(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101689232 B
(45)授权公告日 2013.05.22

(21)申请号 200880022773.2
(22)申请日 2008.06.25
(30)优先权数据
07388048.6 2007.06.29 EP
60/947,695 2007.07.03 US
(85)PCT申请进入国家阶段日
2009.12.29
(86)PCT申请的申请数据
PCT/EP2008/058091 2008.06.25
(87)PCT申请的公布数据
WO2009/003894 EN 2009.01.08
(73)专利权人 艾利森电话股份有限公司
地址 瑞典斯德哥尔摩
(72)发明人 J·埃克 B·约翰逊 C·冯普拉坦
(74)专利代理机构 中国专利代理(香港)有限公司 72001
代理人 刘春元 李家麟

(51)Int.Cl.
G06F 21/14(2013.01)
G06F 21/16(2013.01)
G06F 21/52(2013.01)
G06F 21/55(2013.01)
(56)对比文件
WO 99/01815 A1,1999.01.14, 全文.
US 2004/0003264 A1,2004.01.01, 全文.
审查员 徐春

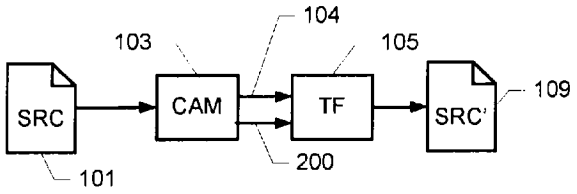
权利要求书3页 说明书13页 附图2页

(54)发明名称

对计算机程序代码的执行轨迹进行模糊化

(57)摘要

一种计算机实现的生成篡改保护的计算机程序代码的方法。所述方法包括获得计算机程序代码的表示,所述计算机程序代码适于使得数据处理系统以第一执行次序执行多个计算任务,每个计算任务在所述计算机程序代码的表示中由至少一个程序语句来表示;获得所述计算任务的多个可替换执行次序;生成程序代码的可执行表示,其适于使得数据处理系统从所述多个可替换执行次序中选择随机化执行次序并且以所选择的随机化执行次序执行计算任务。



CN 101689232 B

1. 一种计算机实现的用于生成篡改保护的计算机程序代码的方法，所述方法包括：

- 获得计算机程序代码的表示，所述计算机程序代码适于使得数据处理系统以第一执行次序执行多个计算任务，每个计算任务在所述计算机程序代码的表示中由至少一个程序语句来表示；

- 获得计算任务的多个可替换执行次序；

- 生成程序代码的可执行表示，其适于使得数据处理系统从所述多个可替换执行次序中选择随机化执行次序并且以所选择的随机化执行次序执行计算任务；

其中生成程序代码的可执行表示进一步包括将计算机可执行指令包含在程序代码的可执行表示中，当程序代码映像由数据处理系统执行时，所述计算机可执行指令适于使得数据处理系统执行以下任务：

a) 执行计算任务中的一个初始任务；

b) 基于对计算任务的执行次序所施加的前趋约束集合的表示从所执行的计算任务的可替换后继集合中选择后续计算任务；

c) 执行所选择的计算任务；

d) 重复步骤 b) 和 c) 直至已经执行了所有计算任务。

2. 如权利要求 1 所述的方法，其中当所述程序代码映像由所述数据处理系统执行时，每个可替换执行次序适于使得数据处理系统产生与第一执行次序相同的程序输出。

3. 如权利要求 1 或 2 所述的方法，进一步包括将可替换执行次序表示为前趋图。

4. 如权利要求 1 至 2 中任一项所述的方法，包括：

- 以预定顺序次序接收指示多个程序语句的输入表示；

- 对程序语句进行分组以获得多个计算任务；

- 识别多个可替换执行次序。

5. 如权利要求 1 至 2 中任一项所述的方法，其中所述前趋约束集合的表示包括前趋图的表示；并且其中生成程序代码的可执行表示进一步包括将计算机可执行指令包含在计算机程序的可执行表示中，所述计算机可执行指令适于使得数据处理系统保持执行状态，所述执行状态由留待执行的计算任务的子集以及留待执行并且其在前趋图中的前导都已经被执行的计算任务的子集中的至少一个来定义。

6. 如权利要求 5 所述的方法，其中执行状态的表示包括用于对生成随机化数据项序列的函数进行初始化的种子数据项。

7. 如权利要求 6 所述的方法，包括为程序代码的不同安装生成程序代码的各个可执行表示，程序代码的每个可执行表示包括指示相应安装的各自种子数据项。

8. 如权利要求 1 至 2 中任一项所述的方法，进一步包括将可执行指令插入用于执行选择后续计算任务的步骤的每个计算任务中。

9. 如权利要求 1 至 2 中任一项所述的方法，进一步包括生成计算任务的多个实例的可执行指令；并且其中选择后续计算任务包括选择计算任务的多个实例之一。

10. 如权利要求 9 所述的方法，进一步包括对所述多个实例中的至少一个执行一个或多个模糊化变换。

11. 如权利要求 1 至 2 中任一项所述的方法，其中生成程序代码的可执行表示包括：生成包含用于使得数据处理系统在所生成的程序代码的可执行表示的每次执行期间从多个

可替换执行次序中选择相同的随机化执行次序的程序代码在内的程序代码的可执行表示。

12. 如权利要求 1 至 2 中任一项所述的方法，包括生成程序代码的多个可执行表示，其中每个可执行表示包括用于使得数据处理系统选择不同的随机化执行次序的程序代码。

13. 如权利要求 1 至 2 中任一项所述的方法，其中程序代码的输入表示包括至少一个输入源代码模块。

14. 如权利要求 1 至 2 中任一项所述的方法，包括生成至少一个经变换的源代码模块。

15. 一种计算机实现的用于生成篡改保护的计算机程序代码的数据处理系统，包括：

- 用于获得计算机程序代码的表示的单元，所述计算机程序代码适于使得数据处理系统以第一执行次序执行多个计算任务，每个计算任务在所述计算机程序代码的表示中由至少一个程序语句来表示；

- 用于获得计算任务的多个可替换执行次序的单元；

- 用于生成程序代码的可执行表示的单元，适于使得数据处理系统从所述多个可替换执行次序中选择随机化执行次序并且以所选择的随机化执行次序执行计算任务；

其中用于生成程序代码的可执行表示的单元进一步包括用于将计算机可执行指令包含在程序代码的可执行表示中的单元，当程序代码映像由数据处理系统执行时，所述计算机可执行指令适于使得数据处理系统执行以下任务：

a) 执行计算任务中的一个初始任务；

b) 基于对计算任务的执行次序所施加的前趋约束集合的表示从所执行的计算任务的可替换后继集合中选择后续计算任务；

c) 执行所选择的计算任务；

d) 重复步骤 b) 和 c) 直至已经执行了所有计算任务。

16. 如权利要求 15 所述的数据处理系统，其中当所述程序代码映像由所述数据处理系统执行时，每个可替换执行次序适于使得数据处理系统产生与第一执行次序相同的程序输出。

17. 如权利要求 15 或 16 所述的数据处理系统，进一步包括用于将可替换执行次序表示为前趋图的单元。

18. 如权利要求 15 至 16 中任一项所述的数据处理系统，包括：

- 用于以预定顺序次序接收指示多个程序语句的输入表示的单元；

- 用于对程序语句进行分组以获得多个计算任务的单元；

- 用于识别多个可替换执行次序的单元。

19. 如权利要求 15 至 16 中任一项所述的数据处理系统，其中所述前趋约束集合的表示包括前趋图的表示；并且其中用于生成程序代码的可执行表示的单元进一步包括用于将计算机可执行指令包含在计算机程序的可执行表示中的单元，所述计算机可执行指令适于使得数据处理系统保持执行状态，所述执行状态由留待执行的计算任务的子集以及留待执行并且其在前趋图中的前导都已经被执行的计算任务的子集中的至少一个来定义。

