(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 111324891 A
(43)申请公布日 2020.06.23

(21)申请号 201911254076.6

(22)申请日 2019.12.06

(30)优先权数据
16/218,625 2018.12.13 US

(71)申请人 北京京东尚科信息技术有限公司
地址 100086 北京市海淀区知春路76号8层
申请人 京东美国科技公司

(72)发明人 曾俊源 詹臻新 陈源 苏志刚

(74)专利代理机构 中科专利商标代理有限责任公司 11021
代理人 纪雯

(51)Int.Cl.
*G06F 21/56*(2013.01)
*G06F 9/455*(2006.01)

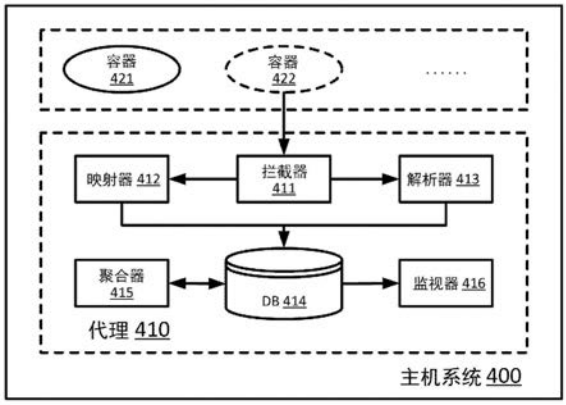
权利要求书3页 说明书15页 附图9页

(54)发明名称

用于容器文件完整性监视的系统和方法

(57)摘要

一种用于在主机计算设备中监视文件完整性的系统,主机计算设备具有处理器和存储计算机可执行代码的存储设备。计算机可执行代码配置为:提供容器、容器外部的代理、以及为容器配置策略的策略文件;拦截指示挂载的系统调用,以及构建容器文件路径和具有挂载对应关系的主机文件路径之间的第一对应关系;拦截该容器的指示打开策略文件的系统调用,以及构建容器文件路径和该容器文件路径的违规之间的第二对应关系;聚合第一对应关系和第二对应关系以获取主机文件路径和违规之间的对应关系;以及通过检测主机文件路径的违规来监视容器的文件完整性。




CN 111324891 A

1.一种用于监视主机计算设备中的多个容器的文件完整性的系统,所述主机计算设备包括处理器和存储计算机可执行代码的存储设备,其中当在所述处理器处执行时,所述计算机可执行代码配置为:

提供所述容器、所述容器外部的代理、以及为所述容器配置策略的策略文件;

由所述代理拦截对所述容器中的一个容器的第一系统调用,所述第一系统调用指示容器文件路径挂载到对应的主机文件路径;

由所述代理构建所述容器文件路径和所述主机文件路径之间的第一对应关系;

由所述代理拦截对所述容器中的所述一个容器的第二系统调用,所述第二系统调用指示打开所述策略文件,并且所述策略文件包含所述容器文件路径和与所述容器文件路径对应的违规;

由所述代理构建所述容器文件路径和所述违规之间的第二对应关系;

由所述代理聚合所述第一对应关系和所述第二对应关系以获取所述主机文件路径和所述容器的违规之间的第三对应关系;以及

由所述代理通过检测所述第三对应关系中主机文件路径的违规来监视所述容器的文件完整性。

2.根据权利要求1所述的系统,其中所述第一对应关系和所述第二对应关系中的每个利用所述容器的标识(cid)来标识,并且当所述第一对应关系和所述第二对应关系具有相同的cid时,将它们聚合。

3.根据权利要求2所述的系统,其中所述第一系统调用的cid是基于第一系统调用的进程标识(pid)来定义的,所述第二系统调用的cid是基于第二系统调用的pid来定义的。

4.根据权利要求3所述的系统,其中所述第一系统调用的pid和所述第二系统调用的pid中的至少一个与初始化所述容器的进程的pid相同。

5.根据权利要求3所述的系统,其中所述第一系统调用的pid和所述第二系统调用的pid中的至少一个是初始化所述容器的进程的pid的派生物。

6.根据权利要求3所述的系统,所述计算机可执行代码进一步配置为:构建进程标识和所述容器的标识之间的映射,使得进程的pid能够基于所述映射转换成进程的cid。

7.根据权利要求1所述的系统,所述计算机可执行代码进一步配置为:使用所述第三对应关系来更新所述策略文件,以及使用更新后的策略文件来监视所述容器的文件完整性。

8.根据权利要求1所述的系统,其中所述计算机可执行代码进一步配置为:在拦截所述第二系统调用之后以及构建所述第二对应关系之前,从所述策略文件中获取用于所述容器的策略。

9.一种用于监视主机计算设备中的多个容器的文件完整性的方法,所述方法包括:

在所述容器外部提供所述主机计算设备中的代理并提供为所述容器配置策略的策略文件;

由所述代理拦截对所述容器中的一个容器的第一系统调用,所述第一系统调用指示容器文件路径挂载到对应的主机文件路径;

由所述代理构建所述容器文件路径和所述主机文件路径之间的第一对应关系;

由所述代理拦截对所述容器中的所述一个容器的第二系统调用,所述第二系统调用指示打开所述策略文件,并且所述策略文件包含所述容器文件路径和与所述容器文件路径对

应的违规；

由所述代理构建所述容器文件路径和所述违规之间的第二对应关系；

由所述代理聚合所述第一对应关系和所述第二对应关系以获取所述主机文件路径和所述容器的违规之间的第三对应关系；以及

由所述代理通过检测所述第三对应关系中主机文件路径的违规来监视所述容器的文件完整性。

10.根据权利要求9所述的方法，其中所述第一对应关系和所述第二对应关系中的每个利用所述容器的标识(cid)来标识，并且当所述第一对应关系和所述第二对应关系具有相同的cid时，将它们聚合。

11.根据权利要求10所述的方法，其中所述第一系统调用的cid是基于第一系统调用的进程标识(pid)来定义的，所述第二系统调用的cid是基于第二系统调用的pid来定义的。

12.根据权利要求11所述的方法，其中所述第一系统调用的pid和所述第二系统调用的pid中的至少一个与初始化所述容器的进程的pid相同。

13.根据权利要求11所述的方法，其中所述第一系统调用的pid和所述第二系统调用的pid中的至少一个是初始化所述容器的进程的pid的派生物。

14.根据权利要求11所述的方法，进一步包括：构建进程标识和所述容器的标识之间的映射，使得进程的pid能够基于所述映射转换成进程的cid。

15.根据权利要求9所述的方法，进一步包括：使用所述第三对应关系来更新所述策略文件，以及使用更新后的策略文件来监视所述容器的文件完整性。

16.根据权利要求9所述的方法，进一步包括：在拦截所述第二系统调用之后以及构建所述第二对应关系之前：从所述策略文件中获取用于所述容器的策略。

17.一种存储计算机可执行代码的非瞬态计算机可读介质，其中当在计算设备的处理器处执行时，所述计算机可执行代码配置为：

提供多个容器、所述容器外部的代理、以及为所述容器配置策略的策略文件；

由所述代理拦截对所述容器中的一个容器的第一系统调用，所述第一系统调用指示容器文件路径挂载到对应的主机文件路径；

由所述代理构建所述容器文件路径和所述主机文件路径之间的第一对应关系；

由所述代理拦截对所述容器中的所述一个容器的第二系统调用，所述第二系统调用指示打开所述策略文件，并且所述策略文件包含所述容器文件路径和与所述容器文件路径对应的违规；

由所述代理构建所述容器文件路径和所述违规之间的第二对应关系；

由所述代理聚合所述第一对应关系和所述第二对应关系以获取所述主机文件路径和所述容器的违规之间的第三对应关系；以及

由所述代理通过检测所述第三对应关系中主机文件路径的违规来监视所述容器的文件完整性。

18.根据权利要求17所述的非瞬态计算机可读介质，

其中所述第一对应关系和所述第二对应关系中的每个利用所述容器的标识(cid)来标识，并且当所述第一对应关系和所述第二对应关系具有相同的cid时，将它们聚合；以及

其中所述第一系统调用的cid是基于第一系统调用的进程标识(pid)来定义的，所述第

二系统调用的cid是基于第二系统调用的pid来定义的。

19.根据权利要求18所述的非瞬态计算机可读介质,其中所述第一系统调用的pid和所述第二系统调用的pid中的至少一个与初始化所述容器的进程的pid相同或是其派生物。

20.根据权利要求19所述的非瞬态计算机可读介质,其中所述计算机可执行代码进一步配置为:使用所述第三对应关系来更新所述策略文件,以及使用更新后的策略文件来监视所述容器的文件完整性。

# 用于容器文件完整性监视的系统和方法

[0001] 交叉引用

[0002] 在本公开的描述中引用并讨论了一些参考文献,其可以包括专利、专利申请和各种公开文献。这种参考文献的引用和讨论仅仅提供用于阐明本公开的描述而不是承认任何这种参考文献为此处所描述公开的"现有技术"。在本说明书中引用和讨论的所有参考文献通过引用而将其整体包含于此,并且其公开程度与每个参考文献通过引用单独包含相同。

## 技术领域

[0003] 本公开总体涉及信息安全领域,更具体地,涉及用于使用一个代理对容器进行文件完整性监视(FIM)的系统和方法。

## 背景技术

[0004] 此处提供的背景描述是用于总体呈现本公开上下文的目的。既不明确地承认也不暗示地承认当前记名的发明人的工作(就本背景部分所描述的程度)以及在申请时可能不符合现有技术的描述的某些方面是本公开的现有技术。

[0005] 文件完整性监视(FIM)是一种验证文件系统(包括操作系统(OS)、软件、配置文件和证书文件等等)在当前状态和期望状态之间的完整性的行为,以便检测无效修改、删除、替换等等,这可以有效检测由常规黑客或普通恶意软件(特洛伊木马、Rootkit等等)执行的非法动作。例如,高级持续性威胁(APT),诸如政府资助的网络间谍活动,可以通过精心设计的FIM策略来检测。具体地,对于持续性攻击,APT恶意软件需要修改OS凭证(例如,Linux上的.ssh/id_rsa.pub./etc/shadow,/usr/bin/passwd等),或者通过用恶意二进制替换普通二进制(例如,ps,ls,ifconfig等)或通过删除日志信息/etc/localtime或bash历史.bash_history而在目标机器中隐藏自己,FIM策略可以配置成检测这些修改或删除。

