



(12)发明专利申请

(10)申请公布号 CN 106415566 A

(43)申请公布日 2017. 02. 15

(21)申请号 201480079426.9

(51)Int.Cl.

(22)申请日 2014.03.31

G06F 21/12(2006.01)

(85)PCT国际申请进入国家阶段日
2016.11.30

G06F 21/14(2006.01)

G06F 21/52(2006.01)

(86)PCT国际申请的申请数据

PCT/EP2014/056422 2014.03.31

(87)PCT国际申请的公布数据

W02015/149828 EN 2015.10.08

(71)申请人 爱迪德技术有限公司

地址 荷兰霍夫多普

(72)发明人 B.西斯塔尼

(74)专利代理机构 中国专利代理(香港)有限公
司 72001

代理人 徐红燕 陈岚

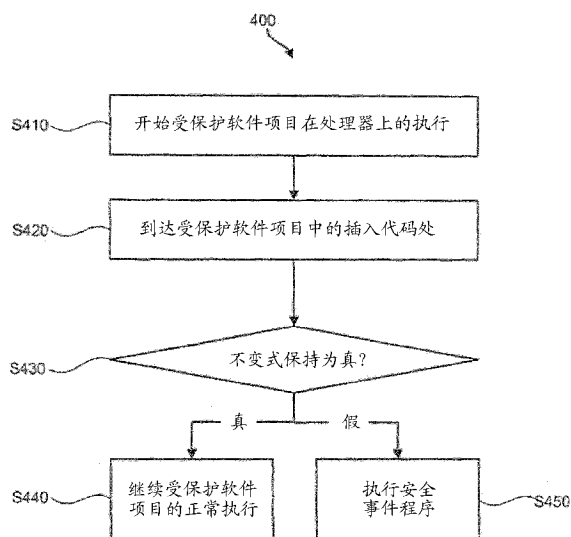
权利要求书1页 说明书14页 附图4页

(54)发明名称

保护软件项目

(57)摘要

描述了一种保护软件项目的方法。所述方法包括(a)识别在软件项目中的指定点处保持为真的不变式;以及(b)通过在软件项目中的指定点处插入代码来生成受保护软件项目。所述代码在被处理器执行时被布置成检查不变式是否保持为真,并且响应于不变式不保持为真而被布置成调用安全事件程序。还描述了一种被布置成执行保护软件项目的方法的装置。还描述了一种计算机程序,其在被处理器执行时引起处理器实现保护软件项目的方法。另外描述了存储上述计算机程序的计算机可读介质。此外,描述了在第一位置处包括代码的软件项目,其中,所述代码在被处理器执行时被布置成检查不变式在第一位置处是否保持为真,并且响应于不变式不保持为真而被布置成调用安全事件程序。



1. 一种保护软件项目的方法,所述方法包括:
识别在软件项目中的指定点处保持为真的不变式;以及
通过在软件项目中的指定点处插入代码来生成受保护软件项目,其中,所述代码在被处理器执行时被布置成检查不变式是否保持为真,并且响应于不变式不保持为真而被布置成调用安全事件程序。
2. 权利要求1的方法,其中,所述安全事件程序在被调用时被布置成引起处理器进行以下各项中的一个或多个:
 - (a) 中断受保护软件项目的执行;
 - (b) 在安全事件程序调用之后的预定时间段内防止受保护软件项目的执行;
 - (c) 防止受保护软件项目的未来执行;
 - (d) 确保由受保护软件项目输出的数据被破坏;
 - (e) 向软件项目的提供方、受保护软件项目的提供方以及另一实体中的至少一个提供关于安全事件程序的调用的通知。
3. 权利要求2的方法,其中,所述通知包括识别执行受保护软件项目的实体的数据。
4. 任何前述权利要求的方法,其中,所述在软件项目中的指定点处插入代码至少部分地自动地执行。
5. 任何前述权利要求的方法,其中,识别不变式的步骤至少部分地自动地执行。
6. 权利要求5的方法,其中,识别不变式的步骤包括使用静态程序分析工具。
7. 任何前述权利要求的方法,其中,识别不变式的步骤包括:
识别多个不变式,其中的每一个在软件项目中的各指定点处保持为真;以及
从该多个不变式中选择不变式为所述不变式。
8. 任何前述权利要求的方法,其中,所述方法还包括识别第二不变式,其在软件项目中的第二指定点处保持为真,并且其中,生成受保护软件项目的步骤还包括在软件项目中的第二指定点处插入第二代码,其中,所述第二代码在被处理器执行时被布置成检查第二不变式是否保持为真,并且响应于第二不变式不保持为真而被布置成调用第二安全事件程序。
9. 任何前述权利要求的方法,其中,所述方法还包括选择要保护的软件项目的一部分,并且其中,识别不变式的步骤包括识别在软件项目的所述部分中的指定点处保持为真的不变式。
10. 任何前述权利要求的方法,其中,所述方法还包括混淆受保护软件项目。
11. 一种被布置成执行根据权利要求1至10中的任一项所述的方法的装置。
12. 一种计算机程序,其在被处理器执行时引起处理器实现根据权利要求1至10中的任一项所述的方法。
13. 一种存储根据权利要求12所述的计算机程序的计算机可读介质。
14. 一种在第一位置处包括代码的软件项目,其中,所述代码在被处理器执行时被布置成检查不变式在第一位置处是否保持为真,并且响应于不变式不保持为真而被布置成调用安全事件程序。

保护软件项目

技术领域

[0001] 本发明涉及用于保护软件项目的方法以及用于执行此类方法的装置和计算机程序。

背景技术

[0002] 众所周知的是经常针对软件项目发起攻击。攻击者可能希望获得包含在软件项目内的秘密信息(诸如密码密钥),目的是滥用该秘密信息(例如通过将密码密钥分发给其他人/系统,使得那些人/系统可以以未授权方式使用该密码密钥)。同样地,攻击者可能希望修改软件项目的执行流程。例如,软件项目可具有决策点,其检查软件项目的用户是否具有某些许可或访问权限——如果用户具有那些许可或访问权限,则软件项目可准予对某些功能或数据的用户访问,否则拒绝此类访问。攻击者可能希望尝试在此决策点处修改软件项目的执行,使得即使用户不具有许可或访问权限,软件项目仍准予对该某些功能或数据的用户访问。

[0003] 存在可以应用于初始软件项目以便生成受保护软件项目、旨在使得攻击者不可能(或者至少十分困难)成功地完成其攻击的众多众所周知的软件保护技术。

[0004] 本发明寻求提供一种用于保护软件项目的替换方法,其相比于现有技术的那些而言提供各种优势。

发明内容

[0005] 根据本发明的第一方面,提供了一种保护软件项目的方法。所述方法包括:(a)识别在软件项目中的指定点处保持为真的不变式;以及(b)通过在软件项目中的指定点处插入代码来生成受保护软件项目。所述代码在被处理器执行时被布置成检查不变式是否保持为真,并且响应于不变式不保持为真而被布置成调用安全事件程序。

[0006] 根据本发明的第二方面,提供了一种被布置成执行第一方面的方法的装置。

[0007] 根据本发明的第三方面,提供了一种计算机程序,其在被处理器执行时引起处理器实现第一方面的方法。

[0008] 根据本发明的第四方面,提供了一种存储第三方面的计算机程序的计算机可读介质。

[0009] 根据本发明的第四方面,提供了包括第一位置处的代码的软件项目,其中,所述代码在被处理器执行时被布置成检查不变式在第一位置处是否保持为真,并且响应于不变式不保持为真而被布置成调用安全事件程序。