20. 如权利要求 19 所述的数据处理系统，其中执行状态的表示包括用于对生成随机化数据项序列的函数进行初始化的种子数据项。

21. 如权利要求 20 所述的数据处理系统，包括用于为程序代码的不同安装生成程序代码的各个可执行表示的单元，程序代码的每个可执行表示包括指示相应安装的各自种子数

据项。

22. 如权利要求15至16中任一项所述的数据处理系统，进一步包括用于将可执行指令插入用于执行选择后续计算任务的步骤的每个计算任务中的单元。

23. 如权利要求15至16中任一项所述的数据处理系统，进一步包括用于生成计算任务的多个实例的可执行指令的单元；并且其中选择后续计算任务包括选择计算任务的多个实例之一。

24. 如权利要求23所述的数据处理系统，进一步包括用于对所述多个实例中的至少一个执行一个或多个模糊化变换的单元。

25. 如权利要求15至16中任一项所述的数据处理系统，其中用于生成程序代码的可执行表示的单元包括：用于生成包含用于使得数据处理系统在所生成的程序代码的可执行表示的每次执行期间从多个可替换执行次序中选择相同的随机化执行次序的程序代码在内的程序代码的可执行表示的单元。

26. 如权利要求15至16中任一项所述的数据处理系统，包括用于生成程序代码的多个可执行表示的单元，其中每个可执行表示包括用于使得数据处理系统选择不同的随机化执行次序的程序代码。

27. 如权利要求15至16中任一项所述的数据处理系统，其中程序代码的输入表示包括至少一个输入源代码模块。

28. 如权利要求15至16中任一项所述的数据处理系统，包括用于生成至少一个经变换的源代码模块的单元。

对计算机程序代码的执行轨迹进行模糊化

技术领域

[0001] 本发明涉及计算机程序代码的篡改保护，特别是通过对计算机程序代码的执行轨迹 (execution trace) 进行模糊化 (obfuscate)。

背景技术

[0002] 软件篡改是一种攻击方式，其目的在于改变一件软件的运行方式以便为攻击者带来违法利益。篡改的目标可以是避开 / 禁用复制保护或安全机制以提取秘密或受版权保护的材料，或者引入诸如计算机病毒之类的恶意代码。

[0003] 在许多情况下，违法利益会为软件生产者带来实质性的财务损失。因此，攻击者和软件供应商预期会分别努力破坏和改进针对软件篡改的保护机制。在移动电话的背景下，SIM 锁定或诸如数字权利管理 (DRM) 之类的其它敏感软件组件的保护特别受到关注。此外，其它软件实体的篡改保护和 / 或出于其它目的和 / 或与之相结合使用会是有益的。

[0004] 为了修改软件组件，攻击者通常必须至少对软件组件如何工作有部分了解。通过使得反向工程更加困难，软件篡改会由此被延迟（如果没有被阻止的话）。使得软件更加难以分析的变换对于这一目的是有用的；这样的变换通常被称作模糊化 (obfuscation)。

[0005] 用于对软件进行反向工程的技术可大致分为两组技术：静态（或“离线”）代码分析和动态（或“在线 (live)”）代码分析。当执行动态分析时，软件在其执行时被观察。相反，静态分析通常限于程序代码的某种表示的检验 / 分析而并不实际执行它。动态分析中所采用的一种技术是执行轨迹的比较。

[0006] 典型地，程序的执行轨迹包括在程序执行期间从其读取可执行指令的存储器地址的序列。由此可通过运行程序来收集执行轨迹，例如通过使用特定硬件支持（所谓的轨迹缓冲器）或通过基于软件的地址记录。使用程序的执行轨迹和可执行代码，能够据此重建所执行指令的实际序列。

[0007] 通过提供两个激励集合并且对所产生的执行轨迹中的差异进行比较，攻击者能够获得对软件组件的了解。特别地，执行轨迹的比较可以识别程序的关键决策点。在移动设备的 SIM 锁定和 DRM 解决方案的背景下，关键决策点的示例是用于校正签名或校验和的测试。例如，攻击者可以通过比较两个不同移动设备（例如，运营商锁定设备和未被运营商锁定的设备）所运行的软件的执行轨迹，以便获得有关程序的哪些部分与 SIM 锁定功能相关的信息。

[0008] 之前使通过动态分析进行反向工程更为困难的尝试包括尝试限制攻击者在软件执行时对其进行观察的机会。然而，这样的对抗措施通常已经专用于特定平台和 / 或特定反向工程工具，诸如特定调试器。

[0009] 这种对抗措施的一个示例包括对可执行代码进行加密以及使用将代码的解密和执行相结合的特定硬件。即使适当实施的基于硬件的解密技术能够提供良好的保护，但是这种保护也是以额外的特定硬件为代价来实现的。

[0010] 被称为反调试器 (anti-debugger) 技术的另一种方法的目的在于使得观察特定

调试器中的程序执行的过程复杂化。在一些平台上，执行代码能够向操作系统查询附于该过程的可能调试器，并且例如如果是这种情况就终止。另一种选择是对调试器所使用的技术进行干扰，例如通过篡改断点的设置。然而，反调试器技术专用于特定调制器而且并不提供通用的篡改保护技术。此外，在调试嵌入式系统时通常使用指令集激励器和硬件支持的调试器，由此降低了反调试器技术的实际有用性。此外，使用完全以硬件实施的轨迹缓冲器仍然可以收集执行轨迹。

[0011] 模糊化是一种用来使代码复杂化的技术。模糊化使得代码在其被反编译时更加难以被理解，但是其通常对于代码的功能没有影响。能够使用程序的模糊化通过使得程序更加难以进行反向工程来对它们进行保护。

[0012] 已经提出了多种模糊化技术。由 Tatsuya Toyofuku 等人在 TEnokido 等 (Eds)：EUC Workshops 2005.LNCS 3823，第 916-925 页发表的文章“Program Obfuscation Scheme Using Random Numbers toComplicate Control Flow”提出了一种这样的模糊化机制。然而，该方法被报告为易于受到动态分析的攻击。

[0013] WO01/69355 公开了一种用于通过在程序中插入附加例程以及多个随机建立的附加控制流来向计算机程序中嵌入水印的方法，由此产生具有相应控制流的程序的特定（即带有水印的）版本。

[0014] 美国专利 6,668,325 和 Christian S.Collberg 和 Clark Thomborson 的文章“Watermarking，Tamper-proofing，and Obfuscation-Tools forSoftware protection”，IEEE Transactions on Software engineering，28：6(2002 年 6 月）描述了多种模糊化变换。在这样的变换上引入冗余的 if 语句。所述 if 语句的条件是所谓的不透明谓词 (opaque predicate)，其具有在程序被模糊化但是难以通过代码的静态分析来识别时已知的某种属性。例如永远评估为例如 TRUE（真）的不透明谓词可以用在这样的 if 语句中。结果，在模糊化时知道仅有 if 语句的一个分支将会被执行。因此，在模糊化期间，可以将待执行的代码插入该分支，而决不会执行的其它分支可包括某种任意的“虚设 (dummy)”代码。以上现有技术文档进一步描述了使用其结果可以是 TRUE 或 FALSE（假）的不透明谓词以便选择给定计算任务的两种可替换实例之一。然而，即使这种技术使得代码的静态分析更为困难，但是其并未有效增加试图识别关键决策点的动态分析的难度。

[0015] 美国专利 6,668,325 中所公开的一种特定变换被称作有序变换 (ordering transformation) 并且包括对源代码项目（表达式内的项目、基本块内的语句等）的随机化放置，从而防止代码的反向工程。对源代码项目的重排序基于依赖性分析来执行，所述依赖性分析被执行以确定哪些重排序在技术上是有效的。因此，该现有技术的变换过程生成待执行的源代码的一种技术上可能的重排序。结果所产生的可执行代码将表示该所生成的排序。