[0006] FIM通过在使用FIM策略的计算设备(例如,主机)的主机系统(也称为基于主机的系统)上运行的软件系统(诸如代理)来实施。FIM策略通常由用户指定,诸如系统管理员,其经由文件结构来定义与文件和/或文件路径有关的违规动作,且检测到违规导致针对文件完整性的警报。例如,通过规则列表来表示FIM策略,规则列表可以格式化为<file,violation>,其中file指的是完整文件路径,其可被主机理解以找到存储在该文件路径中的对应文件,violation指示会破坏文件完整性的动作。例如,</bin/ls,'w,d'>是指如果路径/bin/ls中的文件被修改或删除,则将触发针对文件完整性的警报。例如,EP 2228722 B1、US 7,526,516 B1、KR 20100125116 A、US 7,457,951 B1、US 7,069,594 B1、US 8,819,225 B2(【14】-【19】)等等已经描述了如何在主机上执行FIM。

[0007] 当前,容器作为一种为应用提供虚拟主机环境的轻量虚拟化技术,在诸如主机的物理机器中扮演着越来越重要的角色。不同于传统的虚拟机(【2】-【5】),物理机器上的所有容器共享同一内核以为了更好的资源利用和效率。得益于其效率和优势,容器已经在微服务(【6】)和无服务器计算架构(【7】)中广泛采用。已经表明(【8】)的是包括亚马逊AWS、谷歌云、微软Azure等的云提供商使用容器作为他们的核心技术。例如,代替于直接在计算设备

的主机系统(例如,Linux)上运行,所有应用可以在大量(诸如成千上万)个容器中运行,这些容器继而在主机系统中运行,使得在一个容器中运行的应用可以实现与其他容器中的其他应用隔离的、相对安全的运行环境。

[0008] 针对容器安全性,虚拟化主机通过使用Linux命名空间抽象(【1】)技术将容器相互隔离。特别地,抽象包括进程间通信(IPC)、网络、挂载、进程标识符(PID)、用户和UNIX时间共享系统(UTS)。每个容器具有其自己的用户空间,并且可以仅"查看"它自己的文件系统。容器与主机隔离。

[0009] 关于用于容器的FIM,尽管同一文件路径可能指的是存储在主机上的不同真实文件,挂载为每个容器提供了文件系统的虚拟视图。如图1所示,容器1将主机上的/mount/bin挂载至容器1中的/bin,因此容器1中的绝对路径/bin指向主机上的路径/mount/bin。类似地,容器2中的绝对路径/usr/bin指向主机上的路径/var/lib/docker/overlay2/fa98b6e93/merged/usr/bin。

[0010] 由于基于容器的系统的文件结构不同于传统的基于主机的系统,因此在如前所述的基于主机的系统中使用的现有FIM策略和方法不再适用于基于容器的系统。

[0011] 因此,本领域中存在尚未解决的需求,需要解决前面提到的缺陷和不足。

## 发明内容

[0012] 在某些方面,本公开涉及一种用于监视主机计算设备中的多个容器的文件完整性(FIM)的系统。在一些实施例中,主机计算设备具有处理器和存储计算机可执行代码的存储设备。当在处理器处执行时,计算机可执行代码被配置为:

[0013] 提供容器、容器外部的代理、和为容器配置策略的策略文件;

[0014] 由代理拦截对所述容器中的一个容器的第一系统调用,第一系统调用指示容器文件路径挂载到对应的主机文件路径;

[0015] 由代理构建容器文件路径和主机文件路径之间的第一对应关系;

[0016] 由代理拦截对所述容器中的所述一个容器的第二系统调用,第二系统调用指示打开策略文件,并且策略文件包含容器文件路径和与该容器文件路径对应的违规;

[0017] 由代理构建容器文件路径和违规之间的第二对应关系;

[0018] 由代理聚合所述第一对应关系和所述第二对应关系以获取主机文件路径和容器的违规之间的第三对应关系;以及

[0019] 由代理通过检测第三对应关系中主机文件路径的违规来监视容器的文件完整性。

[0020] 在一些实施例中,第一对应关系和第二对应关系中的每个利用容器标识(cid)来标识,并且当第一对应关系和第二对应关系具有相同的cid时,将它们聚合。

[0021] 在一些实施例中,第一系统调用的cid是基于第一系统调用的进程标识(pid)来定义的,第二系统调用的cid是基于第二系统调用的pid来定义的。

[0022] 在一些实施例中,第一系统调用的pid和第二系统调用的pid中的至少一个与初始化容器的进程的pid相同。

[0023] 在一些实施例中,第一系统调用的pid和第二系统调用的pid中的至少一个是初始化容器的进程的pid的派生物。

[0024] 在一些实施例中,计算机可执行代码进一步配置为:构建进程标识和容器标识之

间的映射,使得进程的pid可以基于该映射转换成进程的cid。

[0025] 在一些实施例中,计算机可执行代码进一步配置为:使用第三对应关系来更新策略文件,以及使用更新后的策略文件来监视容器的文件完整性。

[0026] 在一些实施例中,计算机可执行代码进一步配置为:在拦截第二系统调用之后且在构建第二对应关系之前,从策略文件中获取用于该容器的策略。

[0027] 在一些方面,本公开涉及一种用于监视主机计算设备中的容器的文件完整性的方法。在一些实施例中,该方法包括:

[0028] 在容器外部的主机计算设备中提供代理和为容器配置策略的策略文件;

[0029] 由代理拦截对所述容器中的一个容器的第一系统调用,第一系统调用指示容器文件路径挂载到对应的主机文件路径;

[0030] 由代理构建容器文件路径和主机文件路径之间的第一对应关系;

[0031] 由代理拦截对所述容器中的所述一个容器的第二系统调用,第二系统调用指示打开策略文件,并且策略文件包含容器文件路径和与该容器文件路径对应的违规;

[0032] 由代理构建容器文件路径和违规之间的第二对应关系;

[0033] 由代理聚合第一对应关系和第二对应关系以获取主机文件路径和容器的违规之间的第三对应关系;以及

[0034] 由代理通过检测第三对应关系中主机文件路径的违规来监视容器的文件完整性。

[0035] 在一些实施例中,第一对应关系和第二对应关系中的每个利用容器标识(cid)来标识,并且当第一对应关系和第二对应关系具有相同的cid时,将它们聚合。

[0036] 在一些实施例中,第一系统调用的cid是基于第一系统调用的进程标识(pid)来定义的,第二系统调用的cid是基于第二系统调用的pid来定义的。

[0037] 在一些实施例中,第一系统调用的pid和第二系统调用的pid中的至少一个与初始化容器的进程的pid相同。

[0038] 在一些实施例中,第一系统调用的pid和第二系统调用的pid中的至少一个是初始化容器的进程的pid的派生物。

[0039] 在一些实施例中,该方法进一步包括:构建进程标识和容器标识之间的映射,使得进程的pid可以基于该映射转换成进程的cid。

[0040] 在一些实施例中,该方法进一步包括:使用第三对应关系来更新策略文件,以及使用更新后的策略文件来监视容器的文件完整性。

[0041] 在一些实施例中,该方法进一步包括:在拦截第二系统调用之后且在构建第二对应关系之前,从策略文件中获取用于该容器的策略。

[0042] 在一些方面,本公开涉及一种存储计算机可执行代码的非瞬态计算机可读介质。在一些实施例中,当在计算设备的处理器处执行时,计算机可执行代码被配置为执行上面描述的方法。

[0043] 根据结合以下附图及其说明的对优选实施例的以下描述,本公开的上述及其他方面将变得明显,虽然在不偏离本公开的新颖构思的精神和范围的前提下可能影响其中的变化和修改。

## 附图说明

[0044] 附图图示了本公开的一个或多个实施例,其与书面描述一起用于解释本公开的原理。在可能的情况下,贯穿附图使用相同的参考编号来指示相同或类似的实施例的元素。

[0045] 图1示意性示出现有技术中将容器中的文件路径挂载到主机上的文件路径的示例。

[0046] 图2示意性示出根据本公开一些实施例的针对在主机上运行的多个容器的文件完整性监视(FIM)架构,其中每个容器设置有代理和专用于该容器的FIM策略。

[0047] 图3示意性示出根据本公开某个实施例的针对在主机上运行的多个容器的FIM架构,其中在容器外部设置代理以用于执行容器的集中式FIM。

[0048] 图4示意性示出根据本公开一些实施例的具有集中式FIM代理的主机系统的结构。

[0049] 图5示意性示出在图4的主机系统上运行的集中式FIM代理的处理流。

[0050] 图6A示意性示出根据本公开一些实施例的执行拦截的过程。

[0051] 图6B示意性示出根据本公开一些实施例的执行映射的过程。

[0052] 图6C示意性示出根据本公开一些实施例的执行解析的过程。

[0053] 图6D示意性示出根据本公开一些实施例的执行聚合的过程。

[0054] 图7示意性示出根据本公开某个实施例的计算设备的结构。

[0055] 图8示意性示出根据本公开某个实施例的FIM方法。

[0056] 图9示意性示出根据本公开某个实施例的FIM工作流。

## 具体实施方式

[0057] 在下面的示例中更具体地描述了本公开,下面的示例旨在于仅作为示例说明目的,因为其中很多修改和变体对于本领域技术人员而言将是显然的。现在详细描述本公开的各种实施例。参考附图,类似的附图标记贯穿视图指示类似的组件。如此处描述以及贯穿随附权利要求所使用的,“一”、“一个”和“该”的含义包括复数形式,除非上下文另有明确指示。而且,如此处描述以及贯穿随附权利要求所使用的,“在...中”的含义包括“在...中”和“在...上”,除非上下文另有明确指示。而且,为了读者方便,可以在说明书中使用标题或子标题,但其应当对本公开的范围没有影响。此外,本说明书使用的一些术语在下面更具体地定义。

[0058] 本说明书中使用的术语在本公开的上下文中以及在每个术语所使用的特定上下文中通常具有其领域内的常规含义。在下面或说明书中讨论用于描述本公开的某些术语,以提供有关本公开描述的实践者的附加指导。将会理解,同样的事情可以以不止一种方式来描述。因此,本文中讨论的任何一个或多个术语都可以使用替代语言和同义词,而无论本文是否对某个术语进行了阐述或讨论,都没有任何特殊意义。提供了某些术语的同义词。一个或多个同义词的记载并不排除使用其他同义词。在本说明书中任何地方使用的示例,包括在此讨论的任何术语的示例,都只是为了示例说明,并且绝不限制本公开或任何示例术语的范围和含义。同样,本公开也不限于在本说明书中给出的各种实施例。