[0010] 在所附权利要求中阐述了本发明的其它优选特征。

附图说明

[0011] 现在将参考附图以示例的方式描述本发明的实施例,在所述附图中:

图1示意性地图示出计算机系统的示例。

[0012] 图2示意性地图示出根据本发明的实施例的系统。

[0013] 图3示意性地图示出根据本发明的实施例的保护软件项目的方法。

[0014] 图4示意性地图示出已使用图3的方法来保护的受保护软件项目的执行。

具体实施方式

[0015] 在随后的描述中和在附图中,描述了本发明的某些实施例。然而,将认识到的是本发明不限于所描述的实施例,并且一些实施例可以不包括下面描述的特征中的所有。然而,将显而易见的是在可以在本文中进行各种修改和变更而不脱离如在所附权利要求中阐述的本发明的更宽泛的精神和范围。

[0016] 1-系统概述

图1示意性地图示出计算机系统100的示例。系统100包括计算机102。计算机102包括:存储介质104、存储器106、处理器108、接口110、用户输出接口112、用户输入接口114和网络接口116,其全部被通过一个或多个通信总线118链接在一起。

[0017] 存储介质104可以是任何形式的非易失性数据存储设备,诸如硬盘驱动、磁盘、光盘、ROM等中的一个或多个。存储介质104可存储用于处理器108执行以便使计算机102运行的操作系统。存储介质104还可存储一个或多个计算机程序(或软件或指令或代码)。

[0018] 存储器106可以是适合于存储数据和/或计算机程序(或软件或指令或代码)的任何随机存取存储器(存储单元或易失性存储介质)。

[0019] 处理器108可以是适合于执行一个或多个计算机程序(诸如存储在存储介质104上和/或存储器106中的那些)的任何数据处理单元,所述计算机程序中的某些可以是根据本发明的实施例的计算机程序或者在被处理器108执行时引起处理器108实现根据本发明的实施例的方法并将系统100配置成是根据本发明的实施例的系统的计算机程序。处理器108可包括单个数据处理单元或者并行地或相互合作地操作的多个数据处理单元。处理器108在实现用于本发明的实施例的数据处理操作时可向和/或从存储介质104和/或存储器106存储数据和/或读取数据。

[0020] 接口110可以是用于提供到在计算机102外部或可从其移除的设备122的接口的任何单元。设备122可以是数据存储设备,例如光盘、磁盘、固态存储设备等中的一个或多个。设备122可具有处理能力——例如,该设备可以是智能卡。接口110因此可根据其从处理器108接收到的一个或多个命令从设备122访问数据或向设备122提供数据或者与设备122对接。

[0021] 用户输入接口114被布置成从系统100的用户或操作者接收输入。用户可经由被连接到用户输入接口114或与之通信的系统100的一个或多个输入设备(诸如鼠标(或其它指针设备)126和/或键盘124)来提供此输入。然而,将认识到的是用户可经由一个或多个附加或替换输入设备(诸如触摸屏)向计算机102提供输入。计算机102可将经由用户输入接口114从输入设备接收到的输入存储在存储器106中以用于处理器108随后访问和处理,或者可将其径传递至处理器108,使得处理器108可以相应地对用户输入进行响应。

[0022] 用户输出接口112被布置成向系统100的用户或操作者提供图形/视觉和/或音频输出。同样地,处理器108可被布置成命令用户输出接口112形成表示期望图形输出的图像/视频信号,并将此信号提供给被连接到用户输出接口112的系统100的监视器(或屏幕或显

示单元)120。附加地或替换地,处理器108可被布置成命令用户输出接口112形成表示期望音频输出的音频信号,并且将此信号提供给连接到用户输出接口112的系统100的一个或多个扬声器121。

[0023] 最后,网络接口116提供用于使计算机102从一个或多个数据通信网络下载数据和/或向其上传数据的功能(未示出)。

[0024] 将认识到的是在图1中示出并在上文描述的系统100的架构仅仅是示例性的,并且在本发明的实施例中可使用具有不同架构(例如具有比图1中所示的更少的组件或者具有除图1中所示之外的附加和/或替换组件)的其它计算机系统100。作为示例,计算机系统100可以包括以下各项中的一个或多个:个人计算机;服务器计算机;移动电话;平板电脑;膝上计算机;电视机;机顶盒;游戏控制台;其它移动设备或消费者电子设备等。

[0025] 图2示意性地图示出根据本发明的实施例的系统200。系统200包括:软件生成系统210;软件保护系统250;用户系统280;以及网络290。

[0026] 软件生成系统210包括(或者执行或使用)生成初始软件项目220的软件生成工具212。软件生成工具212可以是例如软件生成系统210的处理器执行的软件应用程序。软件生成系统210可被布置成自主地生成初始软件项目220;附加地或替换地,软件生成系统210可被布置成在至少部分地编写构成初始软件项目220的一部分的软件代码的一个或多个软件开发者的控制下生成初始软件项目220。用于生成或开发软件项目的工具是众所周知的,并且因此在本文中不会更详细地描述。

[0027] 初始软件项目220可包括源代码、目标代码、可执行代码和二进制代码中的一个或多个。可用一个或多个编程语言对初始软件项目220进行编程或编写,所述编程语言可包括编译编程语言和/或解释或脚本编程语言。初始软件项目220可包括一个或多个模块或软件组件或计算机程序,其可被呈现或存储于一个或多个文件内。事实上,初始软件项目220可以是整个软件应用程序、软件库或一个或多个软件函数或程序的整体或一部分或者其之间的任何地方(如本领域的技术人员将认识到的)。

[0028] 初始软件项目220在被处理器执行时被布置成执行(或者引起处理器执行)基于一个或多个数据项目的数据处理。每个数据项目可以分别地是任何类型的数据,诸如音频数据、视频数据、多媒体数据、文本数据、财务数据、一个或多个密码密钥、数字权限管理数据、有条件访问数据等。数据处理可包括以下各项中的一个或多个:(a)至少部分地基于一个或多个数据项目中的至少一个的决策;(b)安全相关功能;(c)访问控制功能;(d)密码功能;以及(e)权限管理功能。然而,将认识到的是除上述示例之外或作为其替换,数据处理可包括一个或多个其它类型的功能或操作。作为一个示例,数据处理可涉及提供对被接收和/或存储为加密内容的内容(诸如音频和/或视频数据)的用户访问,其中,只有当用户具有适当的访问许可/权限时才为用户提供对内容的访问。一个或多个数据项目因此可包括:加密内容、关于用户和/或用户系统280的细节或其标识;指定一个或多个许可和/或权限的数据;以及一个或多个密码密钥(其可以被存储为初始软件项目220的一部分)。因此,期望的是保护初始软件项目220,使得攻击者不能以未授权方式使用初始软件项目220以从而即使攻击者未被授权访问内容也获得对内容的访问,即防止攻击者绕过由初始软件项目220提供的有条件访问和/或数字权限管理功能(例如,通过确定一个或多个解密密钥或者规避涉及是否应为用户提供对内容的访问的初始软件项目220中的决策点或分支点)。将认识到的是当