[0016] 然而，其仍然存在以下一般问题，提供了一种有效的方法，用于对程序代码进行模糊化以便使得更加难以通过分析程序的执行轨迹来获得有用信息，例如以便识别感兴趣的决策点和 / 或其它关键点。

发明内容

[0017] 通过一种计算机实现的生成篡改保护的计算机程序代码的方法来解决以上和其

它问题，所述方法包括：

[0018] - 获得计算机程序代码的表示，所述计算机程序代码适于使得数据处理系统以第一执行次序执行多个计算任务，每个计算任务在所述计算机程序代码的表示中由至少一个程序语句来表示；

[0019] - 获得所述计算任务的多个可替换执行次序；

[0020] - 生成程序代码的可执行表示，其适于使得数据处理系统从所述多个可替换执行次序中选择随机化执行次序并且以所选择的随机化执行次序执行计算任务。

[0021] 这里所描述的方法实施例提供了软件组件的程序代码（例如源代码）到经修改代码的变换，当所述代码被执行时，这产生经有效地模糊化的执行轨迹。

[0022] 特别地，经变换的代码使得执行结果代码的计算机在多个排序中选择一种排序，即在运行时选择多种可能排序之一，而不是在变换和代码生成过程期间事先确定为一种特定排序。因此，由于排序可以因执行而异，所以在利用不同输入来执行代码时难以获得有用信息（如果不是不可能的话），因此由于在使用不同输入执行软件时执行轨迹中的差异而防止了关键决策点可被轻易检测到。

[0023] 因此，这里所描述的方法实施例以这样的方式对软件组件的输入表示进行变换：增加比较执行轨迹所需的努力。相关差异被隐藏在大量表面上随机的差异之中。因此，使得以获得软件组件的知识为目的的包括比较执行轨迹的篡改攻击的初始阶段明显更为困难。因此，这里所描述的方法针对基于动态分析的攻击提供了对程序代码的有效保护。

[0024] 程序代码的表示可以包括适当的输入表示，其中可以识别不同计算任务。典型地，程序代码的表示包括计算任务的表示，所述计算任务可作为诸如编译器、链接器、解释器等之类的一个或多个代码生成或代码解释工具的输入。因此，程序代码的表示可以由数据处理系统例如利用代码解释器而被直接执行，或者其可以例如利用编译器和/或链接器而被转换成可执行的形式。适当的输入表示的示例包括适当编程语言、对象代码等的源代码。

[0025] 根据这里所描述的方法的变换可以被视为计算任务的临时执行次序的模糊化，而不是现有技术的模糊化技术所使用的源代码构造的空间重排序。因此，这里所描述的方法实施例为若干甚至所有安装提供了相同（或者至少基本上相同，原因在于随机种子 (seed) 会因安装而异）代码映像的变化的临时执行次序。

[0026] 这里所使用的术语“程序语句”意在包括构造编程语言的单元，尤其是这种构造的最小单元。程序语句示例的类型示例包括定义、声明、指定、条件语句、循环和函数/过程调用。条件程序语句通常使得程序有选择地执行一组可替换程序分支之一。每个分支可包括一个或多个程序语句。条件程序语句的示例包括 if 语句、case 语句等。

[0027] 将会意识到的是，执行计算任务的第一执行次序可依赖于输入表示和/或可执行程序代码指令生成期间的一个或多个处理步骤（例如，编译步骤）。例如，输入表示可包括程序语句序列，并且该序列可确定执行的顺序次序 (sequential order)。第一执行次序还可依赖于在程序执行期间接收的程序输入和/或其它任务的输出（例如，在条件程序语句的情况下）。执行次序可以是计算任务的顺序次序。另一方面，在一些实施例中，一些计算任务可以同时执行或者甚至彼此同步执行，例如作为多线程程序执行的两个或更多线程。将会意识到的是，可以在不改变程序的总体行为的情况下改变至少一些计算任务的执行次序。然而，还会意识到的是，在不改变程序行为的情况下可能无法交换一些任务的相对次

序。

[0028] 这里所使用的术语“计算任务”意在包括可由一个或多个计算机指令表示的计算步骤的集合。连带地，软件组件的计算任务构成软件组件所执行的全部计算 / 操作。出于本说明书的目的，每个计算任务可以是细粒度的 (fine-grained)，例如源代码语言的单个程序语句，或者是粗粒度的 (coarse-grained)，例如包括若干语句的函数 / 过程 / 子例程，或者甚至是若干函数 / 过程 / 子例程。

[0029] 获得多个可替换执行次序可以包括通过程序分析工具对程序代码的输入表示进行处理，从而识别多个计算任务以及它们彼此的相依性。例如，该过程可以由自动程序分析工具来执行，或者其可以至少部分基于用户输入来执行。将要意识到的是，存在如函数语言和数据流语言之类的源语言，其尤其适于自动或半自动地识别计算任务和可替换执行次序，原因在于用于重排序的机会特别容易识别。在一些实施例中，分析工具可以提供适当的用户接口，允许用户识别计算任务以及可能的可替换执行次序。将会意识到的是，可替换执行次序的集合可包括第一执行次序。

[0030] 在一些实施例中，当所述程序代码由所述数据处理系统执行时，每个可替换执行次序适于使得数据处理系统产生与第一执行次序相同的程序输出。因此，程序代码的变换是保留语义的 (semantic-preserving)，即对计算机程序所创建的程序输出没有影响。不应当被保留语义的变换直接或间接影响的程序语句也被称作提供程序的关键效果 (critical effect)。与程序环境进行交互的所有程序语句都可以被视为对程序输出有所贡献。

[0031] 这里所使用的术语“程序输出”意在包括程序执行期间的任意可观察行为，例如，能被所使用的编程语言所指定并且能被用户、另一计算机程序、另一设备等所观察 / 注意到的任意程序行为。程序输出的示例包括可经由任意适当的输出设备通过数据接口等而输出的数值输出、文本输出、二进制输出、输出信号、可视 (例如图形) 输出、可听输出等，所述输出设备如计算机屏幕、打印机、存储介质、通信接口等。

[0032] 因此，提供了一种有效的方法，其确保了经模糊化的代码具有与输入代码相同的效果 / 产生与输入代码相同的输出。

[0033] 在一个实施例中，所述方法包括 :

[0034] - 以预定的顺序次序接收指示多个程序语句的输入表示 ;

[0035] - 对所述程序语句进行分组以获得多个计算任务 ;

[0036] - 识别多个可替换执行次序。

[0037] 可替换执行次序可以通过用于表示依赖性关系的任意适当的数据结构来表示，所述依赖性关系指定保证语义等同所需的对执行次序的约束。所述依赖性关系可以通过用于依赖性分析的任意适当方法来确定。

[0038] 对执行次序的约束可以有效地通过指示前趋图 (precedencegraph) 的数据结构来表示，所述前趋图包括指示计算任务的节点以及指示计算任务的前趋 (precedence) 的次序的边。所述前趋图可以是有向图，例如有向非循环图。因此，所述前趋图是可以根据其来选择有效执行次序的前趋约束的高效表示。

[0039] 在一些实施例中，生成程序代码的可执行表示进一步包括将计算机可执行指令包含在程序代码的可执行表示中，当程序代码由数据处理系统所执行时，其适于使得所述数据处理系统执行以下任务 :

[0040] a) 执行计算任务中的一个初始任务；

[0041] b) 从前驱图中的所执行计算任务的可替换后继 (successor) 集合中选择后续计算任务，该前驱图指示对计算任务的执行次序所施加的前趋约束集合；