[0059] 除非另有定义,此处使用的所有术语(包括技术术语和科学术语)具有与本公开所属领域的普通技术人员所通常理解的相同的含义。将进一步理解,诸如那些常用字典中定义的术语应该解释为具有与在相关领域和本公开的上下文中的含义一致的含义,并且不会

在一种理想化或过于正式的意思上进行解释，除非在此明确如此定义。

[0060] 除非另有定义，在同一对象之前使用的“第一”、“第二”、“第三”等等旨在于区分这些不同的对象，而不是限制其任何顺序。

[0061] 如本文所使用的，“大概”、“大约”、“基本上”或“近似”应一般指在给定值或范围的20％以内，优选在10％以内，更优选在5％以内。这里给出的数值是近似的，意味着如果没有明确说明，则可以推断出术语“大概”、“大约”、“基本上”或“近似”。

[0062] 如本文所使用的，“多个”意味着两个或更多。

[0063] 如本文所使用的，术语“包含”、“包括”、“携带”、“具有”、“含有”、“涉及”等等应理解为开放式，也即，意味着包括但不限于。

[0064] 如本文所使用的，短语A、B和C中的至少一个应当解释为是指使用非排他性逻辑OR的逻辑(A或B或C)。应当理解，方法内的一个或多个步骤可以按不同顺序(或并发地)执行而不改变本公开的原理。如本文所使用的，术语“和/或”包括关联列表事项的一个或多个的任意和所有组合。

[0065] 如本文所使用的，术语“模块”可以指代以下项、作为以下项的一部分、或包括以下项：专用集成电路(ASIC)；电子电路；组合逻辑电路；现场可编程门阵列(FPGA)；执行代码的处理器(共享的、专用的或群组)；提供所描述功能性的其他合适的硬件组件；或上述部分或全部的组合，诸如在片上系统中。术语模块可以包括存储由处理器执行的代码的存储器(共享的、专用的或群组)。

[0066] 如本文所使用的，术语“代码”可以包括软件、固件、和/或微代码，并且可以指程序、例程、函数、类、和/或对象。如本文所使用的，术语共享是指来自多个模块的部分或全部代码可以使用单个(共享)处理器来执行。此外，来自多个模块的部分或全部代码可以通过单个(共享)存储器进行存储。如本文所使用的，术语群组是指来自单个模块的部分或所有代码可以使用处理器群组来执行。此外，来自单个模块的部分或所有代码可以使用存储器群组进行存储。

[0067] 如本文所使用的，术语“接口”一般是指组件之间交互点处的通信工具或装置以用于执行组件之间的数据通信。通常，接口可以应用于硬件和软件级别，并且可以是单向或双向接口。物理硬件接口的示例可以包括电连接器、总线、端口、线缆、端子以及其他I/O设备或组件。与接口通信的组件例如可以是计算机系统的多个组件或外设。

[0068] 本公开涉及计算机系统。如附图中所示出的，计算机组件可以包括物理硬件组件，其以实线框示出，以及虚拟软件组件，其以虚线框示出。本领域普通技术人员将会理解，除非另有说明，这些计算机组件可以在但不限于软件、固件或硬件组件或其组合的形式中实施。

[0069] 本文描述的装置、系统和方法可以通过由一个或多个处理器执行的一个或多个计算机程序来实施。计算机程序包括存储在非瞬态有形计算机可读介质上的处理器可执行指令。计算机程序也可以包括存储的数据。非瞬态有形计算机可读介质的非限制示例是非易失性存储器、磁性存储器和光学存储器。

[0070] 现在将参考附图对本公开进行更全面的描述，在附图中示出了本公开的实施例。然而，本公开可以以许多不同的形式具体化，不应被解释为仅局限于本文所阐述的实施例；相反，提供这些实施例使得本公开将彻底和完整，并将本公开的范围充分传达给本领域技

术人员。

[0071] 在一些方面,本公开提供如图2所示的稻草人架构(strawman architecture)以执行用于容器的FIM,其中每个容器内部具有相应的专用代理以用于执行对应的FIM策略。类似于用于基于主机系统的FIM策略的文件结构,专用于容器的FIM策略可以由用户经由文件结构来指定,以定义与文件和/或文件的文件路径有关的违规动作。特定于容器的FIM策略可以通过规则列表来表示,其也可以格式化为<file,violation>,其中file指示容器中的文件路径,violation指示会破坏文件完整性的动作。

[0072] 然而,容器中用于执行FIM的代理(也称为FIM代理)仅知道容器中的文件路径,由于容器的隔离而不知道容器外的对应主机文件路径,并且不能识别在容器的挂载期间动态确定的对应主机文件路径。

[0073] 上述基于稻草人的技术至少具有如下缺陷:

[0074] (1)一旦容器损坏就不安全;

[0075] (2)需要附加的计算资源,因为每个容器需要一个代理副本;以及

[0076] (3)需要额外的管理工作,并且不能随着容器数量的增加而扩展。

[0077] 作为执行用于容器的FIM的改进,在某些方面,本发明提供了一种新的集中式FIM架构,用于管理在主机上运行的多个容器。如图3所示,多个容器1-N和一个代理均在计算设备的主机系统上运行,且该代理部署在多个容器的外部,用于监视该多个容器中每个容器内的文件完整性。也即,本公开提供了一种用于基于容器的系统的基于主机的FIM方案。

[0078] 在如图2所示的基于容器的系统中,用户经由文件结构来指定用于容器的FIM策略,从而定义此容器中的文件和/或文件路径的违规动作,并且特定于该容器的FIM策略可以通过规则列表来表示,其也可以格式化为<file,violation>,其中violation指示会破坏文件完整性的动作,file指示容器中的文件路径(也称为容器文件路径),其可以被容器中的代理理解以找到对应文件,如前面所描述的。然而,根据本公开一些方面部署在主机系统上的代理不能识别用户为基于容器的系统指定的FIM策略中的容器文件路径,而是仅能理解主机文件路径,类似于前面描述的部署在基于主机的系统上的代理。即,在容器和主机之间,特别是在用户指定的FIM策略中的容器文件路径和部署在主机系统上的代理可以理解的主机文件路径之间,存在语义代沟。

[0079] 为了实现如图3所示的这种集中式FIM架构,本公开提供了一种用于桥接容器和主机之间的语义代沟的方法。图4示意性示出在计算设备上运行的主机系统,其使用一个代理来管理多个容器的FIM,图5示出在图4的主机系统上运行的代理中的工作流。如图4所示,根据本公开一些实施例的主机系统400包括多个容器421、422、...和部署在该多个容器外部的代理410。代理410包括拦截器411、映射器412、解析器413、数据库(DB)414、聚合器415和监视器416。如图5的整体工作流所示出的,用于FIM系统的输入是主机操作系统的系统调用(或syscall)。拦截器将捕获两种类型的syscall:用于容器文件系统挂载的syscall和用于打开策略文件的syscall。这些syscall接着被发送到映射器和解析器以生成:主机文件路径和容器文件路径的映射以及容器文件路径和违规的映射。前者表示在<cid,HostFilePath,ContainerFilePaht>中,而后者表示在<cid,ContainerFilePath,Violation>中。具体地,(1)cid是用于每个容器的唯一ID;(2)HostFilePath是主机中针对每个文件的完整路径(例如,/var/123456/usr/bin/ls);(3)ContainerFilePath是容器中

针对每个文件的完整路径(例如,/bin/ls);(4)Violation指示破坏文件完整性的动作。最后,聚合器使用键cid和ContainerFilePath将两个映射联合起来,继而生成输出<cid,HostFilePath,Violation>,其可以由主机FIM直接使用。

[0080] 这里,用实线示出的容器421旨在于示例性指代这种已经创建并在代理410的FIM下运行的容器;用虚线示出的容器422旨在于示例性指代这种正在创建的容器,在创建期间,代理410用于桥接容器和主机之间的语义代沟,以便在其创建后执行用于容器的FIM,类似于容器421。本领域技术人员将会理解,代理410针对容器422的当前处理已经针对容器421按同样方式执行过。

[0081] 此后,代理410针对容器422的操作将结合图4和图5来详细描述。应当理解,容器422被描述为正在创建的容器的示例;尽管仅针对一个正在创建的容器(例如,容器422)来示例性描述随后的处理,本领域技术人员可以理解,这种处理可以应用于正在创建的多个容器的每一个。

[0082] 拦截器411配置为拦截主机系统400的操作系统的系统调用,以及桥接容器与主机之间的语义代沟。当主机系统400的用户或管理员使用指令开始创建容器422时,可以通过拦截器411来检测一系列系统调用,不仅包括与创建容器422的进程相关的系统调用,而且包括与创建容器422的进程不相关的系统调用,诸如一些常规进程的系统调用。当拦截器411检测到创建进程或终止进程的系统调用时,拦截器411确定创建了常规进程或容器。

[0083] 如果系统调用是由创建容器422的进程做出的,例如,如果创建容器422的进程的进程ID(也表示为“pid”,其是针对进程的唯一ID)是用于容器服务的根进程的派生物,则拦截器411确定已创建容器422。

[0084] 接着,如果拦截器411检测到由创建容器422的进程做出的用于挂载容器422的文件系统的系统调用或由同一进程做出的用于打开容器422的FIM策略文件的系统调用,则拦截器411基于创建容器422的进程来获取容器422的容器ID(也表示为“cid”,其是容器的唯一ID)。

[0085] 在一些实施例中,容器422的容器ID可以基于创建容器422的进程的进程ID来生成。备选地,容器422的容器ID可以基于创建容器422的进程的进程ID以及进程ID与容器ID之间的存储映射关系来获取。

[0086] 在容器422的创建期间,容器的文件系统需要挂载到主机系统中的对应路径上,并且生成用于挂载文件系统的系统调用。当拦截器411检测到文件系统挂载的系统调用时,拦截器411配置为获取挂载信息——包括容器ID以及在容器和主机中的对应文件路径,以及向映射器412发送该挂载信息。

[0087] 同时,在容器422的创建期间,需要读入容器的配置或对应于该容器的FIM策略,并且生成用于打开对应于该容器的FIM策略的系统调用。当拦截器411检测到打开对应策略文件的系统调用时,拦截器411配置为获取策略文件相关信息,以及向解析器413发送该策略文件相关信息。策略文件相关信息可以包括容器的id、一个或多个容器文件路径、以及为容器文件路径定义的违规。

