(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 112541178 A
(43)申请公布日 2021.03.23

(21)申请号 202010973287.1

(22)申请日 2020.09.16

(30)优先权数据
16/578,455 2019.09.23 US

(71)申请人 株式会社电装
地址 日本爱知县

(72)发明人 阿米·卡沙尼
戈帕拉克里希南·耶尔

(74)专利代理机构 北京集佳知识产权代理有限公司 11227
代理人 王萍 候艳超

(51)Int.Cl.
G06F 21/57(2013.01)

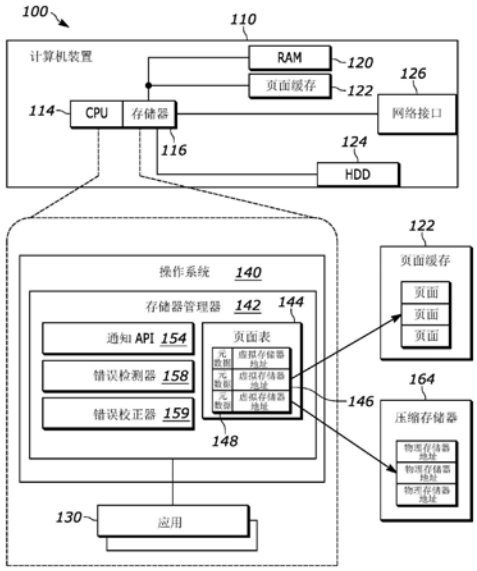
权利要求书2页 说明书8页 附图3页

(54)发明名称

控制流完整性实施的设备和方法

(57)摘要

涉及控制流完整性实施的设备和方法。一种计算机装置(110)包括存储器(116)。该计算机装置还包括至少一个处理器(114),所述至少一个处理器(114)被配置成执行过程并且针对该过程管理存储器。处理器还被配置成:执行与应用相关联的一个或更多个程序指令;到达针对所述一个或更多个程序指令的控制流转移;响应于未能满足目标控制流,展开与所述一个或更多个程序指令相关联的调用堆栈;识别违规函数调用;以及重写违规函数调用。经重写的函数调用包括存储器操作边界检查。



CN 112541178 A

1.一种计算机装置,包括:

存储器(50、116);以及

至少一个处理器(48、114),其被配置成执行过程并且针对所述过程管理所述存储器,其中,所述至少一个处理器被配置成:

执行与应用相关联的一个或更多个程序指令;

到达针对所述一个或更多个程序指令的控制流转移;

响应于未能满足目标控制流,展开与所述一个或更多个程序指令相关联的调用堆栈;

识别违规函数调用;以及

重写所述违规函数调用,其中,所述经重写的函数调用包括存储器操作边界检查。

2.根据权利要求1所述的计算机装置,其中,所述处理器还被配置成:响应于未能满足所述目标控制流,启用针对所述存储器的写入权限。

3.根据权利要求2所述的计算机装置,其中,所述处理器还被配置成:响应于未能满足所述目标控制流,禁用针对所述存储器的写入权限。

4.根据权利要求1所述的计算机装置,其中,所述过程被配置成发生在运行时间的操作期间。

5.根据权利要求1所述的计算机装置,其中,所述处理器还被配置成确定所述违规函数调用的数据大小限制。

6.根据权利要求5所述的计算机装置,其中,所述存储器操作边界检查与所述数据大小限制相关联。

7.一种针对在处理器上执行的过程来管理存储器的方法,包括:

执行存储在存储器中并且与应用相关联的一个或更多个程序指令;

到达针对所述一个或更多个程序指令的控制流转移;

响应于未能满足目标控制流,展开与所述一个或更多个程序指令相关联的调用堆栈;