[0042] c) 执行所选择的计算任务；

[0043] d) 重复步骤 b) 和 c) 直至已经执行了所有计算任务。

[0044] 在一些实施例中，生成程序代码的可执行表示进一步包括将计算机可执行指令包含在程序代码的可执行表示中，其适于使得数据处理系统保持 (maintain) 执行状态，所述执行状态由留待执行的计算任务的子集以及留待执行并且其在前趋图中的前导 (predecessor) 都已经被执行的计算任务的子集中的至少一个来定义。因此，为程序代码提供了有效的机制以使得数据处理系统在给定前趋约束下以随机化次序执行任务。

[0045] 可以根据随机数据项序列来确定随机化执行次序。如本领域中已知的，随机化数据项序列可以由伪随机数生成器来确定，所述伪随机数生成器即使用算术来生成近似随机数属性的数序列的算法，或者可以由本领域中已知的用于生成向观察者表现出随机性的数据项序列的任意其它适当机制来确定。

[0046] 例如，当通过诸如伪随机数生成器之类的用于生成随机化数据项序列的函数来确定随机化数据项序列时，执行状态可进一步包括用于初始化所述用于生成随机化数据项序列的函数的种子数据项。

[0047] 作为选择，随机化次序可由硬件随机数生成器来生成，所述硬件随机数生成器即用于例如基于诸如热噪声、光电效应或其它量子现象之类的适当微观机理而从适当物理过程生成随机数的设备、装置或电路。

[0048] 在一些实施例中，所述方法包括将可执行指令插入每个计算任务中以用于执行选择后续计算任务的步骤。因此，通过把对计算任务进行排序并且更新程序状态的控制逻辑集成到单独任务中，该功能得以被分布而不是作为集中函数而提供，由此使得对手更加难以对负责选择执行次序并且以所选择次序调用任务的程序代码部分进行反向工程。

[0049] 例如，每个任务可以被实现为编程语言的函数 / 过程 / 子例程或者其它适当结构单元，其中每个函数 / 过程 / 子例程在给定的前趋约束下调用后继函数 / 过程 / 子例程。在可替换实施例中，所述方法包括在程序代码由数据处理系统执行时将编程语言的至少一个单独函数 / 过程 / 子例程或者其它适当结构实体插入到适于执行计算任务排序的程序代码中。因此，在这样的可替换实施例中，任务的排序由作为软件组件的实际任务的辅助的单独函数来执行。

[0050] 在一些实施例中，所述方法进一步包括生成计算任务的多个实例的可执行指令；并且选择计算任务包括选择计算任务的多个实例之一。以这种方式，不仅是执行次序，而且从其执行代码的地址也会在软件组件的执行之间发生变化，由此进一步使得执行轨迹的比较复杂化。该方法可以进一步包括对所述多个实例中的至少一个执行一个或多个模糊化变换。从而，通过将不同的保留语义的变换应用于各种实例，实现了额外的多样性 (diversity)。结果，不仅地址有所变化，而且从这些地址取出的指令也有所变化，这进一步使得执行轨迹的比较复杂化。

[0051] 在一些实施例中，例如通过以不同种子数据项发起的或者硬件随机数生成器所生成的各自随机化次序执行计算任务而对给定安装软件组件的每次执行都会产生执行轨迹，

其是唯一的（至少概率很高）。

[0052] 在可替换实施例中，生成程序代码的可执行表示包括：生成包括用于使得数据处理系统在所生成的程序代码的可执行表示的每次执行期间从多个可替换执行次序中选择相同的随机化执行次序的程序代码在内的程序代码的可执行表示。因此，可以使得软件组件的每次安装的执行轨迹对于所述软件组件的每次执行都是相同的，但是随安装而变化。特别地，可以使得所述执行轨迹对于特定安装或安装组是唯一的。

[0053] 例如，所述方法可包括为程序代码的不同安装生成程序代码的各个可执行表示，程序代码的每个可执行表示包括指示相应安装的各自种子数据项。特别地，如果可能的执行次序的数目相对有限，则生成对于软件组件的每次安装唯一的固定轨迹具有防止通过多次执行相同安装来对变化特性进行分析的优势。作为代替，对于某一分析而言将需要大量的安装并且因此需要付出大得多的努力。

[0054] 这里所使用的术语软件组件的安装是指例如以一个或多个可执行文件、安装包等的形式安装在或至少可安装在数据处理系统上的软件的可执行表示的单独副本。例如，安装可被实现为计算机可读介质，诸如其上存储有软件组件的副本的 CD ROM，或者存储在服务器计算机上的可下载文件。

[0055] 在一些实施例中，所述方法包括生成程序代码的多个可执行表示，每个可执行表示包括用于使得数据处理系统选择不同的随机化执行次序的程序代码。因此，可以生成每个所安装的软件组件以便具有唯一的执行轨迹。在这种情况下，相同安装的重复执行产生相同的执行轨迹，这与任意其它安装的轨迹都有所不同。特别地，将会意识到的是，这可被用来使得运营商锁定的移动电话和未锁定移动电话的执行轨迹不相似，而两个安装的重复执行并不提供执行轨迹的变化特性的额外知识。

[0056] 注意到以上和以下所描述的方法的特征可以以软件来实施，并且在数据处理设备或者通过执行诸如计算机可执行指令之类的程序代码装置所引起的其它处理装置上执行。这里和此后，术语处理装置包括适于执行以上功能的任意电路和 / 或设备。特别地，以上术语包括通用或专用的可编程微处理器、数字信号处理器 (DSP)、专用集成电路 (ASIC)、可编程序逻辑阵列 (PLA)、现场可编程门阵列 (FPGA)、专用电路等或者其组合。

[0057] 例如，程序代码装置可以从存储介质或经由计算机网络从另一计算机加载到诸如 RAM（随机存取存储器）之类的存储器中。作为选择，所描述的特征可通过硬连线电路而不是软件来实施，或者通过硬件电路与软件的组合来实施。

[0058] 本发明涉及包括以上和以下所描述的方法、相应设备和计算机程序在内的不同方面，它们均提供结合以上所提到的方法所描述的一个或多个好处和优势，并且均具有与结合以上所提到的方法所描述的实施例相对应的一个或多个实施例。

[0059] 特别地，根据一个方面，一种数据处理系统被适当地配置为执行以上和以下所描述的方法的步骤。

[0060] 根据另一方面，一种计算机程序产品包括计算机可执行指令，当在数据处理系统上执行时，其适于使得所述数据处理系统执行以上和以下所描述的方法。在一些实施例中，所述计算机程序产品被实现为其上存储有计算机可执行指令的计算机可读介质。例如，所述计算机可读介质可以是其上存储有计算机可执行指令的紧致盘 (CD)、光盘、磁盘、磁存储介质、记忆棒等。例如，所述计算机可读介质可以在其上存储用于对程序代码进行篡改保护

的软件应用程序。在其它实施例中，所述计算机程序产品被实现为数据信号，例如适当调制的载波信号。例如，所述计算机可执行指令可以被提供以用于经由计算机网络从服务器计算机进行下载。

[0061] 在一些实施例中，所述计算机程序产品可实现为源代码到源代码的变换器，即接收一个或多个输入源代码模块并生成一个或多个可由传统编译器进行编译的输出源代码模块的计算机程序。在一些实施例中，所述计算机程序产品可集成到编译器中，即其可以被实现为软件编译器，所述软件编译器包括适于使得数据处理系统将以上和以下所描述的方法作为编译器所执行的许多趟编译 (a number of compilationpass) 之一来执行的功能。因此，提供了用于篡改保护和编译的集成软件工具。此外，由于这里所描述的篡改保护的实施例包括一些如传统编译器所使用的用于分析源代码的相同代码分析技术，所以可以重用相应的软件功能，由此提供高效的软件实施方式。