[0088] 如图3所示,容器的FIM策略文件位于主机内部和容器外部。对于许多容器,可以有一个或若干FIM策略文件。如果仅有一个策略文件,此一个策略文件包括用于所有容器的策略。如果有若干策略文件,该若干策略文件的每个可以对应于具有类似策略的一类容器。特

定于容器的FIM策略可以通过规则列表来标识,其可以格式化为<file,violation>,其中file是指容器中的文件路径,violation指示会破坏文件的文件完整性的动作。

[0089] 在一些实施例中,拦截器411使用图6A所示的过程来执行拦截操作。如图6A所示,拦截器411钩住进程创建和终止系统调用,其调用方为用于容器服务的根进程的派生物(02和05行),使用进程ID(pid)来识别创建该容器的进程,以及将进程ID存储在con_process中(03行),从而维护负责容器文件系统挂载的一组容器相关的进程(01-06行)。如果系统调用是由容器相关的进程做出的(08行),并且系统调用打开策略文件(表示为"FILE_OPEN")(09行)或其挂载文件系统(表示为"FILESYSTEM_MOUNT")(10行),则拦截器411将针对每个正挂载的容器获取容器ID(cid)(11-15行)。具体地,容器ID可以是基于创建该容器422的进程的进程ID以及进程ID与容器ID之间的存储映射关系来获得的(12行),或者可以是基于创建该容器的进程的进程ID(pid)来生成的(14行)。如果系统调用(syscall)是FILE_OPEN,则容器ID和策略文件接着将被分派到解析器413(16、17行),且如果系统调用(syscall)是FILESYSTEM_MOUNT,则容器ID和挂载信息接着将被分派到映射器412(18、19行)。

[0090] 在一些实施例中,代替于上述过程,如果容器管理系统"docker"可用于代理410,则拦截器411可以配置为在"docker"下使用进程"dockerd"来检测容器文件系统挂载。

[0091] 在从拦截器411接收到获取的容器422的容器ID和其文件系统的挂载信息之后,映射器412基于容器ID和挂载信息生成容器422与主机系统410之间的文件路径映射关系,并将生成的文件路径映射关系保存在DB 414中。

[0092] 在一些实施例中,映射器412可以首先基于容器ID和挂载信息来确定基本挂载点;接着基于所确定的基本挂载点来生成容器422与主机系统410之间的文件路径映射关系。

[0093] 在一些实施例中,映射器412所生成的文件路径映射关系包括获取的容器ID、容器422的文件系统中文件的第一容器文件路径集合,以及第一容器文件路径集合中的文件的主机文件路径集合,其中主机文件路径是在容器422的挂载期间动态确定的。例如,映射器412所生成的文件路径映射关系可以表示为<cid,HostFilePath,ContainerFilePath>形式,其中cid(例如,图5中示出的1234)是容器422的唯一ID,ContainerFilePath是容器422中每个文件的完整路径(例如,图5中示出的/usr/bin/ls),以及HostFilePath是主机系统400中对应文件的完整路径(例如,图5中示出的/var/123456/usr/bin/ls),其是在容器422的挂载期间动态确定的。

[0094] 在一些实施例中,映射器412使用图6B中示出的过程来执行映射操作。如图6B所示,映射器412针对每个挂载操作,生成容器和主机系统400之间的文件路径映射关系,以及将文件路径映射关系存储在DB 414中。具体地,映射器412首先找到基本挂载点并将其保存到DB 414(06行);接着基于该基本挂载点生成容器和主机系统400之间的文件路径映射关系(07-12行)并将其保存到DB 414(12行)。在一些实施例中,以上信息作为数据库的条目保存到DB 414中。

[0095] 在从拦截器414接收到打开策略文件的系统调用之后,解析器413从主机中的策略文件(FIM策略文件)读取用于该容器的对应策略规则,并将该策略规则存储到DB 414中。在一些实施例中,在存储到DB 414之前,解析器413可以将这些策略规则组装成策略文件,其中生成的策略文件称为基于容器的FIM策略文件,这是因为它是特定于该一个容器的,它具有容器文件路径并且仅在容器内部可被理解。

[0096] 容器的FIM策略文件是预先存储在主机系统400上的文件，指示由用户经由文件结构定义的容器的FIM策略，以定义容器中文件上的哪些动作是违规动作。特定于容器的FIM策略可以表示为规则列表，其可以格式化为<file,violation>，其中file是指容器中的文件路径，violation指示会破坏文件的文件完整性的动作。在一些实施例中，FIM策略文件由用户定义用于多个容器中的至少一个，也即，同一FIM策略文件可以用于不同容器。

[0097] 假设主机系统400中的FIM策略文件包括容器的文件系统中的文件的第二容器文件路径集合以及用于第二容器文件路径集合中的文件的违规，则解析器413从打开策略文件系统调用和FIM策略文件中获取第二容器文件路径集合、容器文件路径的违规以及关联的容器ID(cid)，并将信息存储到DB 414，该存储的信息可由聚合器415访问。每个违规可以指示会破坏第二容器文件路径集合中的每个文件的文件完整性的动作。

[0098] 可以理解，第二容器文件路径集合是包括在前面讨论的由映射器412生成的文件路径映射关系中的第一容器文件路径集合的子集，这是因为不是容器的文件系统中的所有文件都具有对应的FIM规则。在一些实施例中，第二集合不是第一容器文件路径集合的子集，但是第一容器文件路径集合和第二容器文件路径集合具有重叠的文件路径。

[0099] 解析器413所获取的信息可以表示成<cid,ContainerFilePath,Violation>形式，其中cid(例如，图5中示出的1234)是容器422的唯一ID，ContainerFilePath是容器422中每个文件的完整路径(例如，图5中示出的/usr/bin/ls)，以及Violation指示会破坏对应文件的文件完整性的动作(例如，图5中示出的“write”(写入))。

[0100] 在一些实施例中，解析器413使用图6C示出的过程来执行解析操作。如图6C所示，解析器413可以将策略规则定义为<containerId＝cid,type＝RULE,ContainerPath,Violation>(02行)，其中ContainerPath可以是到容器内的文件或目录的完整路径，而Violation指示会破坏该文件的文件完整性的动作。解析器413配置为将结果保存到DB 414(03-07行)。在一些实施例中，结果作为数据库的条目保存到DB 414。

[0101] 在接收到来自DB 414(由映射器412存储的)容器ID、主机文件路径和对应容器文件路径，以及来自DB 414(由解析器413存储的)容器ID、容器文件路径和违规之后，聚合器415配置为生成记录，并将该记录保存到DB 414。在一些实施例中，该记录作为数据库的条目保存到DB 414。所保存的记录可由监视器416访问。如图5所示，记录包括容器ID、主机文件路径、和为该主机文件路径定义的违规动作。在一些实施例中，记录保存成配置文件的形式。在一些实施例中，聚合器415以FIM策略文件形式生成该记录，其可以称为基于主机的FIM策略文件，这是因为容器的文件路径是主机文件路径的形式并且可由容器外部读取。

[0102] 具体地，聚合器415聚合容器ID、该容器的文件系统中的文件的第一容器文件路径集合和在文件路径映射关系中包括的第一容器文件路径集合中的文件的主机文件路径集合，以及容器ID、该容器的文件系统中的文件的第二容器文件路径集合、和针对第二容器文件路径集合中的文件的违规，从而生成包括容器ID、主机文件路径和针对主机文件路径的违规的记录。聚合是基于来自映射器412和解析器413的对应容器文件路径。

[0103] 例如，聚合器415可以使用键cid和ContainerFilePath，将映射器412生成的<cid,HostFilePath,ContainerFilePath>形式的第一路径映射关系与解析器413生成的<cid,ContainerFilePath,Violation>形式的FIM策略规则联合起来，以生成可以由主机系统400上的代理410直接使用的、<cid,HostFilePath,Violation>形式的记录。在位于主机内、容

器外的FIM策略文件中，容器文件路径不能被代理410所理解。现在，记录包括对应于容器文件路径的主机文件路径，其可以被代理410所理解。在一些实施例中，聚合器415可以将这些策略规则组装成策略文件，其中生成的策略文件称为基于主机的FIM策略文件，因为它可以被主机中的代理所理解。备选地，记录可以输入到原始FIM策略文件以获取更新的FIM策略文件，从而更新后的FIM策略文件中的策略涉及容器文件路径，而不是主机文件路径。然而，由于容器挂载的动态特性，不使用策略记录来更新原始FIM策略文件可能更好，从而原始FIM策略是鲁棒的。因此，策略记录优选地保存在新的基于主机的FIM策略文件中，或者布置成其他合适的格式，只要监视器416可以使用策略记录来监视容器的文件完整性。在一些实施例中，上述记录作为数据库的条目保存到DB 414。

[0104] 在一些实施例中，聚合器415使用图6D所示的过程来执行聚合操作。如图6D所示，聚合器415可以使用键cid和ContainerFilePath，将映射器412的结果和解析器413的结果联合起来，以生成可供监视器416使用的<Container ID，type，HostFilePath，ContainerFilePath，Violation>形式的记录。

[0105] 如上所述，聚合器415进一步配置为向监视器416发送记录(或多个记录)。接着基于该记录，监视器416配置为实时监视主机文件路径(对应于容器文件路径)，并实现代理410的FIM功能。

[0106] 具体地，监视器416根据第一容器文件路径集合中的文件的主机文件路径和针对第二容器文件路径集合中的文件的违规，针对容器422的文件系统在主机系统400上执行FIM。在图5的示例中，监视器416可以遵循FIM策略记录<1234，/var/123456/usr/bin/ls，'write'>来执行FIM，这意味着如果检测到对容器ID为"1234"的容器的文件的"write"动作并且其在主机上的真实文件路径是/var/123456/usr/bin/ls，则应当发布警报。

[0107] 在一些实施例中，代替于从聚合器415接收记录，监视器416可以利用type＝FIM(类型＝FIM)从DB 414获取新记录并与主机系统400通信，以便通过例如注册来自OS内核的FIM回调(例如Linux中的inotify系统调用)，或者通过周期性检查指定文件的哈希值是否改变，来监视文件完整性。