然存在初始软件项目220可以执行的其它功能和/或初始软件项目220使用的初始软件项目220将期望(由于类似或可能替换的原因)保护其免于攻击者的其它信息。因此,如图2中所示,初始软件项目220被提供(或传输或传送)至软件保护系统250。

[0029] 软件保护系统250包括(或者执行或使用)软件保护工具252。软件保护工具252可以是例如软件保护系统250的处理器执行的软件应用程序。软件保护工具252被布置成接收初始软件项目220作为输入。软件保护工具252基于接收到的初始软件项目220而生成受保护软件项目260。稍后将会描述软件保护工具252用来生成受保护软件项目260的方法。

[0030] 软件生成系统210和软件保护系统250可被不同的实体运行或操作。因此,如图2中所示,软件保护系统250可将受保护软件项目260输出到软件生成系统210。用此模型,软件保护系统250向软件生成系统210提供保护服务。替换地,软件生成系统210和软件保护系统250可被同一实体运行或操作——事实上,软件生成系统210和软件保护系统250可构成单个系统的一部分(在图2中用虚线270示出),所述单个系统使用软件生成工具212来生成初始软件项目220并使用软件保护工具252通过生成受保护软件项目260来保护该初始软件项目220。

[0031] 因此,软件生成系统210和/或软件保护系统250可经由网络290将受保护软件项目260输出(或提供或传送)到用户系统280。然而,将认识到的是受保护软件项目260的分发可由图2中未示出的不同实体执行。

[0032] 还将认识到的是受保护软件项目260可在已经由软件保护系统250生成受保护软件项目260之后且在到用户系统280的分发之前经历各种附加处理。因此,将认识到的是在以下描述中,对受保护软件项目260的分发和使用的提及包括由对受保护软件项目260应用附加处理而引起的软件块的分发或使用。例如,受保护软件项目260可能需要被编译和/或与其它软件项目链接(例如,如果受保护软件项目260将构成要分发到用户系统280的较大软件应用程序的一部分)。然而,将认识到的是可不需要此类附加处理(例如,如果受保护软件项目260是准备好用于分发的最终JavaScript块的话)。

[0033] 网络290可以是适合于向用户系统280传送或传输受保护软件项目260的任何种类的数据通信网络。因此,网络290可包括以下各项中的一个或多个:局域网、广域网、城域网、互联网、无线通信网、有线或线缆通信网、卫星通信网、电话网等。软件生成系统210和/或软件保护系统250可被布置成经由任何适当的数据通信协议经由网络290与用户系统280通信。事实上,可经由物理介质(诸如被存储在一个或多个CD或DVD上)将受保护软件项目260提供给用户系统280,使得网络290于是可包括用于将物理介质在物理上递送给用户系统280的递送系统。

[0034] 用户系统280被布置成例如通过在用户系统280的一个或多个处理器上执行受保护软件项目280来使用受保护软件项目260。

[0035] 用户系统280可以是适合于执行受保护软件项目280的任何系统。因此,用户系统280可以是以下各项中的一个或多个:个人计算机、膝上计算机、笔记本、平板计算机、移动电话、机顶盒、电视、服务器、游戏控制台等。软件保护系统250和软件生成系统210可例如包括一个或多个个人计算机和/或服务器计算机。因此,用户系统280、软件保护系统250和软件生成系统210中的每一个可包括如上文参考图1所述的一个或多个相应系统100。

[0036] 将认识到的是虽然图2将系统200示为包括单个用户设备280、单个软件生成系统

210以及单个软件保护系统250,但系统200可包括多个用户设备280和/或多个软件生成系统210和/或多个软件保护系统250。

[0037] 2-软件保护技术

如上所述,软件保护工具252的目的是保护初始软件项目220的功能或数据处理和/或保护初始软件项目220所使用或处理的数据。特别地,受保护软件项目260将提供与初始软件项目220相同的功能或数据处理——然而,此功能或数据处理是以这样的方式在受保护软件项目260中实现的,即使得用户系统280的操作者不能以非预计或未授权方式从受保护软件项目260访问或使用此功能或数据处理(而如果向用户系统280提供了初始软件项目220,则用户系统280的操作者可能是能够以非预计或未授权方式访问或使用功能或数据处理的)。同样地,受保护软件项目260可以以受保护或已混淆方式存储秘密信息(诸如密码密钥)以从而使得攻击者更加难以(如果不是不可能的话)推断或访问该秘密信息(而如果向用户系统280提供了初始软件项目220,则用户系统280的操作者可能是能够从初始软件项目220推断或访问该秘密信息的)。

[0038] “白盒”环境是用于软件项目的执行环境,其中,假设软件项目的攻击者具有对于对其进行操作的数据(包括中间值)、存储器内容和软件项目的执行/过程流程的完全访问以及可见性。此外,在白盒环境中,假设攻击者能够例如通过使用调试器来修改对其进行操作的数据、存储器内容和软件项目的执行/过程流程——这样,攻击者可以对软件项目进行实验并尝试操纵其操作,旨在规避最初预计的功能和/或识别秘密信息和/或出于其它目的。事实上,甚至可假设攻击者知道由软件项目执行的底层算法。

[0039] 安全软件程序可被设计成抵抗白盒攻击,并且使用大范围的数据流程和控制流程变换来混淆由软件项目实现的功能。该保护适用于静态攻击和运行时攻击两者。在攻击情形中,敌方具有修改代码和数据两者的能力。

[0040] 存在可在受保护软件项目260内实现上述软件保护的许多方式,即,存在其中可对初始软件项目应用上述软件保护技术以获得受保护软件项目260的许多方式。特别地,为了从初始软件项目220生成受保护软件项目260,软件保护工具252可修改初始软件项目220内的一个或多个代码部分和/或可向初始软件项目220中添加或引入一个或多个新代码部分。用来进行这些修改的实际方式或者用来编写新代码部分的实际方式当然可以改变——毕竟,存在编写软件以实现相同功能的许多方式。

[0041] 期望的是防止攻击者修改软件项目的执行/控制流程,例如防止攻击者迫使软件项目在决策块之后采取一个执行路径而不是合法的执行路径。替换地/附加地,期望知道软件项目何时被攻击者篡改了,并且如果检测到攻击则采取适当的行动。

[0042] 3-软件项目的形式验证

软件项目(诸如计算机程序)的形式验证是用以证明软件项目的某种形式性质保持为真(诸如通信协议或算法(的实现)的正确性)的已知科学领域。验证是“形式的”,因为其基于数学上合理的技术方法。正确性(或其它性质)的证明通常是以合理的数学和逻辑系统中的形式证明的形式。形式证明是在软件项目的抽象数学模型上相对于某些形式规格或性质完成的。大多数一般形式验证系统是基于霍尔逻辑(也称为佛洛伊德-霍尔逻辑),而其它逻辑(诸如分离逻辑)被用于证明存储器相关性质。霍尔逻辑的中心特征是“霍尔三元组”,其描述了软件项目中的一段代码的执行如何改变计算的状态。霍尔三元组是以下形式的:

{P} C {Q}

其中,P和Q是断言且C是命令。断言是置于软件项目中以指示开发者认为谓词在该位置处始终为真的谓词(真-假语句)。如果断言在运行时间评估为假,则这导致断言失败,其可例如引起软件项目的执行异常中断。在霍尔逻辑中,P称为前条件且Q称为后条件:当满足前条件P时,执行命令C建立后条件Q。相关地,不变式是在软件项目的执行期间或者在其某个部分期间可以取决于其而为真的条件。其是在执行中的已知点或位置处始终保持为真的逻辑断言。换言之,不变式在形式上被定义为被证明在软件项目的执行中在至少一个特定点处保持为真的谓词。将理解的是还可以在其它逻辑系统(例如分离逻辑,如上所述)中定义和使用不变式。