[0062] 出于本说明书的目的，术语存储装置 / 设备和计算机可读介质意在包括任意的适当存储介质、设备或电路，例如只读存储器 (ROM)、随机存取存储器 (RAM)、闪存、可擦除可编程只读存储器 (EPROM)、易失性或非易失性存储器、光存储设备、磁存储设备、磁盘、CD、硬盘等。

附图说明

[0063] 根据以下参考附图所描述的实施例，以上和其它方面将是显而易见的并且通过所述实施例而被阐明，其中：

[0064] 图 1 示出了用于对程序代码进行篡改保护的过程的示意性框图。

[0065] 图 2 示意性图示了前趋图的示例。

[0066] 图 3 示意性图示了产生包括集中任务选择和分派 (dispatch) 函数的经变换源代码的源代码变换的示例。

[0067] 图 4 示意性图示了产生包括分布式任务选择和分派函数的经变换源代码的源代码变换的示例。

[0068] 图 5 示意性图示了产生包括计算任务的多个实例的经变换源代码的源代码变换的另一示例。

[0069] 图 6 示出了用于对程序代码进行篡改保护的系统的示意性框图。

具体实施方式

[0070] 图 1 示出了用于对程序代码进行篡改保护的过程的示意性框图。

[0071] 该过程接收源代码 101。典型地，源代码 101 采用程序员已在其中典型地以诸如 C、C++、Java 等之类的正式编程语言编写了计算机程序的形式。所述源代码能够由编译器自动编译为目标代码或机器代码或者由解释器来执行。源代码 101 可以被表示为一个或多个文本文档或者任意其它适当的数字表示。即使可以使用以任意适当编程语言所编写的源代码作为输入，但是将会意识到的是，一些源代码语言尤其适于识别重排序的机会，例如函数语言和数据流语言。

[0072] 函数语言通常遵循将计算视为数学函数的评估 (evaluation) 并且避免状态和易变数据的编程范例。与强调状态改变的命令式编程风格相比，其强调函数的应用。函数语

言的示例包括 APL、Erlang、Haskell、Lisp、ML 和 Scheme。

[0073] 数据流语言是实施数据流原则和体系结构的编程语言，并且将程序（不是在物理上就是在概念上）模型化为在操作之间流动的数据的有向图。数据流语言共享函数语言的一些特征，并且通常被研发以便为更适于数字处理的语言带来一些函数概念。在数据流语言中，操作典型地被视为具有输入和输出的“黑盒”，其全部都总是明确定义的。数据流语言的示例包括 Lustre、Ptolemy/CAL、LabView 和 Simulink。

[0074] 作为选择，篡改保护过程可接收不同类型的输入表示，其中能够识别计算任务及其相依性，例如，目标代码。

[0075] 源代码 101 被馈入代码分析模块 103 中。所述代码分析模块对源代码进行解析并且识别一组计算任务以及它们彼此的相依性，尤其是在计算任务的执行次序上所施加的前趋约束。代码分析模块 103 将前趋约束表示为前趋图，其示例如图 2 所示。

[0076] 图 2 图示了前趋图的示例。前趋图包括表示各个计算任务的节点以及表示任务之间的前趋约束的边。在图 2 的示例中，一般指定为 200 的前趋图是包括标记为从 A 至 F 的节点的有向非循环图，每个节点表示相应的计算任务。将会意识到的是，所述前趋图可包括与给定软件组件中所识别的计算任务的数量相对应的任意数量的节点。节点 A-F 通过单向边 201 而连接，其中每条边连接两个节点并且标识所述两个节点之间的前趋约束，即计算任务之一必须在其它计算任务可以执行之前被执行。由此每条边从前导节点指向有效的后继节点。例如，在图 2 所示的示例中，任务 E 的执行要求前导任务 B 和 D 已经被执行。因此，任务 A-F 的以下执行次序是满足前趋图 200 所施加的前趋约束的执行轨迹的示例：

[0077] 执行轨迹 #1：

[0078] A，B，C，D，E，F

[0079] 执行轨迹 #2：

[0080] A，D，B，E，C，F

[0081] 执行轨迹 #3：

[0082] A，D，F，B，E，C

[0083] 例如，代码分析模块 103 可以提供允许用户识别计算任务及其有效排序的用户接口。可替换地或除此之外（例如，基于作为开始点的用户定义的任务），代码分析模块 103 可以自动或者至少半自动地识别有效的任务排序（即，前趋图），例如通过生成对任务的初始划分并且通过生成相应前趋图，并且任选地通过提供允许用户对计算任务和 / 或有效排序的选择进行修改的用户接口。

[0084] 代码分析模块 103 可以使用本领域中已知的任意适当的用于对源代码进行解析的技术，例如传统编译器在解析源代码以便识别相应编程语言（如函数、条件、循环等）的结构组件时所使用的技术的子集。所述代码分析模块可以处理全部输入代码或者仅处理其子集。

[0085] 特别地，代码分析模块 103 可以采用程序分析领域中已知的任意适当的用于依赖性分析并且产生前趋图或者指定对执行次序的约束的所识别依赖性关系的另一种适当表示的技术，所述约束可以是保证语义等同性所要求的。这样的技术示例例如在 Ken Kennedy & RandyAllen(2001)，Morgan Kaufmann 的“Optimizing Compilers for ModernArchitectures：A Dependence-based Approach”中有所描述。

[0086] 通常，依赖性分析产生语句和 / 或指令之间的执行次序约束。例如，如果语句 S1 必须在语句 S2 之前执行，就可以说语句 S2 依赖于语句 S1。

[0087] 函数语言中缺少状态会显著促进依赖性分析。在纯函数程序中，不存在侧放作用 (side effect)，即函数仅依赖于其变元。例如，假设 A 至 F 为纯函数，则以下程序产生如图 2 所示的前趋图：

[0088] a = A(x)

[0089] b = B(a)

[0090] c = C(b)

[0091] d = D(a)

[0092] e = E(b, d)

[0093] f = F(d)

[0094] 如以上所提到的，可以对程度不同的细粒度任务执行依赖性分析。这些方法中的一些可以完全自动执行，而其它则需要一定程度的用户输入。

[0095] 例如，对于诸如 C、C++ 和 Java 之类的命令式语言而言，存在若干公知的依赖性分析方法和细粒度前趋图，例如以便与指令调度以及循环的矢量化或并行化结合使用。

[0096] 依赖性分析技术的其它示例包括用于分析函数的侧放作用的技术。识别无侧放作用的（“纯”）函数可被认为是这种分析的特殊情况。

[0097] 代码分析模块 103 将与所识别的计算任务相关的信息 104（例如，以指向其各自在代码中的位置的指针的形式）和所生成的前趋图 200 转发到代码变换模块 105。

[0098] 变换模块 105 对输入代码进行变换以便产生经变换的源代码 109。特别地，变换模块 105 在前趋图 200 所限定的前趋约束下将程序代码插入到执行计算任务的选择和调用过程的源代码中。所述变换模块可以进一步修改与各个任务相对应的源代码，例如通过插入用于返回到单独代码选择和分派函数的代码或者通过插入用于选择和调用后继任务的代码。此外，所述变换模块可以将与各个计算任务相对应的代码翻译成相应编程语言的预定结构实体。例如，所述变换模块可以为每个计算任务生成各自的函数 / 过程 / 子例程。

[0099] 随后可以对经变换的源代码 109 进行进一步处理，例如编译、链接、压缩、编码等。

[0100] 用于对任务进行排序的代码可作为单独的选择和分派函数 / 过程 / 子例程被插入，其作为对软件组件的实际任务的辅助，如图 3 所示。

[0101] 图 3 示出了输入源代码模块 101、相应源代码模块 104（其中被标记为 T1, T2, ..., TN 的多个计算任务例如已经被代码分析模块 103 识别）、以及例如由变换模块 105 所生成的经变换的源代码模块 109 的示例。所述输入源代码模块包括适于使得数据处理系统执行多个计算任务的源代码 301。经变换的源代码模块包括经变换的计算任务 T1′, T2′, ..., TN′，以及选择和分派函数 310。