识别违规函数调用;以及

重写所述违规函数调用,其中,所述经重写的函数调用包括存储器操作边界检查。

8.根据权利要求7所述的方法,其中,所述方法还包括以下步骤:将程序调用堆栈恢复至先前状态。

9.根据权利要求7所述的方法,其中,所述方法还包括以下步骤:响应于未能满足所述目标控制流,启用针对所述存储器的写入权限。

10.根据权利要求9所述的方法,其中,所述方法还包括以下步骤:响应于未能满足所述目标控制流,禁用针对所述存储器的写入权限。

11.根据权利要求7所述的方法,其中,所述方法发生在运行时间的操作期间。

12.根据权利要求7所述的方法,其中,所述处理器还被配置成确定所述违规函数调用的数据大小限制。

13.根据权利要求12所述的方法,其中,所述存储器操作边界检查与所述数据大小限制相关联。

14.一种在交通工具中的计算机装置,包括:

存储器(50、116);以及

至少一个处理器(48、114),其被配置成执行过程并且针对所述过程管理所述存储器,

其中,所述至少一个处理器被配置成:

执行与应用相关联的一个或更多个程序指令;

到达针对所述一个或更多个程序指令的控制流转移;

响应于未能满足目标控制流,展开与所述一个或更多个程序指令相关联的调用堆栈;

识别违规函数调用以及与所述违规函数调用相关联的存储器中的寄存器的数据大小限制;以及

重写所述违规函数调用,其中,所述经重写的函数调用包括存储器操作边界检查。

15.根据权利要求14所述的计算机装置,其中,所述处理器被配置成:将程序调用堆栈恢复至先前状态。

16.根据权利要求14所述的计算机装置,其中,所述处理器还被配置成:响应于未能满足所述目标控制流,启用针对所述存储器的写入权限。

17.根据权利要求16所述的计算机装置,其中,所述处理器还被配置成:响应于未能满足所述目标控制流,禁用针对所述存储器的写入权限。

18.根据权利要求14所述的计算机装置,其中,所述过程被配置成发生在运行时间的操作期间。

19.根据权利要求14所述的计算机装置,其中,所述处理器还被配置成:确定所述违规函数调用的数据大小限制。

20.根据权利要求19所述的计算机装置,其中,所述存储器操作边界检查与所述数据大小限制相关联。

控制流完整性实施的设备和方法

技术领域

[0001] 本公开内容涉及与软件相关的控制流完整性。

背景技术

[0002] 攻击者利用大多数漏洞改变应用的正常控制流,以使用所利用的应用的特权执行任意恶意活动。控制流完整性(control flow graph,CFI)是一种安全机制,其不允许对编译后的二进制代码的原始控制流图进行更改,从而使得执行这样的攻击显著地变得更加困难。

发明内容

[0003] 根据一个实施方式,一种计算机装置包括存储器,该计算机装置还包括至少一个处理器,所述至少一个处理器被配置成执行过程并且针对该过程管理存储器。处理器还被配置成:执行与应用相关联的一个或更多个程序指令;到达(reach)针对所述一个或更多个程序指令的控制流转移;响应于未能满足目标控制流;展开(unwind)与所述一个或更多个程序指令相关联的调用堆栈;识别违规函数调用;以及重写违规函数调用。经重写的函数调用包括存储器操作边界检查(memory operation boundary check)。

[0004] 根据第二实施方式,一种针对在处理器上执行的过程来管理存储器的方法,包括:执行与应用相关联的一个或更多个程序指令;到达针对所述一个或更多个程序指令的控制流转移;响应于未能满足目标控制流,展开与所述一个或更多个程序指令相关联的调用堆栈;识别违规函数调用;以及重写违规函数调用,其中,经重写的函数调用包括存储器操作边界检查。

[0005] 根据第三实施方式,一种在交通工具中的计算机装置包括存储器和至少一个处理器,所述至少一个处理器被配置成执行过程并且针对该过程管理存储器。处理器被配置成执行与应用相关联的一个或更多个程序指令。处理器还被配置成:到达针对所述一个或更多个程序指令的控制流转移;响应于未能满足目标控制流,展开与所述一个或更多个程序指令相关联的调用堆栈;识别违规函数调用以及与违规函数调用相关联的存储器中的寄存器的数据大小限制;以及重写违规函数调用,经重写的函数调用包括存储器操作边界检查。

附图说明

[0006] 图1是根据本公开内容的实现方式的用于检测分配给过程的存储器的一部分中的错误的示例计算机系统的图;