[0108] 这样，利用创建容器期间的上述处理，直接在主机系统400上运行并布置在容器外部的代理410可以根据从聚合器415接收的或从DB 414获取的记录来为所创建的容器执行FIM，从而将某些主机文件路径或主机文件路径下的文件的操作链接到特定容器。在一些实施例中，代理410也可以使用记录来更新基于主机的FIM策略文件。在一些实施例中，代理410可以使用记录来创建新的FIM策略文件。

[0109] 图7示意性示出根据本公开一些实施例的计算设备的结构。在一些实施例中，计算设备700可以用于实现针对多个容器的集中式FIM。在一些实施例中，计算设备700可以是服务器计算机、集群、云计算机、通用计算机或专用计算机，其可以在多个容器上执行集中式FIM。

[0110] 如图7所示，计算设备700可以包括但不限于处理器702、存储器704和存储设备706。在一些实施例中，计算设备700可以包括其他硬件组件和软件组件(未示出)以执行其对应任务。这些硬件和软件组件的示例可以包括但不限于其他需要的存储器、接口、总线、输入/输出(I/O)模块或设备、网络接口、和外围设备。在一些实施例中，计算设备700是云计算机，并且处理器702、存储器704和存储设备706是通过互联网按需提供的共享资源。

[0111] 处理器702可以是中央处理单元(CPU),其配置为控制计算设备700的操作。处理器702可以执行计算设备700的操作系统(OS)或其他应用。在一些实施例中,计算设备700可以具有不止一个CPU作为处理器,诸如两个CPU、四个CPU、8个CPU或任何合适数目的CPU。

[0112] 存储器704可以是易失性存储器,诸如随机访问存储器(RAM),用于存储计算设备700操作期间的数据和信息。在一些实施例中,存储器704可以是易失性存储器阵列。在一些实施例中,计算设备700可以在不止一个存储器704上运行。

[0113] 存储设备706是非易失性数据存储介质,用于存储计算设备700的OS(未示出)和其他应用。存储设备706的示例可以包括非易失性存储器,诸如闪存、记忆卡、USB驱动、硬驱、软盘、光驱或任何其他类型的数据存储设备。在一些实施例中,计算设备700可以具有多个存储设备706,其可以是相同的存储设备或不同类型的存储设备,并且计算设备700的应用可以存储在计算设备700的一个或多个存储设备706中。如图7所示,存储设备706包括主机系统708。主机系统708提供用于在多个容器上执行集中式FIM的平台。

[0114] 如图7所示,除其他之外,主机系统708包括容器模块710、代理712和FIM策略726。容器模块710配置为初始化主机系统708中的一个或多个容器。在容器之一的初始化期间,容器模块710可以将容器的文件路径挂载到主机文件系统中的文件路径上,以及读取FIM策略726以定义对容器的限制。代理712配置为基于FIM策略726来监视容器的文件完整性。FIM策略726是包含用于容器的FIM策略的文件。FIM策略可以包括容器标识、容器文件路径和容器文件路径的违规。在一些实施例中,每个容器具有一组对应的策略规则。在一些实施例中,容器被划分成多类,每类容器具有一组策略规则。在上述实施例中,仅有一个FIM策略726用于代理712。在其他实施例中,每个容器或每类容器具有对应的FIM策略726。

[0115] 如图7所示,代理712包括拦截器714、映射器716、解析器718、数据库720、聚合器722和监视器724。在一些实施例中,代理712可以包括上述模块714-724(例如,数据库)的操作所需的其他应用或模块,其作为常见组件而被省略以便不模糊本公开的原理。具有类似于图4的主机系统400的结构的这种主机系统708的详细描述可以参考前面讨论过的图4和图5中的描述。应该注意,模块均是由计算机可执行代码或指令、或数据表或数据库来实现的,其共同形成一个应用程序。在一些实施例中,每个模块可以进一步包括子模块。备选地,一些模块可以组合成一个堆栈。在其他实施例中,一些模块可以实现为电路,而不是可执行代码。

[0116] 拦截器714配置为监视主机系统708的系统调用。系统调用包括与容器的操作不相关的主机系统708的常规系统调用以及与容器的操作相关的系统调用。当系统调用是创建或终止进程时(PROCESS_CREATION或PROCESS TERMINATION),系统调用与进程的创建或终止相关。在这种情况下,拦截器714配置为检查进程的进程标识(pid)。当pid是容器守护进程的派生物时,其指示该进程是容器的创建。在一些实施例中,拦截器714配置为标记或记录该pid,以及将此pid存储在con_process中。之后,当监视具有该pid的进程,并且系统调用是FILE_OPEN(打开文件)而打开的文件是FIM策略726时,或者系统调用是MOUNT(挂载)时,拦截器714配置为将系统调用分别分派到映射器716和解析器718。在一些实施例中,拦截器714进一步配置为生成对应于与该容器相关的pid的容器id(cid),构建pid到cid关系,并在pid_cid中存储pid-cid关系,从而拦截器714可以容易地将进程与容器链接起来。

[0117] 映射器716配置为:一旦接收到来自拦截器714的系统调用(MOUNT),为每个挂载操

作生成源(容器)和目标(主机)之间的文件路径映射,并将输出存储在数据库720中。返回参考图5,每个映射包括容器标识、挂载的主机文件路径和对应于该主机文件路径的容器文件路径。

[0118] 解析器718配置为:一旦接收到来自拦截器714的系统调用(打开策略文件),定义容器文件路径的策略规则,并将策略规则存储在数据库720中。返回参考图5,策略规则包括容器标识、容器文件路径、以及容器文件路径的违规。在一些实施例中,解析器718可以从打开策略文件系统调用直接获取容器标识、容器文件路径和违规。在其他实施例中,解析器718配置为读取FIM策略726并基于包含在系统调用中的标识从FIM策略726获取这些信息。

[0119] 聚合器722配置为获取由映射器716存储在数据库720中的文件路径映射以及由解析器718存储在数据库720中的策略规则,将这些结果联合起来生成策略记录,并将策略记录存储在数据库720中,该存储的策略记录可由监视器724访问。在一些实施例中,聚合器722配置为基于结果的容器标识来联合结果,其中若可能,则将具有相同容器标识的结果联合在一起。在一些实施例中,策略记录包括容器标识、主机文件路径和违规。在一些实施例中,策略记录还可以包括容器文件路径。在此实施例中,聚合器722配置为从数据库720获取信息。在其他实施例中,数据库720不是必需的,并且聚合器722直接从映射器716和解析器718接收这些信息。

[0120] 监视器724响应于接收到来自聚合器722的策略记录,基于策略记录来监视容器,并在主机文件路径的操作违规时提供警告。在一些实施例中,监视器724配置为直接使用策略记录来监视主机系统708。在一些实施例中,监视器724配置为使用策略记录来更新FIM策略,以及基于更新后的FIM策略726来监视主机系统708。

[0121] 图8描述了根据本公开一个实施例的FIM方法800。在一些实施例中,方法800由图7所示的计算设备700来执行,并应用于图7所示的主机系统708或图4所示的主机系统400,以针对在主机系统上创建的多个容器执行集中式FIM。此后,图8的方法将结合图7的计算设备700来描述,不过应当理解,图7中计算设备700的主机系统708上的处理也可以应用于图4中的主机系统400。应当特别注意,除非本公开另有声明,方法的步骤可以按不同顺序布置,因此不限于图8中所示的顺序。为了简单起见,在此将省略前面已经讨论过的一些详细描述。

[0122] 如图8所示,在过程802处,拦截器714监视主机系统708的系统调用,拦截容器的挂载系统调用,并向映射器716发送容器的挂载系统调用。

[0123] 在过程804处,一旦接收到挂载系统调用,映射器716生成容器文件路径与主机文件路径之间的文件路径映射,以及将该映射存储到数据库720。

[0124] 在过程806处,拦截器714监视主机系统708的系统调用,拦截打开策略文件(FIM策略726)系统调用,并向解析器718发送打开策略文件系统调用。

[0125] 在过程808处,一旦接收到打开策略系统调用,解析器718定义容器文件路径的策略规则,以及将策略规则存储在数据库720中。

[0126] 在过程810处,聚合器获取数据库720中由映射器716存储的映射和由解析器718存储的策略规则,联合结果以生成策略记录,以及将策略记录存储在数据库720中,所存储的策略记录可由监视器724访问。在一些实施例中,聚合器也可以直接向监视器724发送策略记录。通过联合两种类型的结果,将针对容器的违规直接链接到主机文件路径,而不是链接到容器文件路径。

[0127] 在过程812处，监视器724基于策略记录针对违规动作监视主机文件路径的操作。代理712在容器外部操作。由于容器的隔离效应，代理712的监视器724不能看到容器中的容器文件路径。通过上述方法，基于策略记录，监视器724能够监视对应于容器文件路径的主机文件路径，从而实现监视容器中的文件完整性。

[0128] 图9示意性示出根据本公开一个实施例的FIM工作流。在此实施例中，关于图8描述的策略规则和策略记录不是独立的规则和记录，而是通过更新FIM策略726而完成的。在一些实施例中，代理712也可以创建动态FIM策略文件来存储这些与操作容器相关的策略记录。当容器被创建时，用于该容器的策略记录被添加到动态FIM策略文件中，当容器被终止时，策略记录可以从动态FIM策略文件中移除。

[0129] 如图9所示，在过程902处，拦截器714监视主机系统708的系统调用。

[0130] 由于在创建容器时拦截器714可以检测到一系列系统调用，不仅包括与创建该容器的进程相关的系统调用，而且包括与创建该容器的进程不相关的系统调用，因此拦截器714在过程904处确定是检测到由创建容器的进程做出的用于挂载容器的文件系统的系统调用(也称为第一系统调用)，还是由同一进程做出的用于打开容器的FIM策略文件的系统调用(也称为第二系统调用)。

[0131] 如果拦截器714确定检测到第一系统调用(过程904的左分支，则标为“第一系统调用”)，方法900进行到过程906，其中拦截器714基于创建容器的进程来获取容器的容器ID且从第一系统调用获取容器的文件系统的挂载信息，并发送给映射器716；映射器716基于容器ID和挂载信息生成容器与主机系统708之间的文件路径映射关系。

[0132] 如果拦截器714确定检测到第二系统调用(过程904的右分支，标为“第二系统调用”)，方法900进行到过程908，其中拦截器714基于创建容器的进程来获取容器的容器ID并发送给解析器716，以及通知解析器718容器的FIM策略文件需要从主机系统708获取；解析器718基于容器的容器ID和从主机系统708获取的FIM策略文件生成基于容器的FIM策略文件。