[0102] 所插入的选择和分派函数 310 可简单地通过在遵循前趋图所限定的前趋约束的同时逐个选择计算任务来对它们进行排序。例如，所述前趋图可由适当初始化的数据结构来表示，或者所述约束可以在所插入的函数 310 中直接编码。对经变换的计算任务进行修改以使得它们在完成时向选择和分派函数 310 返回程序控制。

[0103] 选择和分派函数 310 可以保持所执行程序的状态以便在选择任务的同时保持对前趋的追踪。在一个实施例中，给定程序执行的特定状态，选择和分派函数使用函数 κ 来

选择“下一个”任务 t。此外，所述选择和分派函数使用函数 δ_t 在选择了任务 t 之后更新状态。如果 s_0 表示初始状态且 N 表示任务总数，则可如以下伪代码所示的那样对任务进行排序，所述伪代码说明了选择和分派函数的示例：

[0104] $S := s_0$

[0105] for i = 1,2,...,N do

[0106] t := κ (s)

[0107] execute task t

[0108] $s := \delta_t(s)$

[0109] od

[0110] 因此，当前的状态变量 s 最初被初始化为初始程序状态 s_0。之后，循环被执行 N 次，其中每次迭代都通过执行函数 κ (s) 选择下一个任务 t。接着执行所选择的任务 t，并且通过执行函数 δ_t 来更新程序状态。

[0111] 将会意识到的是，执行状态的集合以及函数 κ 和 δ_t 可以以多种方式来定义。在一个实施例中，从用来拓扑归类的标准方法得出它们的定义（例如，参见 Niklaus Wirth 的“Algorithms+Data Structures = Programs”，第 182-189 页，Prentice-Hall，Englewood Cliffs，新泽西，1975），其被适当修改以便提供对任务的随机化选择。为此，注意到所有有效执行次序与相应前趋图的拓扑排序相对应是有用的。

[0112] 在一个实施例中，执行状态由三元组 (T, Z, r) 来表示，其中 T 表示留待排序的任务的集合（即，前趋图中的节点），Z 表示没有前导有待排序的其余任务的集合，并且 r 表示（伪）随机种子。由此 Z 是 T 的子集，其包含用于在迭代过程的下一步骤中执行的候选任务。函数 κ 使用（伪）随机种子 r 选择 Z 中的一个候选。函数 δ_t 例如以以下方式更新状态：

[0113] $\delta_t(T,Z,r) = (T', Z', r')$ where

[0114] $T' = T \setminus \{t\}$

[0115] $Z' = (Z \setminus \{t\}) \cup \{u \mid u \in succ(t), \mathbf{p \notin T'} \text{ for all } p \in pred(u)\}$

[0116] $r' = prng(r)$

[0117] 从任务集合 T 移除任务 t 产生留待调度的任务的更新的集合 T′。还从用于在算法的下一步骤中执行的候选的集合 Z 移除任务 t，从而产生更新的集合 Z′。添加到候选集合中的是前趋图中 t 的后继 u，其所有前导 p 都已经被执行，即不包括在 T′ 中。通过函数 prng 更新（伪）随机种子 r，所述函数 prng 实现用于伪随机数生成的合适算法，例如线性同余法（例如，参见 D.E.Knuth.The Art of Computer Programming，Volume 2：Seminumerical Algorithms，第三版.Addison-Wesley，1997，ISBN 0-201-89684-2，Section 3.2.1：The Linear Congruential Method，第 10-26 页）、Mersenne Twister 算法（Makoto Matsumoto 和 TakujiNishimura：“Mersenne twister：a 623-dimensionally equidistributeduniform pseudo-random number generator”，ACM Trans.Model.Comput.Simul.，vol.8，nr.1，1998，第 3-30 页，ACM 出版社，纽约，NY，美国）或任意其它适当方法。作为选择，可以使用随机数生成硬件或者生成多个其它数据项的（伪）随机序列的任意其它适当方式。

[0118] 进一步注意到，在以上实施例中，子集 T 和 Z 这二者都被保存，由此提供了高效的

实施方式。然而将会意识到的是，在可替换实施例中，可以仅保存一个子集，原因在于 T 可以从 Z 得出，反之亦然。此外，将会意识到的是，可以使用其它表示，例如，已经被执行的任务集合及其在所执行任务的集合中没有后继的任务的子集。

[0119] 不同于通过以上所描述的单独函数执行任务选择和调用，如图 4 所示，每个任务可以计算其自己的后继。

[0120] 图 4 示出了输入源代码模块 101、相应源代码模块 104（其中被标记为 T 1，T2，...，TN 的多个计算任务例如已经被代码分析模块 103 识别）、以及例如由变换模块 105 所生成的经变换的源代码模块 109 的示例。经变换的源代码模块包括经变换的计算任务 T1′，T2′，...，TN′，其中每一个包括附加的一个或多个用于选择和调用后继任务的程序语句 410-1，410-2，...，410-N。经变换的源代码模块可进一步包括用于初始化例如包括随机序列的程序状态并且用于调用经变换的计算任务中的一个初始任务的初始化例程 411。

[0121] 例如，这样的分布式任务排序可通过使用源到源变换器（例如，代码变换模块 105）将函数 κ 和 δ_t 的评估集成到每个任务的源代码中来实现。通过对初始状态应用 κ 在初始化函数 411 中选择所要执行的第一个任务，也就是说：

[0122] $t_1 = \kappa(s_0)$

[0123] execute task t_1 and provide the state s_0 as a parameter

[0124] 所述变换模块对与每个任务相对应的源代码进行变换，以使得任务取当前状态 s 作为输入参数，使用 δ_t 更新状态 s 并使用函数 κ 选择下一个要执行的任务 t。每个任务的源代码由此被补充以源代码 410-1，410-2，...，410-N 以用于在执行任务的正常计算之后执行后面的步骤。

[0125] $s' = \delta_t(s)$

[0126] $t' = \kappa(s')$

[0127] execute task t′ and provide the state s′ as a parameter

[0128] 注意到，程序状态的更新（例如以上示例中 δ_t 的评估）能够被专用于 (specialized) 特定任务 t 的背景中，即可以向不同任务插入不同的函数 δ_t。特别地，给定特定前趋图，给定任务的后继以及那些后继的前导对于变换模块是已知的。因此，所述变换模块可以生成专用函数 $\delta_t(s)$。针对任务 t 提供专用函数 δ_t 允许对前趋图的表示进行分布。

[0129] 如现在将参见图 5 进行描述的，变换模块 105 和代码分析模块 103 可进一步对一个或多个任务的源代码进行复制。以这种方式，所述过程可以生成相同任务的多个实例。

[0130] 图 5 示意性地图示了产生包括计算任务的多个实例的经变换源代码的源代码变换的另一示例。特别地，图 5 示出了输入源代码模块 101、相应源代码模块 104（其中被标记为 T1，T2，...，TN 的多个计算任务例如已经被代码分析模块 103 被识别）、以及例如由变换模块 105 所生成的经变换的源代码模块 109 的示例。所述经变换的源代码模块与图 4 的经变换的源代码模块类似，并且其包括经变换的计算任务，其中每一个都包括用于选择和调用后继任务的附加程序语句。所述经转换的源代码模块进一步包括用于对例如包括随机序列的程序语句进行初始化并且用于调用经变换的计算任务中的一个初始任务的初始化例程 411。在图 5 的示例中，变换模块 105 生成每个所识别计算任务的两个实例，以使得来自任务的每个所生成实例与从其中生成该实例的任务执行相同的计算任务。特别地，变换模