[0043] 在“防御性编程”中,断言意图作为叙述不变式在软件项目中的特定点处保持为真的文档。断言还在编程语言中用来帮助在开发期间捕捉错误假设。一旦此类断言语句被添加到代码,则基于验证的系统自动地检查其在运行时间是否保持为真。如果断言并不保持为真,则验证系统在运行时间生成错误。在C编程语言的`assert.h`标准库中定义的宏`assert()`实现用于C的简单验证系统。然而,到目前为止,断言(或类似物)未被用于保护软件项目。

[0044] 存在用以自动地识别/检测不变式并将其公式化的许多方式/工具,但其一般地分成两个阵营:运行时工具和编译时工具。“抽象解释”是用于落入编译时工具阵营的非常尖端的分析工具的理论基础。抽象解释可以用来识别范围从简单到较复杂的不变式。抽象解释是在考虑到特定性质的情况下而完成的。用抽象域的选择来表达性质的种类。下面描述某些示例性抽象域和每个域适合于的性质的类型。

[0045] 可以使用数值抽象域来发现软件项目中的程序变量的数值性质。例如,使用符号抽象域来计算软件项目中的各种点处的一个或多个程序变量的符号。因此,在符号抽象域中的一个示例中,前条件P可断言特定程序变量x在命令C之前是正的,并且后条件Q可断言上述程序变量x在命令C之后是负的。在本示例中,让我们假设命令C将x的值(其最初是正的)设定成另一值y,其中,y是负的。因此,本示例中的霍尔三元组将是:

$$\{x \text{ is '+'} \} \ x = y \ \{x \text{ is '-'} \}$$

区间抽象域更加精确,并且被用来根据程序变量x落入其中的区间或范围来识别不变式。因此,在区间抽象域中的一个示例中,前条件P可断言特定程序变量x在命令C之前落入区间[2, 8]中,并且后条件Q可断言x在命令C之后落入区间[-7, -2]中。再次地,让我们假设命令C将x的值设定成另一值y。因此,本示例中的霍尔三元组将是:

$$\{2 < x < 8 \} \ x = y \ \{-7 < x < -1 \}$$

关系抽象域甚至更加精确,因为其考虑到程序变量之间的关系。例如,线性方程抽象域将识别 $\{ax + by = m\}$ 形式的不变式;多面体抽象域被用来识别 $\{ax + by \geq m\}$ 形式的不变式;并且椭圆体抽象域被用来识别 $\{a(x^2) + b(y^2) + c(x^2y) \leq m\}$ 形式的不变式。关系数值抽象域的一些另外的示例对于整数、凸多面体、“八边形”以及差分有界矩阵是同余关系。可通过考虑上述抽象域(及任何其它的)的组合来识别另外的不变式。

[0046] 4—一种保护软件项目的方法

已发明性地认识到可以将断言(或类似物)用于保护软件项目,而不仅仅用于软件项目

的形式验证。

[0047] 因此,如在图3中示意性地所示,本发明提供了一种保护软件项目(诸如上述软件项目220)的方法300。方法300包括识别在软件项目中的指定点处保持为真的不变式的步骤S310。方法300还包括通过在软件项目中的指定点处插入代码来生成受保护软件项目(诸如上述受保护软件项目260)的步骤S320。所述代码在被处理器执行时被布置成检查不变式是否保持为真,并且响应于不变式不保持为真而被布置成调用安全事件程序。

[0048] 方法300可包括生成软件项目的可选初始步骤S305。此步骤可由图2中所示的软件生成系统执行。本方法还可包括混淆受保护软件项目和/或应用一个或多个另外的软件保护技术的可选步骤S325。优选地,在不变式识别步骤S310之后执行混淆步骤S325(并且更优选地,还在生成受保护软件项目的步骤S320之后执行混淆步骤S325),使得可使用软件项目的更加基础(即,更干净)的版本来执行不变式识别和代码插入。这使得更容易识别不变式,因为混淆代码比原始代码更长且更复杂/难处理。然而,将认识到的是在步骤S310和S320之前或者在步骤S310和S320之间执行混淆也将是可能的。在某些实施例中,如上文所讨论的,可由软件保护工具252来执行步骤S310、S320和S325。处理器可形成图2中所示的用户系统280的一部分。在图3中,用虚线来指示可选步骤S305和S325。

[0049] 如上文所讨论的,不变式是在软件项目的执行期间或者在其某个部分期间可以取决于其而为真的条件。在上述方法300中,不变式是在软件项目的执行期间在指定点处可以取决于其而为真的条件。软件项目包括一个或多个程序变量,并且这些程序变量所取的值在软件项目的执行过程中可改变。可根据指定点处的软件项目中的至少一个程序变量的一个或多个性质或值来定义条件和/或其可根据指定点处的软件项目中的程序变量之间的一个或多个关系来定义。换言之,不变式识别在软件项目的执行期间在指定点处可以取决于其而为真的软件项目中的一个或多个程序变量的一个或多个性质和/或值(和/或其之间的关系)。重要的是,我们在这里参考未被攻击时的初始软件项目的执行;换言之,不变式在正常操作条件下的执行期间应在指定点处保持为真。因此,可以将不变式视为软件项目中的一个或多个程序变量的函数,并且可将该函数视为谓词,因为其根据其变量的值/性质而可以是真或假的。根据方法300,步骤S310识别(初始/未受攻击)软件项目中的指定点处保持为真的不变式。

[0050] 因此,方法300向软件项目中插入代码以检查不变式是否的确在运行时间在指定点处保持为真。因此,重要的是,“不变式检查”是在运行时(即在受保护软件项目的执行期间)而不是在编译时执行的。

[0051] 方法300有效地提供一种“不变式检查”生成系统,其使用基于形式验证的系统来产生软件项目所隐含的且对攻击者而言常常模糊的潜在复杂的不变式。这些不变式检查被添加到例如软件项目的源代码,并且所添加的代码针对数据的操纵且针对软件项目的控制流程的修改进行保护。

[0052] 期望的是不变式检查被以从而隐藏存在可策动安全事件程序的不变式检查的事实的方式被集成/插入到软件项目中。在这方面可使用已知的软件混淆技术,并且还参见W02013/142980和US6192475,并且其整个公开被通过引用结合到本文中。不变式检查生成系统以这样的方式插入添加的代码,即,使得稍后的软件混淆工具(例如,在步骤S325中)能够利用不变式检查语句来容易地生成对已变换数据且对已变换代码进行操作的不变式检

查的已混淆版本。

[0053] 图4示意性地图示出已根据图3的方法300受到保护的受保护软件项目的运行时执行400。在步骤S410处,受保护软件项目的执行开始。受保护软件项目的执行持续直至诸如执行到达插入代码处的时间(如在上述图3的步骤S320中所参考的)为止。在步骤S420处,受保护软件项目的执行到达插入代码处。如先前提到的,插入代码被布置成检查不变式是否保持为真。因此,在步骤S430处,此不变式检查发生。如果发现不变式保持为真,则受保护软件项目的执行在步骤S440处如常继续。换言之,受保护软件项目的执行以与针对初始软件项目将预期的相同方式继续;具体地,受保护软件项目的执行继续至紧接在不变式检查之后的代码。如果另一方面,不变式检查失败(即如果发现不变式为假),则插入的代码在步骤S450处调用安全事件程序。

