信表取得P的ID（PTID）
74
在置信表中利用PTID查找P的置信组（TG）
76
从客体置信表中取得O的置信组ID（OTID）
78
在置信组表中利用OTID查找O的置信组（OG）
80
PG的TV>OG的TV？
否
84
PG的TV<OG的TV？
否
82
应用PG的TOLOWER规则
86
应用OG的FROMLOWER规则
88
允许访问客体.

图6

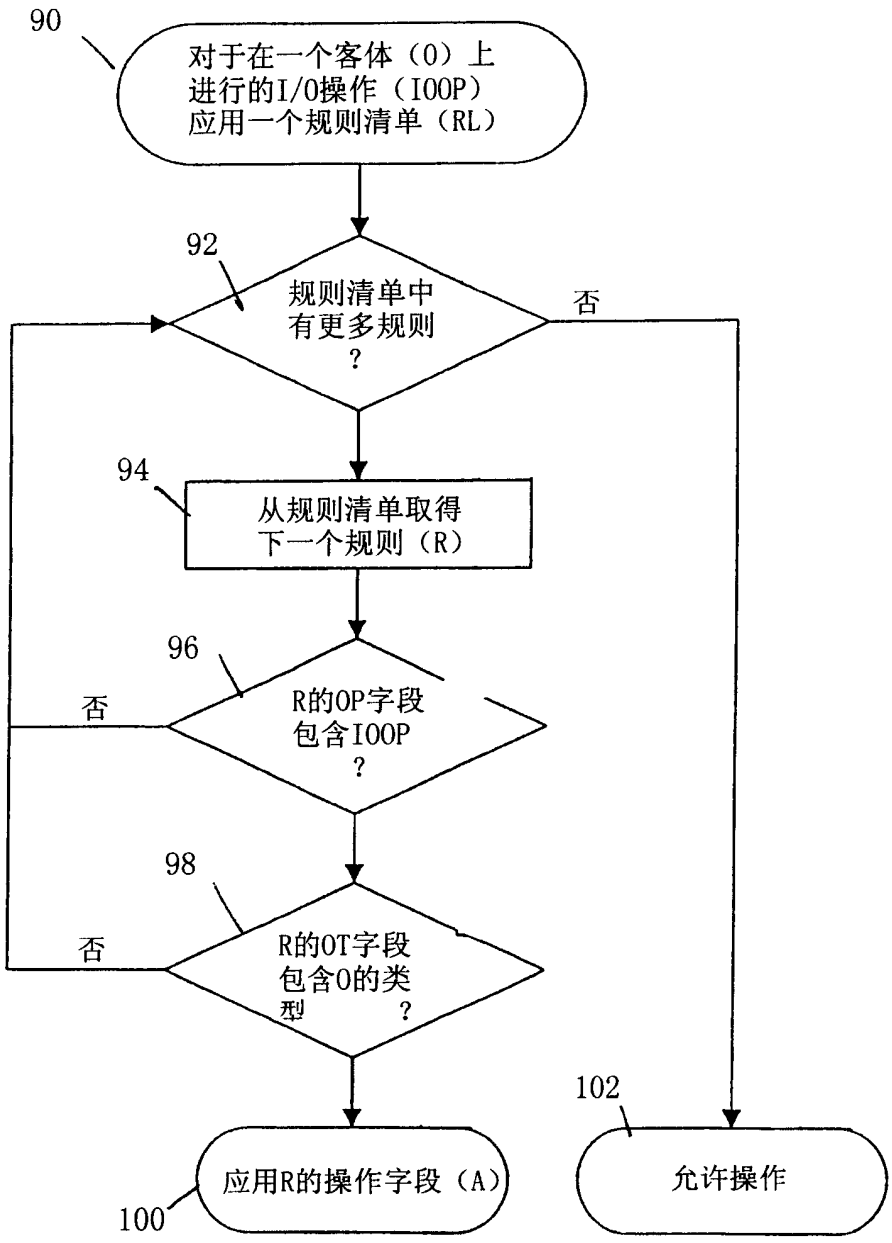

90
对于在一个客体（O）上
进行的I/O操作（IOOP）
应用一个规则清单（RL）
92
规则清单中
有更多规则
？
否
94
从规则清单取得
下一个规则（R）
96
R的OP字段
包含IOOP
？
否
98
R的OT字段
包含O的类
型 ？
否
100
应用R的操作字段（A）
102
允许操作

图7