[0133] 在过程906或908处，在一些实施例中，拦截器714可以基于创建容器的进程的进程ID来生成该容器的容器ID；备选地，拦截器714可以基于创建容器的进程的进程ID以及进程ID与容器ID之间的存储映射关系来获取该容器的容器ID。

[0134] 在一些实施例中，文件路径映射关系包括容器ID、容器的文件系统中的文件的第一容器文件路径集合、以及第一容器文件路径集合中的文件的主机文件路径集合；基于容器的FIM策略文件包括容器ID、容器的文件系统中的文件的第二容器文件路径集合、以及针对第二容器文件路径集合中的文件的违规，其中第二集合是第一集合的子集。

[0135] 在过程910处，聚合器722基于映射器716生成的文件路径映射关系和解析器718生成的基于容器的FIM策略规则来生成基于主机的FIM策略文件。

[0136] 在一些实施例中，在过程810处，聚合器720聚合包括在文件路径映射关系中的容器ID、容器的文件系统中的文件的第一容器文件路径集合和第一容器文件路径集合中的文件的主机文件路径集合，以及包括在基于容器的FIM策略文件中的容器ID、容器的文件系统中的文件的第二容器文件路径集合和针对第二容器文件路径集合中的文件的违规，以生成所生成的基于主机的FIM策略文件，其包括容器ID、第一容器文件路径集合中的文件的主机文件路径和针对第二容器文件路径集合中的文件的违规。

[0137] 在过程812处，监视器722根据生成的基于主机的FIM策略文件为容器的文件系统执行FIM。在一些实施例中，监视器722根据第一容器文件路径集合中的文件的主机文件路径和针对第二容器文件路径集合中的文件的违规，针对容器的文件系统在主机系统708上执行FIM。

[0138] 在一些实施例中，方法800应用于主机系统708，以便针对在主机系统708上创建的多个容器执行集中式FIM，其可以具有下列益处：

[0139] 除其他之外，根据本公开一些实施例的用于多个容器的集中式FIM的系统和方法具有下列优势：

[0140] (1)更少的资源消耗——其实现了更好的资源利用，这是因为仅一个代理部署在主机系统上(无需为每个容器复制)；

[0141] (2)便于代理管理和更大的扩展性——当代理或策略应当更新时，仅需要更新集中式代理以支持大量的容器；以及

[0142] (3)更安全的FIM，这是因为如果容器被攻击者损坏，主机中的集中式代理由于与容器命名空间隔离而仍然是安全的。

[0143] 提供前述对本公开示例性实施例的描述仅为了示例说明和描述的目的，而不旨在于穷尽或限制本公开至所公开的精确形式。鉴于上述教导，可以有许多修改和变体。

[0144] 选择和描述实施例以便解释本公开的原理以及它们的实践应用，从而使得本领域其他技术人员能够利用本公开和各种实施例以及与所预期的特定用途相适应的各种修改。备选实施例对于本公开所属领域的技术人员将变得明显，而不偏离其精神和范围。因此，本公开的范围由所附权利要求而不是在此描述的前述说明和示例实施例来定义的。

[0145] 参考文献

[0146] 【1】http://man7.org/linux/man-pages/man7/namespaces.7.html；

[0147] 【2】https://www.linux-kvm.org/page/Main_Page；

[0148] 【3】https://www.xenproject.org/；

[0149] 【4】https://www.virtualbox.org/；

[0150] 【5】https://www.vmware.com/products/esxi-and-esx.html；

[0151] 【6】https://microservices.io/patterns/microservices.html；

[0152] 【7】https://aws.amazon.com/serverless/；

[0153] 【8】

[0154] https://www.datamation.com/cloud-computing/aws-vs.-azure-vs.-google-cloud-comparison.html；

[0155] 【9】https://coreos.com/clair/docs/latest/；

[0156] 【10】https://docs.docker.com/engine/security/seccomp/；

[0157] 【11】

[0158] https://cloudplatform.googleblog.com/2018/05/Open-sourcing-gVisor-a-sandboxed-container-runtime.html；

[0159] 【12】https://docs.docker.com/engine/security/apparmor/；

[0160] 【13】使用动态信息流控制的安全无服务技术(Secure Serverless Computing Using Dynamic Information Flow Control)，Kalev Alpernas等人，https://arxiv.org/

abs/1802.08984;

[0161] 【14】EP 2228722 B1,Kaspersky Lab ZAO,2011;

[0162] 【15】US 7,526,516 B1,Kaspersky Lab ZAO,2009;

[0163] 【16】KR 20100125116A,2010;

[0164] 【17】US 7,457,951 B1,Hewlett-Packard Development Co LP,2008;

[0165] 【18】US 7,069,594 B1,McAfee LLC,2006;

[0166] 【19】US 8,819,225 B2,George Mason Research Foundation Inc.,2014。

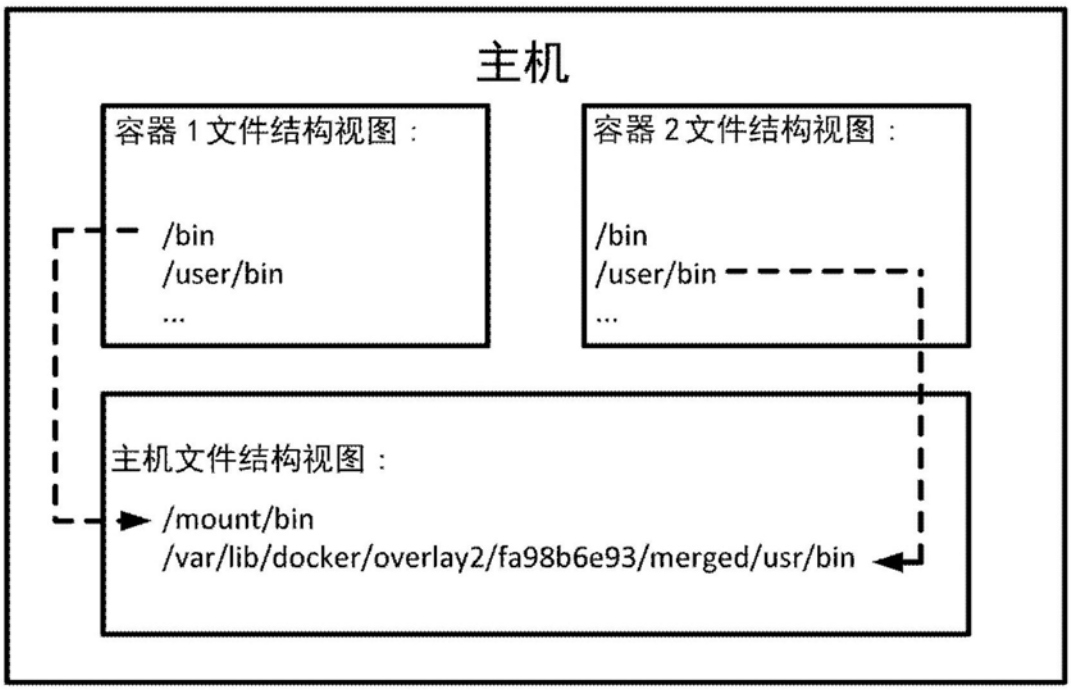

主机
容器 1 文件结构视图：
/bin
/user/bin
...
容器 2 文件结构视图：
/bin
/user/bin
...
主机文件结构视图：
/mount/bin
/var/lib/docker/overlay2/fa98b6e93/merged/usr/bin