[0054] 如上文所讨论的,不变式已被定义成使得其在软件项目(例如初始软件项目220)中的指定点处保持为真。因此,在不存在对受保护软件项目(例如,受保护软件项目260)的攻击或任何篡改的情况下,不变式应在受保护软件项目的执行期间在指定点处也保持为真(即,步骤S430处的检查应导致“真”的发现,从而导致步骤S440处的受保护软件项目的继续执行)。然而,如果存在对受保护软件项目的攻击(或篡改或破坏),则受保护软件项目或正被受保护软件项目处理或使用的数据可能已被修改,使得不变式在运行时在指定点处不再保持为真。因此,虽然不变式应(即意图)在受保护软件项目中在运行时保持为真,但可能的是不变式将由于对受保护软件项目的攻击而不保持为真。在这种情况下,已根据方法300受到保护的受保护软件项目的执行能够在运行时借助于失败的不变式检查(参见图4中的步骤S430和S450)而间接地识别到存在攻击。响应于运行时的失败的不变式检查,代码被布置成调用(或策动或实行或执行)如在图4的步骤S450处所示的安全事件程序。

[0055] 方法300是特别有用的,因为攻击者不太可能意识到不变式存在于受保护软件项目中,特别是如果不变式是基于程序变量的性质和/或值的相对复杂的组合的话。很明显,如果攻击者不知道存在不变式,则他们将不知道以不变式在指定点处仍保持为真的这样的方式修改受保护软件项目。因此,通过执行步骤S430的不变式检查,在不变式检查已借助于方法300被插入到软件项目中的情况下,意图不可检测的攻击变得可检测。

[0056] 将认识到的是方法300可不涉及到保护整个软件项目。在这种情况下,方法300还可包括选择要保护的软件项目的一部分。该部分可包括一个或多个单独的代码块。在本实施例中,指定点(即在该处不变式保持为真的点)位于软件项目的选定部分中。例如,很可能软件项目中的某些代码部分比其它部分更符合攻击的条件(即,对攻击者更加有吸引力)。特别地,攻击者很可能以潜在地使得攻击者能够获得包含在软件项目内的秘密信息(诸如密码密钥)的(一个或多个)代码部分或者在执行特定操作之前执行秘密信息的验证的代码部分为目标。因此,方法300可首先涉及到选择关于这些高度敏感性操作的那些代码部分。

[0057] 当在步骤S450处被调用时,安全事件程序可被布置成引起处理器按需采取预定行动。换言之,可酌情配置安全事件程序,使得发生期望的作为结果的行动。安全事件程序可由软件保护系统250配置。例如,软件保护工具252可向软件项目添加代码以便执行安全事件程序。在失败的不变式检查之后发生的预定行动可根据情况而不同,使得方法300非常灵活。例如,如果发现高度关键(非常重要)的不变式在运行时在相关指定点处并未保持为真,则相应地郑重的预定行动可能是适当的。例如,当安全事件程序被调用时,其可被布置成引

起处理器中断受保护软件项目的执行和/或在安全事件程序的调用之后的预定时间段内防止受保护软件项目的执行和/或防止受保护软件项目的未来执行。替换地,如果发现不那么关键的不变式在运行时在相关指定点处并未保持为真,则相应地不那么郑重的预定行动可能是适当的。例如,当安全事件程序被调用时,其可被布置成引起处理器确保由受保护软件项目输出的数据被破坏和/或提供关于安全事件程序的调用的通知。被破坏的输出数据可使得受保护软件项目不可用。可向软件项目的提供方(例如图2的软件生成系统210)和/或受保护软件项目的提供方(例如图2的软件保护系统250)和/或另一感兴趣实体提供该通知。该通知可包括识别执行受保护软件项目的实体的数据(例如,关于图2的用户系统280的数据或关于处理器的某些其它数据)。该通知优选地包括识别导致了失败检查的不变式的数据。

[0058] 在某些实施例中,不变式可仅在软件项目中的指定点处保持为真。替换地,不变式可附加地在除软件项目中的指定点之外的附加点处保持为真。例如,不变式可在软件项目的一部分的执行期间保持为真,其中,该部分包括软件项目中的一个或多个特定代码块。在这种情况下,可将指定点定义为一个或多个特定代码块中的任何点。换言之,如果不变式在一个或多个特定代码块的执行期间保持为真,则可在软件项目中的一个或多个特定代码块中的任何点处插入在步骤S320处要插入到软件项目中的代码,从而提供受保护软件项目。如果期望的话,事实上可在一个或多个特定代码块中的多个点处插入上述代码,从而提供多个软件保护不变式检查点。在某些情况下,不变式可在整个软件项目的执行期间(即纵贯软件项目的执行)保持为真。在这种情况下,可将指定点定义为软件项目中的任何点。再次地,如果期望的话,可在软件项目中的多个点处插入上述代码,从而提供多个软件保护不变式检查。

[0059] 在某些实施例中,在软件项目中的指定点处插入代码可至少部分地自动地执行。例如,可由图2的软件保护工具252自动地执行该插入。替换地/附加地,该插入可至少部分地手动地或用人类交互来执行。

[0060] 同样地,在某些实施例中,识别不变式的步骤S310可至少部分地自动地执行。例如,该识别可由图2的软件保护工具252自动地执行。例如,识别不变式的步骤可包括使用静态程序分析工具。示例性静态程序分析工具包括Astrée、CPAchecker、ÉCLAIR、Fluctuat、Polyspace、Coverity Prevent、Klocwork Insight、Parasoft Jtest、Parasoft C/C++ test、Red Lizard的Goanna以及Frama-C值分析。替换地/附加地,该识别可至少部分地手动地或用人类交互来执行。

[0061] 在某些实施例中,识别不变式的步骤S310包括识别多个不变式,其中的每一个在软件项目中的各指定点处保持为真,并且然后从所述多个不变式中选择不变式为所述不变式。当然,将认识到的是可从所述多个不变式中选择超过一个不变式,并且可关于所选定的不变式中的每一个应用方法300。下面给出了关于方法300对超过一个不变式的适用性的进一步的细节。

[0062] 在某些实施例中,可将不变式视为第一不变式,可将指定点视为第一指定点,并且可将插入代码视为第一代码,使得第一不变式在软件项目中的第一指定点处保持为真,并且使得在软件项目中的第一指定点处插入第一代码,从而生成受保护软件项目。在这种情况下,方法300还可包括识别在软件项目中的第二指定点处保持为真的第二不变式。第二不

变式可与第一不变式相同或者可不与其相同,并且第二指定点可与软件项目中的第一指定点相同或者可不与其相同。然而,为了避免冗余,如果第一和第二指定点是相同的,则第一和第二不变式不应是相同的,并且反之亦然。生成受保护软件项目的步骤S320还可包括在软件项目中的第二指定点处插入第二代码。第二代码在被处理器执行时被布置成检查第二不变式是否保持为真,并且响应于第二不变式不保持为真而被布置成调用第二安全事件程序。第二安全事件程序可与第一安全事件程序相同或不同。