块 105 从任务 T1 生成实例 T 1a 和 T1b，从任务 T2 生成实例 T2a 和 T2b，并且从任务 TN 生成实例 TNa 和 TNb。即使图 5 的示例为每个计算任务示出了两个实例，但是将会意识到的是，所述变换模块可以生成不同数目的实例，例如三个、四个或者甚至更多的实例。将会进一步意识到的是，所述变换模块可以为不同计算任务生成不同数目的实例，和 / 或仅为计算任务的子集生成多个实例。

[0131] 所述变换模块可以简单地通过复制原始计算任务的源代码来生成多个实例。可以在所述变换模块向给定任务的一个或多个实例应用保留语义的不同变换时实现额外的多样性。结果，不仅地址有所变化，从这些地址取出的指令也有所变化。可在这种背景下使用的保留语义的变换的适当示例例如在 Alfred V. Aho、Ravi Sethi 和 Jeffrey D. Ullman 的“Compilers-Principles, Techniques and Tools”，Adison-Wesley，Reading，美国，1986 或者 Christian S. Collberg 和 Clark Thomborson 的“Watermarking, Tamper-proofing, and Obfuscation-Tools forSoftware protection”，IEEE Transactions on Software engineering，28 :6(2002 年 6 月) 中有所公开。

[0132] 所述变换模块可进一步将每个所生成的复制实例表示为前趋图中的各个节点以获得经修改的前趋图。所插入的用于选择任务的代码由此在软件组件的特定执行中 (伪) 随机地选择每个任务的一个实例。以这种方式，不仅执行的次序有所变化，而且从其执行代码的存储器地址也在软件组件的执行之间发生变化。在该示例中，所生成的计算任务的实例被生成以便包括用于选择和调用根据所修改的前趋图所选择的后继任务的各自附加的一个或多个程序语句 510-1a，510-1b，510-2a，510-2b，...，510-Na，510-Nb。

[0133] 将会意识到的是，计算任务的多个实例的生成也可以在具有中央任务选择和分派函数的实施例中实施。

[0134] 因此，在上文中已经描述了用于生成经变换的程序代码的过程的不同实施例，其使得计算任务以随机化次序执行，例如如数字或其它数据项的 (伪) 随机序列所定义的次序。如以上所提到的，可以针对软件组件的不同安装而等同地或不同地生成伪随机序列。

[0135] 典型地，利用初始状态 - 也称作种子 - 作为输入对伪随机数生成器进行初始化。因此，当伪随机序列由伪随机数生成器 (PRNG) 生成时，可以通过利用不同软件安装中的不同种子对 PRNG 进行初始化来生成不同的伪随机数序列。例如，可以从唯一标识安装的序列号或另一标识符得到种子。作为选择，PRNG 的种子可以被选择为指示安装的预定组，例如提供给给定顾客的安装、软件组件的不同语言版本等。

[0136] 因此，初始状态 s_0 可包括伪随机种子，其对于软件组件的特定安装是唯一的，但是其并不在相同安装的不同执行之间发生变化。因此，给定包括例如出现诸如网络响应之类的输入的时间的相同激励，使用确定性的伪随机数生成器，在执行轨迹在相同安装的不同执行之间并不发生变化的意义上产生可能对于安装是唯一的执行轨迹。

[0137] 作为选择，可以选择初始状态以使得因执行而异 (至少概率很高)，例如通过选择种子作为日期和时间、实时时钟和 / 或其它变化信息的函数。在一些实施例中，随机化序列可以由硬件随机数生成器来生成，从而相同安装的每次执行可能产生不同轨迹。

[0138] 因此，在这里所描述的过程的一个实施例中，代码生成过程

[0139] - 将用于对计算任务进行排序的代码集成在计算任务自身之内，并且针对每个任务对状态更新进行专用化，

[0140] - 生成任务的至少一个子集中的每一个的若干实例，并且对每个实例应用保留语义的不同变换，

[0141] - 生成使得执行轨迹对于软件组件的特定安装为固定，但是在不同安装之间有所不同的代码。

[0142] 使用关键代码的多个实例，在原始任务中集成选择函数并且对实例应用保留语义的变换产生额外的多样性，这进一步使得分析复杂化。

[0143] 图 6 示出了用于对程序代码进行篡改保护的系统的示意性框图。所述系统包括数据处理系统 600，例如诸如 PC 之类的计算机。所述数据处理系统包括处理单元 621，例如诸如计算机的 CPU 之类的微处理器。所述处理单元 621 连接到存储设备 620，所述存储设备 620 诸如硬盘、存储卡接口、光盘设备等。所述处理单元 621 被适当编程为执行自动软件工具 622，诸如源到源变换器。例如，自动软件工具 622 可适于使得处理单元 621 从存储设备 620 加载程序代码的输入表示（例如源代码），并且执行这里所描述的方法的步骤。处理单元 621 可接着在存储设备 620 上存储经变换的源代码。作为选择，处理单元可以使得经变换的源代码可用于在数据处理系统上执行的其它程序（例如编译器），和 / 或经由另一适当接口输出结果。在又一种可替换实施例中，所述自动软件工具可以直接处理（例如编译）经变换的源代码。

[0144] 虽然已经详细示出并描述了一些实施例，但是本发明并不局限于此，而是还可以以随后权利要求中所定义主题的范围内的其它方式来实现。

[0145] 这里所描述的方法、产品装置和设备能够利用包括若干不同部件的硬件来实施，并且能够利用适当编程的微处理器来实施。在列举了若干装置的设备权利要求中，这些装置中的一些能够由同一项硬件（例如适当编程的微处理器）、一个或多个数字信号处理器等来实现。在不同从属权利要求中相互引用特定措施或者在不同实施例中描述特定措施的这一事实并不表示不能有利地使用这些措施的组合。

[0146] 应当强调的是，当在本说明书中使用时，术语“包括 / 包含”被认为是指定所提到特征、整数、步骤或组件的存在，但是并不排除存在或添加一个或多个其它特征、整数、步骤、组件或其组合。