图1

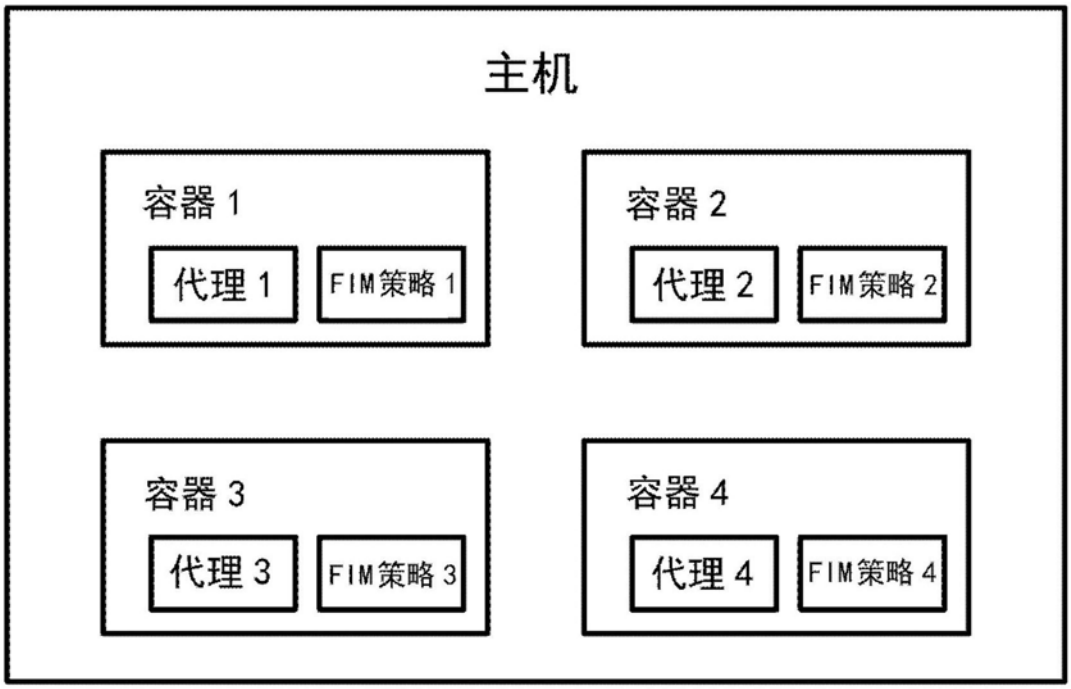

主机
容器 1
代理 1
FIM策略 1
容器 2
代理 2
FIM策略 2
容器 3
代理 3
FIM策略 3
容器 4
代理 4
FIM策略 4

图2

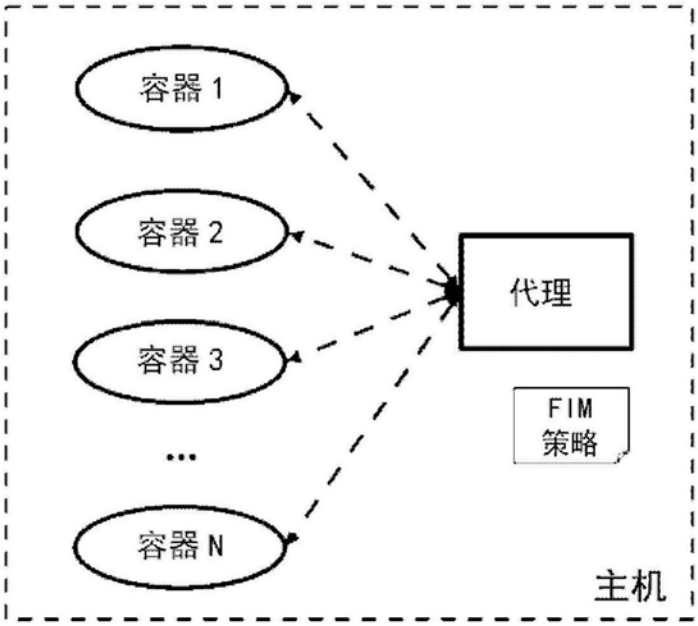

容器 1
容器 2
容器 3
...
容器 N
代理
FIM
策略
主机

图3

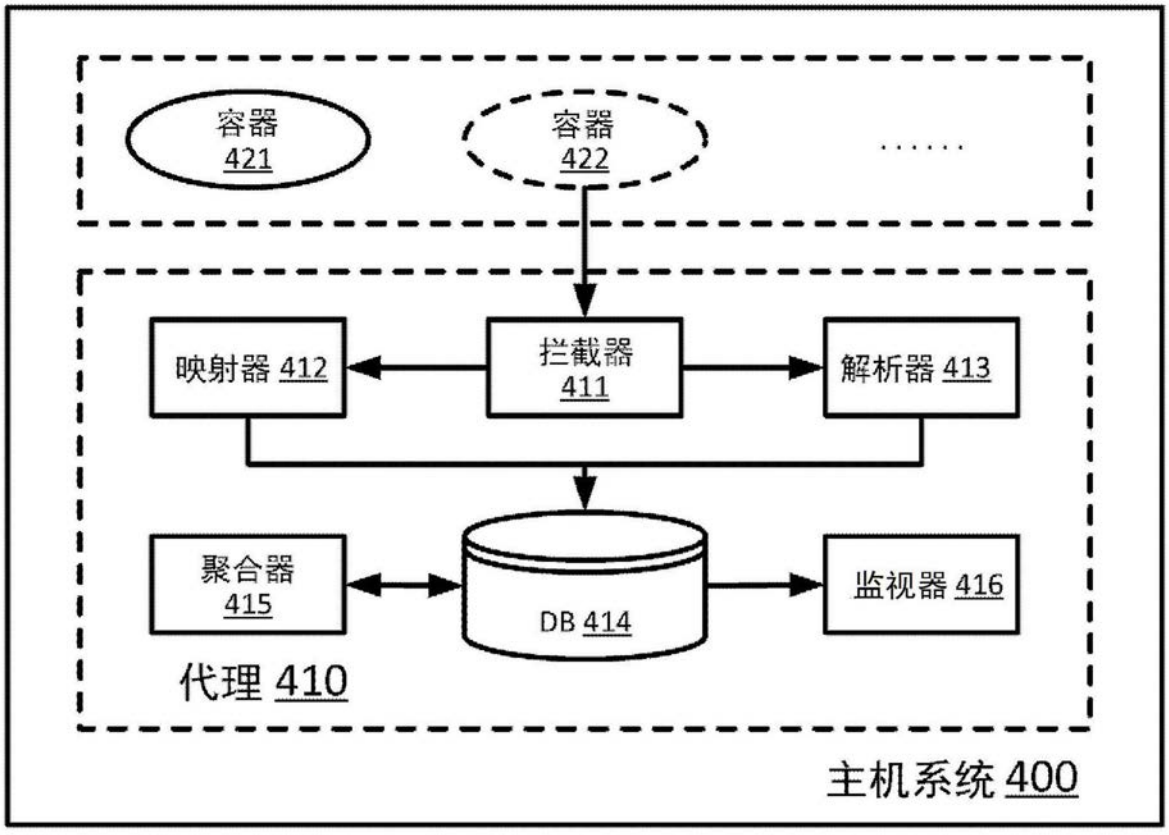

容器
421
容器
422
.....
映射器 412
拦截器
411
解析器 413
聚合器
415
DB 414
监视器 416
代理 410
主机系统 400

图4

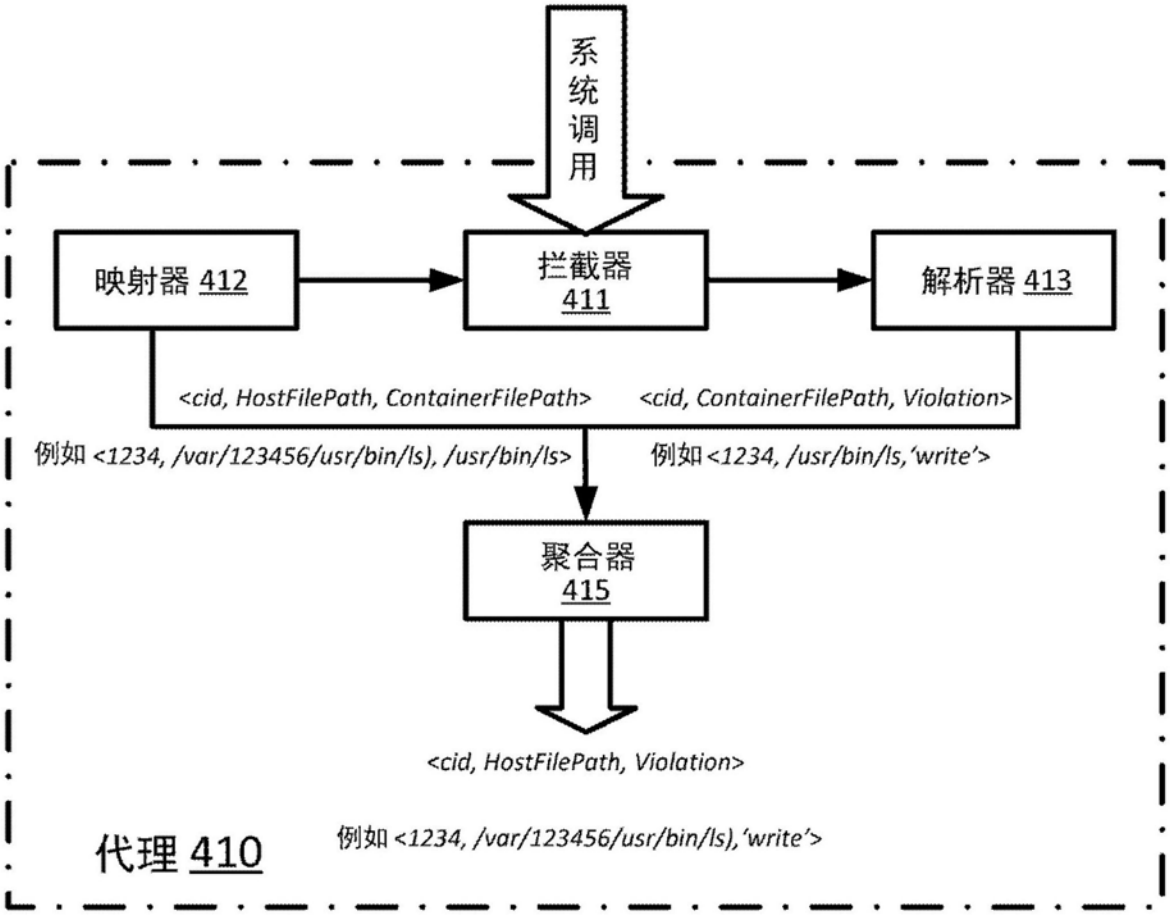

系统调用
映射器 412
拦截器 411
解析器 413
<cid, HostFilePath, ContainerFilePath>
<cid, ContainerFilePath, Violation>
例如 <1234, /var/123456/usr/bin/ls), /usr/bin/ls>
例如 <1234, /usr/bin/ls,'write'>
聚合器 415
<cid, HostFilePath, Violation>
代理 410
例如 <1234, /var/123456/usr/bin/ls),'write'>

图5

```
全局变量：
con_process ← Set().init(con_root.pid): 与容器创建相关的一组进程，其利用
容器的根进程进行初始化（例如，用于 docker 容器的 dockerd）；
pid_cid ← Dict(): 用于将进程 ID映射到容器 ID的词典；

输入：syscall
01 if syscall.type == PROCESS_CREATION:
02     if syscall.caller.pid.parent ∈ con_process AND is_descendant (con_root.pid):
03             con_process.add(syscall.caller.pid)
04 elif syscall.type == PROCESS_TERMINATION:
05     if syscall.caller.pid ∈ con_process:
06             con_process.remove(syscall.caller.pid)
07             pid_cid.remove(syscall.caller.pid)
08 elif syscall.caller.pid ∈ con_process:
09     if (syscall.type == FILE_OPEN AND is_policy_file(syscall.args.file))
10             OR (syscall.type == FILESYSTEM_MOUNT):
11             if syscall.caller.pid ∈ pid_cid.keys:
12                     cid ← pid_cid[syscall.caller.pid]
13             else:
14                     cid ← generate_cid(256)
15                     pid_cid[syscall.caller.pid] ← cid
16             if syscall.type == FILE_OPEN:
17                     Dispatch_Parser(cid, syscall.args.file)
18             elif syscall.type == FILESYSTEM_MOUNT:
19                     Dispatch_Mapper(cid, syscall.args.mount)
```

图6A

```
01 SaveToDB(cid, type, base, h_path, c_path):
02     prefix ← common_prefix(base,h_path)
03     path ← str_replace(h_path, 0, len(prefix), c_path)
04     SaveDB(cid, type, h_path, path)

输入 : cid, mount
05  if mount.type == OVERLAY:
06     record ← < ContainerId=cid, type=BASE, HostPath=mount.dst.fullpath,
ContainerPath= "/" >
07     SaveToDB(record.cid, MAPPED, record.HostPath, record.HostPath,
record.ContainerPath)
08  elif mount.type == BIND:
09     base ← LookupDB(ContainerId=cid, type=BASE)
10     if not_exist(base):
11             throw exception
12     SaveToDB(cid, MAPPED, base.HostPath, mount.dst.fullpath,
base.ContainerPath)
13  else:
14     do_nothing
```