[0063] 让我们考虑其中在软件项目中(例如使用诸如Frama-C值分析之类的静态程序分析工具)识别多个不变式 $I_1, I_2, \dots, I_n$ 的特定示例。在这种情况下,可选择已识别不变式的子集作为受保护软件项目的一部分用于检查:让我们假设选择三个不变式的子集,亦即 I_1 、 I_7 和 I_{22} (当然,将认识到的是可以同样地选择不不变式的不同子集,或者替换地可以选择不变式中的所有)。假设不变式 I_1 仅在软件项目中的指定点 P_1 处保持为真。假设不变式 I_7 仅在软件项目中的不同点 P_7 处保持为真。假设不变式 I_{22} 在软件项目中的多个点处保持为真,即点 P_1 、 P_7 和 P_{22} 。在本示例中,可以通过向软件项目中插入包括以下各项中的一个或多个的代码来生成受保护软件项目:

1. 在软件项目中的点 P_1 处插入的用以执行针对不变式 I_1 的不变式检查的代码
2. 在软件项目中的点 P_7 处插入的用以执行针对不变式 I_7 的不变式检查的代码
3. 在软件项目中的点 P_1 处插入的用以执行针对不变式 I_{22} 的不变式检查的代码
4. 在软件项目中的点 P_7 处插入的用以执行针对不变式 I_{22} 的不变式检查的代码
5. 在软件项目中的点 P_{22} 处插入的用以执行针对不变式 I_{22} 的不变式检查的代码

可以将上面所列的每个代码部分循序地插入到软件项目中从而渐进地生成受保护软件项目。替换地,可以将上面所列的每个代码部分同时地插入到软件项目中从而一次性地生成受保护软件项目。

[0064] 上面所列的代码部分中的至少一个可包括被插入到软件项目中的明示代码。例如,可以如下将IF-THEN语句插入到软件项目中:

```
IF (INVARIANT == FALSE) THEN
    CALL SECURITY_INCIDENT_PROCEDURE
ENDIF
```

替换地/附加地,上面所列的代码部分中的至少一个可包括与C编程语言中的`assert()`宏类似的相应的宏。然而,所插入的代码部分所调用的功能与`assert()`宏的不同,因为所插入的代码部分被布置成在适当的情况下调用安全事件程序。替换地/附加地,上面所列的代码部分中的至少一个可调用执行相应的不变式检查的函数。很明显,宏和/或函数还将需要可用于软件项目(例如在其中定义),如果使用的话。

[0065] 还设想被布置成执行方法300的装置。如上所述,此类装置可以是图2的软件保护系统250。还设想在被处理器执行时引起处理器实现方法300的计算机程序。另外,还设想存储此类计算机程序的计算机可读介质。

[0066] 5-示例

可以以被插入到不变式在其处保持为真的(一个或多个)指定点处的布尔条件的形式指定不变式检查。例如:

```
INARIANT_CHECK (2x+3y>=13)
```

其中, x 和 y 是程序变量, 并且“INARIANT_CHECK”是对执行不变式检查并在必要时调用安全事件程序的宏或函数的调用。在本示例中, 不变式是 $2x+3y \geq 13$ 。上文给出的示例性不变式检查利用在上述小节3中提到的多面体抽象域。不变式检查可以被单独地且在开发后并且在形式验证工具已在软件项目上运行并产生了不变式之后插入, 可能是由安全保证人员, 其可与创建了初始软件项目220的软件开发人员不同。

[0067] 考虑C类型编程语言中的以下函数“main”:

```
static char string[11]="A nice cat";
static char key[11]="ABCDEFGHILJ";

int main(void)
{
    for(int x=0; x<10; x++)
    {
        string[x]=string[x]^key[x];    <===== 行 8
    }
    return 0;
}
```

根据方法300, 有必要在步骤S310中识别不变式。如上文所讨论的, 这可使用诸如Frama-C值分析之类的静态程序分析工具来进行。下面是由Frama-C工具对在“main”函数的第8行中的语句之前的上面所列的函数“main”进行的值分析的结果的一部分:

函数:main

语句:8(xor. c 中的第8 行)

变量串具有类型“char [11]”。

[0068] 其是全局变量。

[0069] 其被参考且不获取其地址。

[0070] 在语句之前:

```
string[0] ∈ {0; 65}
  [1] ∈ {32; 98}
  [2] ∈ {45; 110}
  [3] ∈ {45; 105}
  [4] ∈ {38; 99}
  [5] ∈ {35; 101}
  [6] ∈ {32; 103}
  [7] ∈ {43; 99}
  [8] ∈ {40; 97}
  [9] ∈ {62; 116}
  [10] ∈ {0}
```

同样地,在“main”函数的第8行中的语句之后,Frama-C工具提供以下值分析:

```
string[0] ∈ {0}
  [1] ∈ {32; 98}
  [2] ∈ {45; 110}
  [3] ∈ {45; 105}
  [4] ∈ {38; 99}
  [5] ∈ {35; 101}
  [6] ∈ {32; 103}
  [7] ∈ {43; 99}
  [8] ∈ {40; 97}
  [9] ∈ {62; 116}
  [10] ∈ {0}
```

作为示例,可使用string[2]在第8行中的语句的执行之前或之后将不变式公式化。因此,下面我们示出了通过对上文所示的原始“main”函数执行方法300而产生的受保护软件项目的示例:

```

static char string[11]="A nice cat";
static char key[11]="ABCDEFGHILJ";

int main(void)
{
    for(int x=0; x<10; x++)
    {
        string[x]=string[x]^key[x];    <===== 行 8
        INVARIANT_CHECK (x==2 && string[x] >= 45 && string[x] <= 110)
    }
    return 0;
}

```

上文新添加的语句是在图3的步骤S320中参考的插入代码。在运行时,新添加的语句调用执行不变式检查的宏或函数,并且如果不满足关于string[2]的条件则调用安全事件程序。

[0071] 将理解的是上述代码是示例性的,并且本发明的方法300并不是C/C++编程语言所特定的。事实上,方法300甚至不是传统命令式语言所特定的。不变式存在于在以诸如声明式语言(例如DRM策略语言)之类的任何语言编写的程序中。如果对于特定编程语言而言(尚)不存在分析工具,则在方法300的步骤S310中以逐个情况为基础(至少部分地手动地)计算不变式是可能的。

[0072] 6-修改

将认识到的是所述的方法已被示为按照特定顺序实现的单独步骤。然而,技术人员将认识到这些步骤可被组合或者按照不同的顺序实现,同时仍达到期望的结果。

[0073] 将认识到的是可使用各种不同的信息处理系统来实现本发明的实施例。特别地,虽然附图及其讨论提供了示例性计算系统和方法,但是这些仅仅是为了在讨论本发明的各种方面时提供有用参考而提出的。可在任何适当的数据处理设备实现本发明的实施例,所述数据处理设备诸如个人计算机、膝上型计算机、个人数字助理、移动电话、机顶盒、电视、服务器计算机等。当然,已出于讨论的目的简化了系统和方法的描述,并且其仅仅是可被用于本发明的实施例的许多不同类型的系统和方法中的一个。将认识到的是逻辑块之间的边界仅仅是说明性的,并且替换实施例可将逻辑块或元素合并,或者可对各种逻辑块或元素施加功能的替换分解。