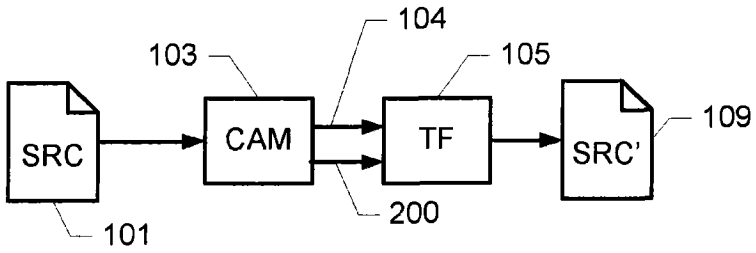

104
103
105
SRC
CAM
TF
SRC'
109
200
101

图 1

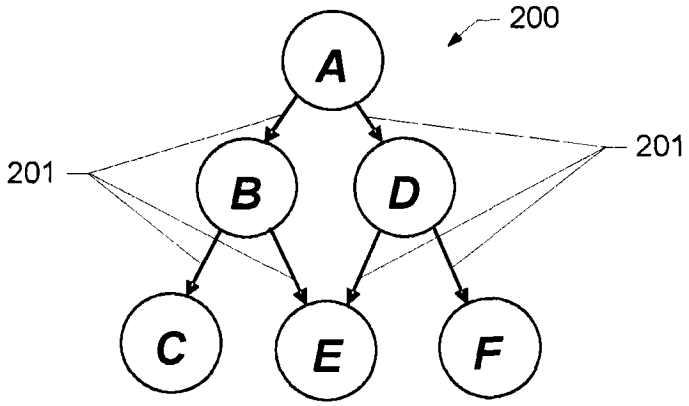

200
A
201
201
B
D
C
E
F

图 2

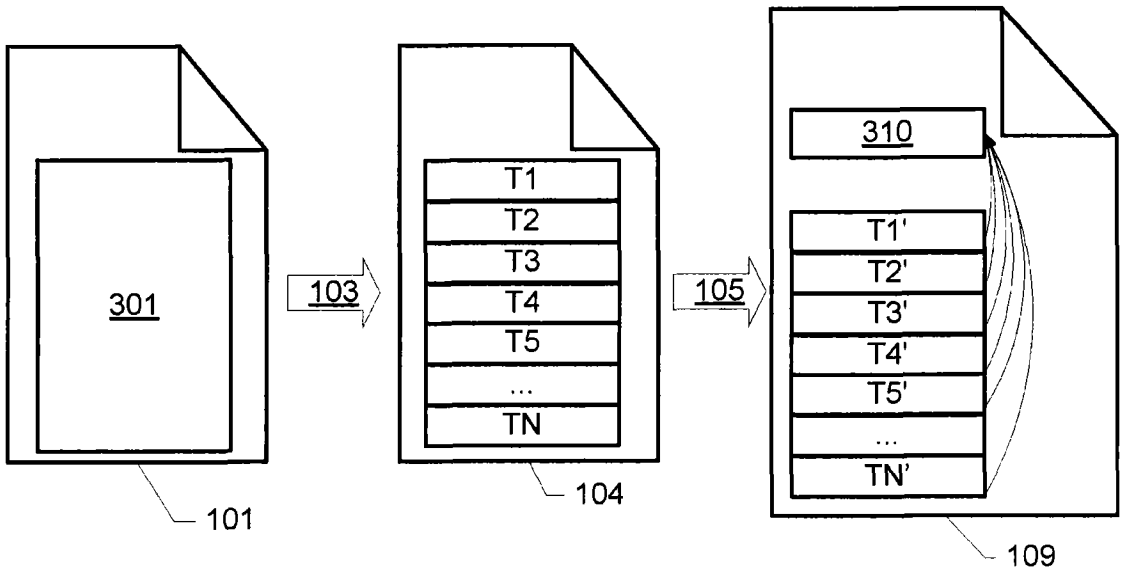

301
103
T1
T2
T3
T4
T5
...
TN
105
310
T1'
T2'
T3'
T4'
T5'
...
TN'
101
104
109

图 3

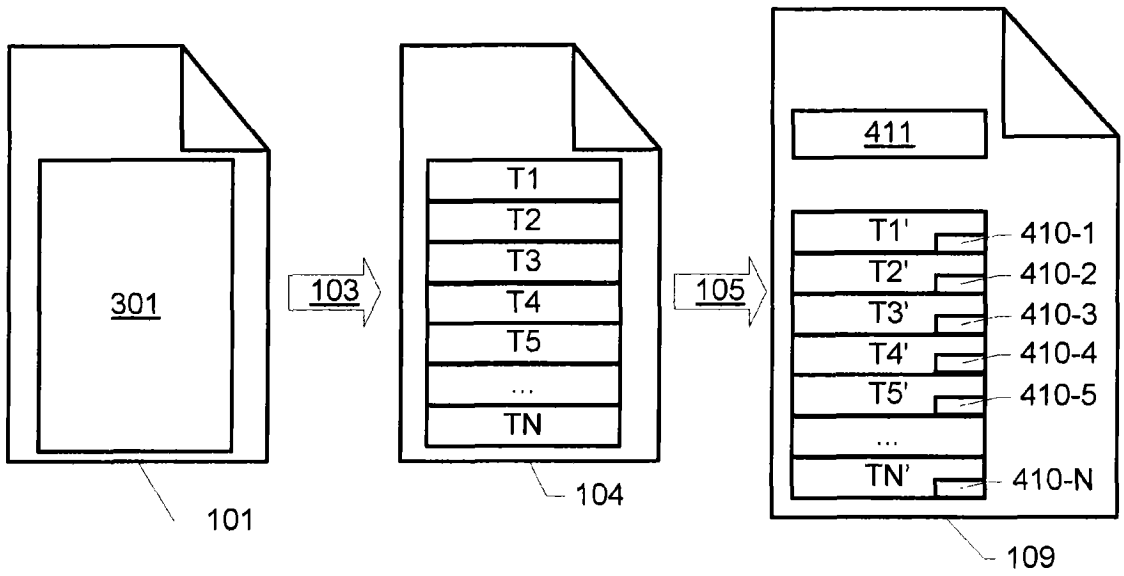

301
101
103
T1
T2
T3
T4
T5
...
TN
104
105
411
T1'
T2'
T3'
T4'
T5'
...
TN'
410-1
410-2
410-3
410-4
410-5
410-N
109

图 4

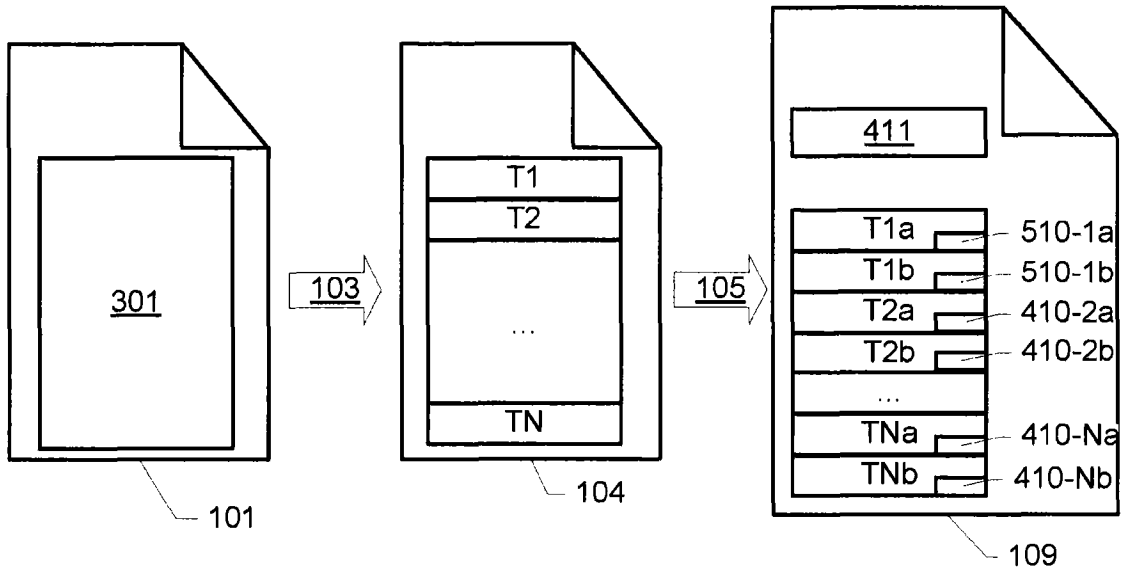

301
101
103
T1
T2
...
TN
104
105
411
T1a
T1b
T2a
T2b
...
TNa
TNb
510-1a
510-1b
410-2a
410-2b
410-Na
410-Nb
109

图 5

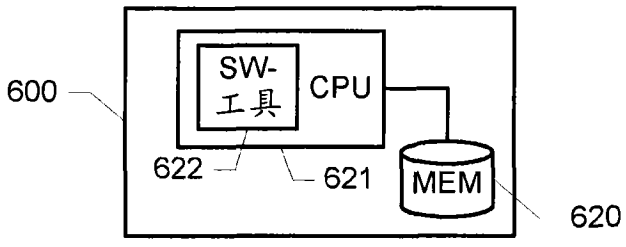

600
SW-
工具
CPU
622
621
MEM
620

图 6