图6B

```
输入 : cid, file
01 foreach rule in open(file):
02   record ← <containerId=cid, type=RULE, ContainerPath, Violation> ←
format(cid, rule)
03   if record.CotainerPath.isDir():
04           foreach file in record.ContainerPath:
05                   SaveDB(<record.cid, record.type, file.path, record.Violation>)
06   else:
07           SaveDB(record)
```

图6C

```
01 for each cid in DB:
02   rule  lookupDB(cid, type=RULE)
03   if rule.ContainerPath.isDir():
04            foreach file in rule.ContainerPath:
05                     Mapper  lookupDB(cid, type=MAPPED, file.path)
06                     SaveDB(<cid, type=FIM, Mapper.HostPath,
Mapper.ContainerPath, rule.Violation>)
07   else:
08            Mapper  lookupDB(cid, type=MAPPED, rule.ContainerPath)
09            SaveDB(<cid, type=FIM, Mapper.HostPath, Mapper.ContainerPath,
rule.Violation>)
```

图6D


计算设备 700
处理器 702
存储器 704
存储设备 706
主机系统 708
容器模块 710
代理 712
拦截器 714
映射器 716
解析器 718
数据库 720
聚合器 722
监视器 724
FIM策略 726

图7

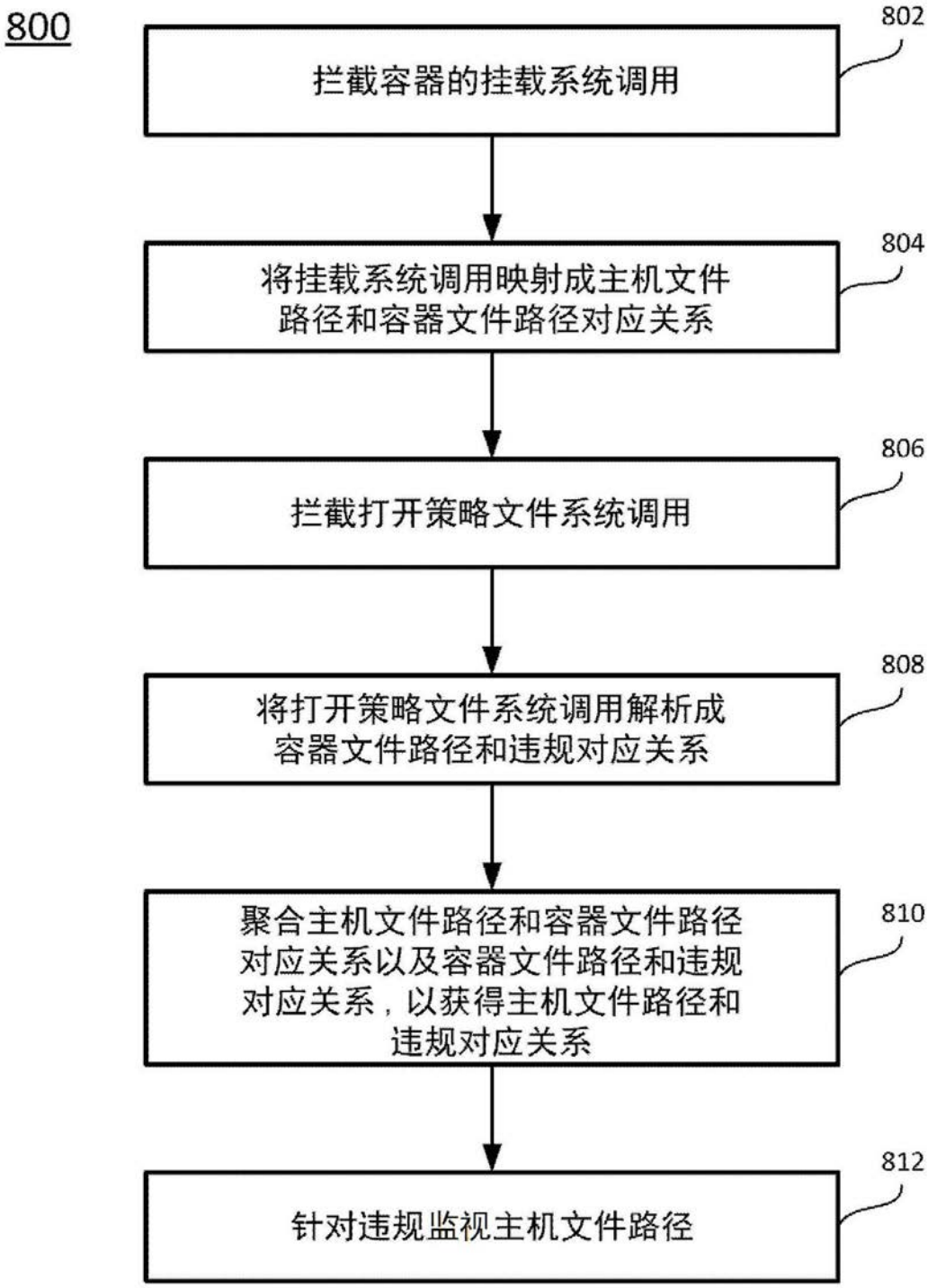

800
802
拦截容器的挂载系统调用
804
将挂载系统调用映射成主机文件路径和容器文件路径对应关系
806
拦截打开策略文件系统调用
808
将打开策略文件系统调用解析成容器文件路径和违规对应关系
810
聚合主机文件路径和容器文件路径对应关系以及容器文件路径和违规对应关系，以获得主机文件路径和违规对应关系
812
针对违规监视主机文件路径

图8

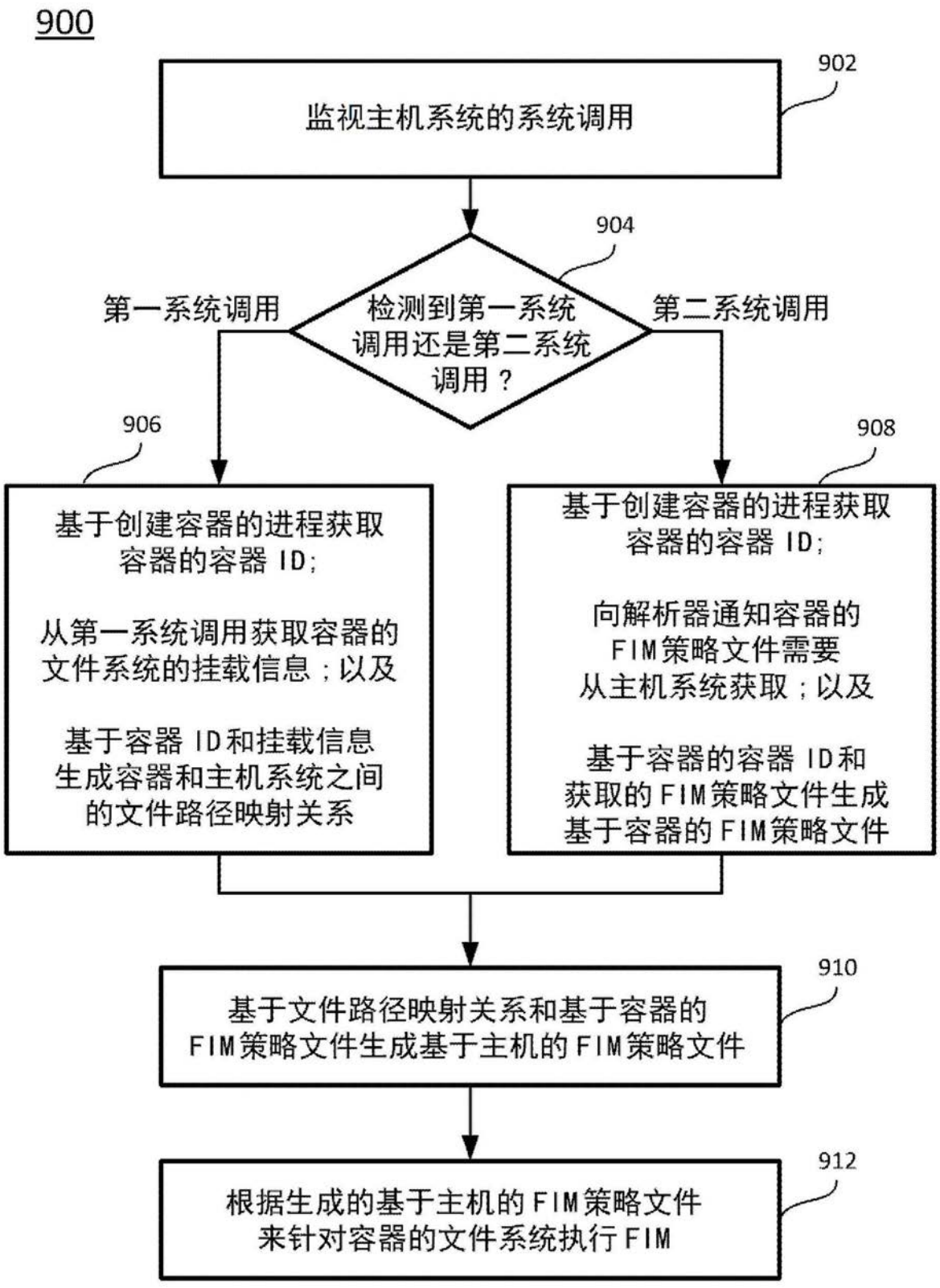

900
902
监视主机系统的系统调用
904
检测到第一系统调用还是第二系统调用？
第一系统调用
第二系统调用
906
基于创建容器的进程获取容器的容器 ID；
从第一系统调用获取容器的文件系统的挂载信息；以及
基于容器 ID和挂载信息生成容器和主机系统之间的文件路径映射关系
908
基于创建容器的进程获取容器的容器 ID；
向解析器通知容器的FIM策略文件需要从主机系统获取；以及
基于容器的容器 ID和获取的FIM策略文件生成基于容器的FIM策略文件
910
基于文件路径映射关系和基于容器的FIM策略文件生成基于主机的FIM策略文件
912
根据生成的基于主机的FIM策略文件来针对容器的文件系统执行FIM

图9