[0074] 将认识到的是可将上述功能实现为作为硬件和/或软件的一个或多个对应模块。例如,可将上述功能实现为一个或多个软件组件以便由系统的处理器执行。替换地,可将上述功能实现为硬件,诸如在一个或多个现场可编程门阵列(FPGA)和/或一个或多个专用集成电路(ASIC)和/或一个或多个数字信号处理器(DSP)和/或其它硬件布置上。如上所述的

方法步骤每个可由对应的相应模块实现；多个方法步骤可一起由单个模块实现。

[0075] 将认识到的是只要由计算机程序实现本发明的实施例的范围，则承载计算机程序的存储介质和传输介质构成本发明的方面。计算机程序可具有一个或多个程序指令或程序代码，其在被计算机执行时实现本发明的实施例。如本文所使用的术语“程序”可以是针对在计算机系统上执行而设计的指令序列，并且可包括子例程、函数、程序、模块、对象方法、对象实现、可执行应用程序、小应用程序、小服务程序、源代码、目标代码、共享库、动态链接库和/或针对在计算机系统上执行而设计的其它指令序列。存储介质可以是磁盘（诸如硬驱或软盘）、光盘（诸如CD-ROM、DVD-ROM或BluRay盘）或存储器（诸如ROM、RAM、EEPROM、EPROM、闪存或便携式/可移除存储器设备）等。传输介质可以是通信信号、数据广播、两个或更多计算机之间的通信链路等。

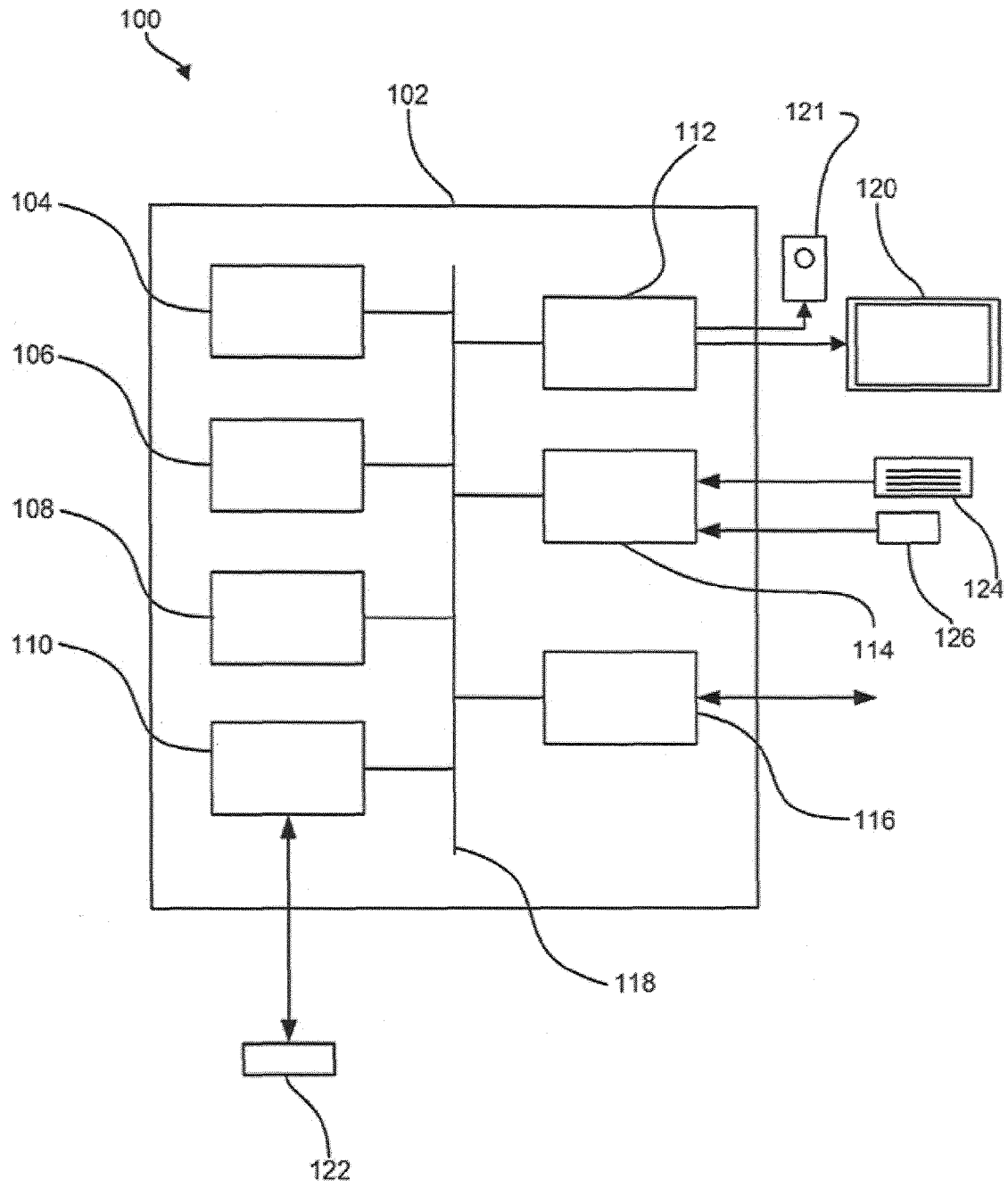


图 1

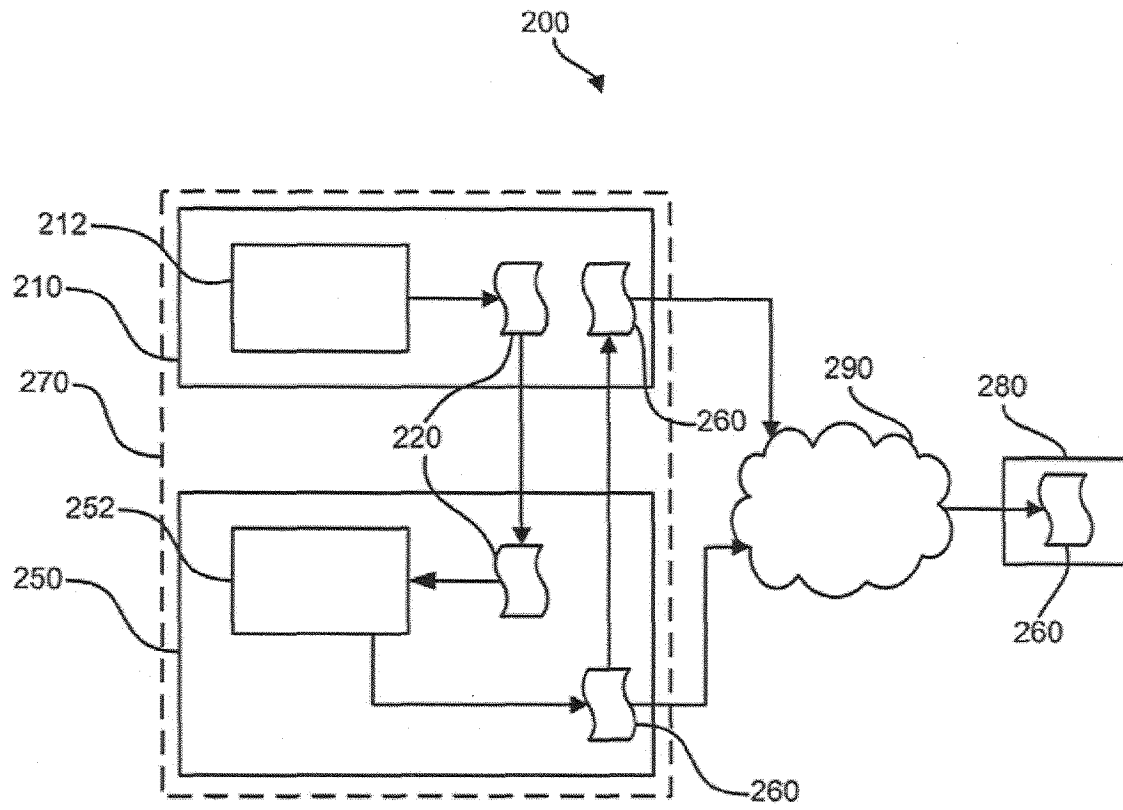


图 2

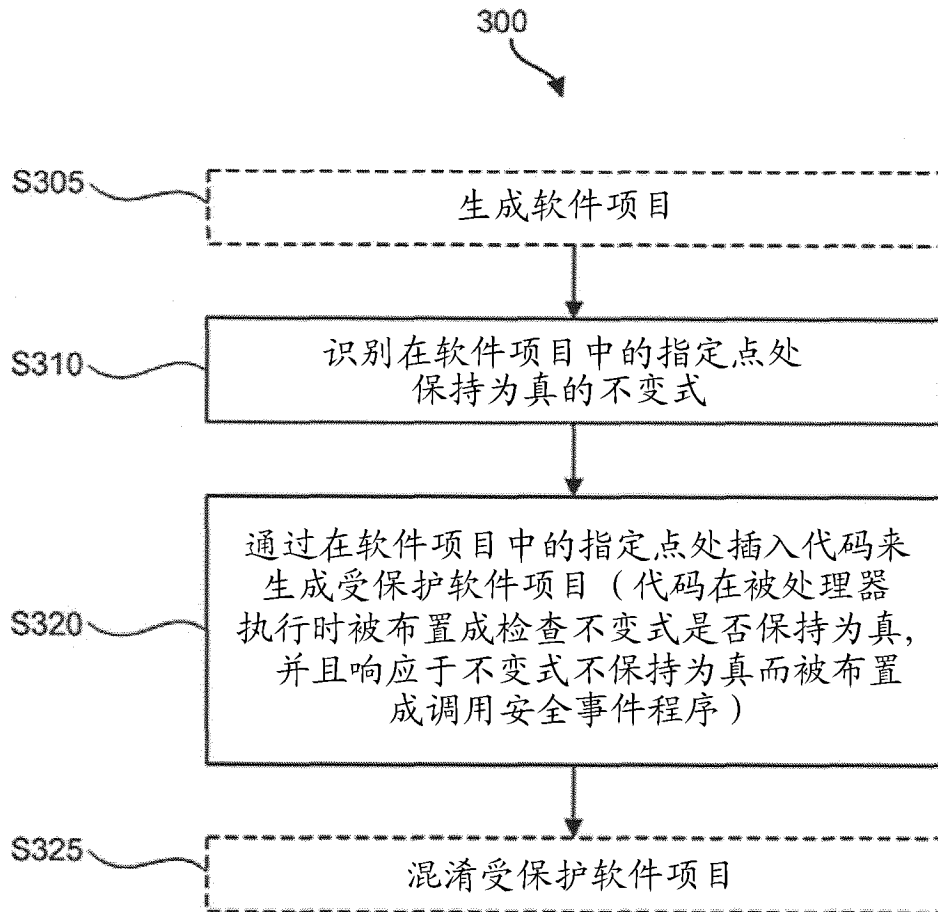


图 3

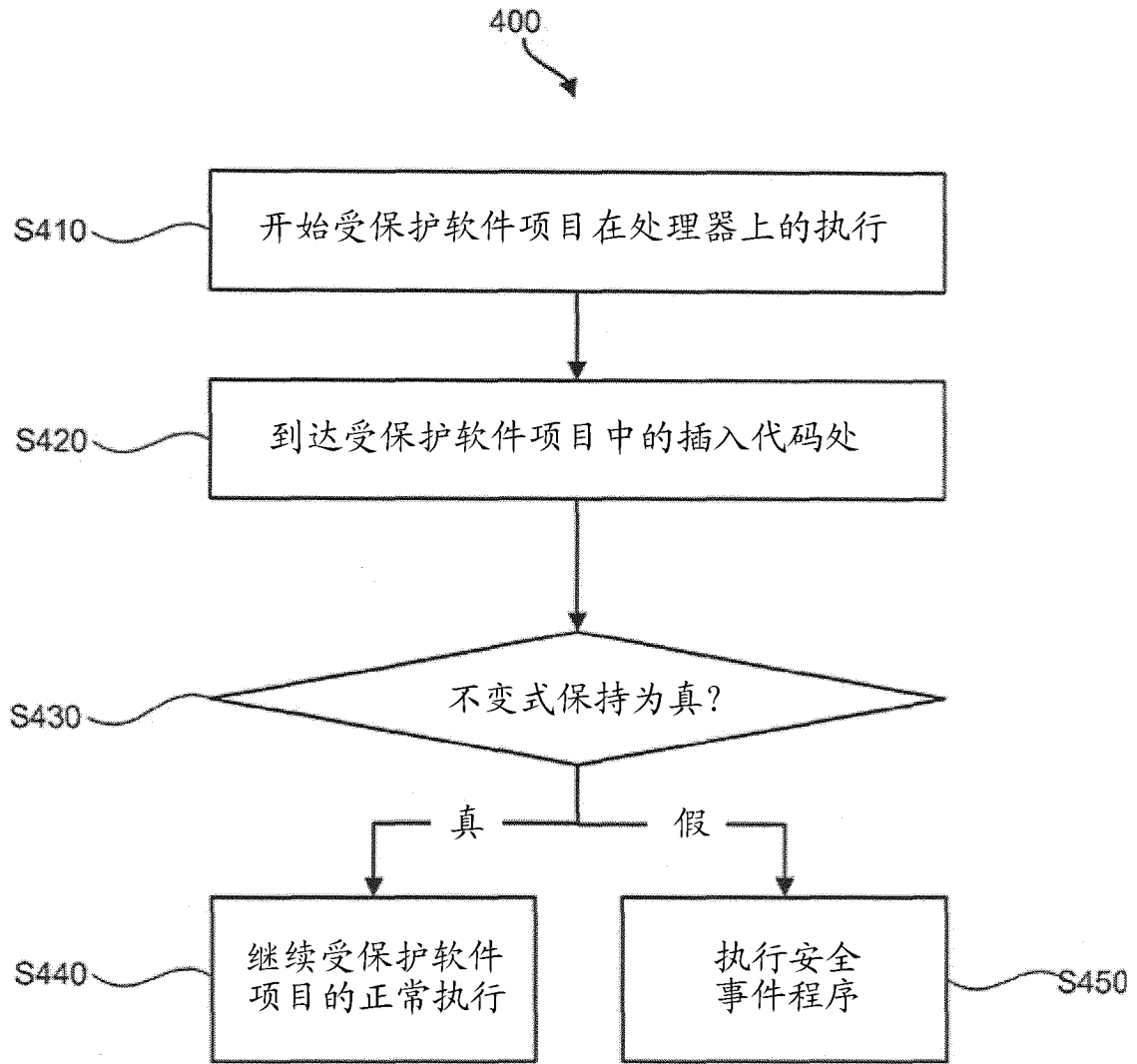


图 4