[0007] 图2是根据本公开内容的实现方式的示例计算机装置的示意性框图;以及

[0008] 图3是根据本公开内容的实现方式的控制流完整性(CFI)实施的示例方法的流程图300。

具体实施方式

[0009] 本文描述了本公开内容的实施方式。然而，应当理解，所公开的实施方式仅仅是示例，并且其他实施方式可以采取各种不同的和可替选的形式。附图不一定按比例绘制；一些特征可以被放大或缩小以示出特定部件的细节。因此，本文所公开的具体结构性和功能性细节不应被解释为是限制性的，而仅仅是作为用于教示本领域技术人员以各种方式采用实施方式的代表性基础。如本领域的普通技术人员将理解的，参照任一附图说明和描述的各种特征可以与在一个或更多个其他附图中说明的特征组合以产生未明确说明或描述的实施方式。所说明的特征的组合针对典型应用提供代表性的实施方式。然而，针对特定的应用或实现可能需要与本公开内容的教示一致的特征的各种组合和修改。

[0010] 例如，存储器损坏(memory corruption)可能出现在利用以低级语言例如C或C++编写的软件的计算系统中。这是因为这样的语言在对存储器的读或写的数据验证检查中默认不编译。例如，编译后的C代码可能不包括用于阻止数据被写入分配给数据缓冲区的存储器的区域之外的验证。这样的越界存取可以被称为缓冲区溢出(buffer overflow)或缓冲区溢位(buffer overrun)。

[0011] 当过程(a process)执行编译后的代码时，数据的堆栈作为存储器的块被分配给该过程。当通过正在执行的代码调用函数时，新的帧被推入该堆栈。新的帧包括被传送至函数的参数、用于函数的本地数据变量以及一旦函数完成就继续执行所在的返回地址。当函数完成时，帧从堆栈中弹出，并且返回地址被置于程序计数器中以恢复(resume)程序流。返回地址通常是紧接在函数被调用的位置之后的编译后的代码的指令。

[0012] 堆栈缓冲区溢出是可以用于更改程序流的特定类型的缓冲区溢出。例如，攻击者可以设计堆栈上的局部变量的缓冲区溢出，形成该缓冲区溢出以使返回地址被覆写而跳至恶意代码。这是攻击者可以如何利用某些编程语言中固有的数据验证中的遗漏来劫持正在计算系统上执行的程序的控制流的一个示例。

[0013] 控制流完整性(CFI)技术可以用于检测某些种类的存储器损坏，但是其响应能力仍然有限。CFI首先通过跟踪代码或程序中的所有可能的执行路径以构建控制流图(control flow graph，CFG)并且然后通过在运行时间实施程序流遵守控制流图来工作。CFI实施检查(CFI enforcement checks)可以以阻止代码的控制流偏离控制流图的安全形式实现在代码中，使得显著地更加难以执行改变控制流的攻击以及使用由所利用的应用产生的特权执行任意恶意活动(例如，恶意软件、病毒、特洛伊木马、间谍软件等)。因此，CFI技术可以监视和保存计算机环境内的控制转移和/或程序流。CFI技术可以使用引用监视器(例如，内嵌式或外置式)以及被传送经过引用监视器并且与基础事实(ground truth)CFI图进行比较的代码的函数转换来确定异常或偏离行为。CFI技术可以识别由无辜或自然原因引起的故障，包括环境辐射(例如，电场和磁场等)、随机噪声、信号错误或部件故障。更重要的是，CFI技术可以检测由故意行为引起的故障，例如由用户提供的对恶意程序输入的操作引起的故障。典型的CFI违例处理导致执行的停止，并且以新状态重新执行程序。然而，这样的处理机制不能消除对漏洞的敌对性(adversarial)访问，并且它还通过允许持续的利用降低了系统的可用性。

[0014] 在改进的CFI技术中，程序仍然可以配置有初始CFI实施指令。然而，在检测到CFI违例时，代替停止或重新开始执行，程序调用堆栈可以替代地从故障点展开以识别引起违

例的违规函数。然后，可以将该违规函数修改(即，重写)成包括对在违规函数中溢位的目标缓冲区执行边界检查的指令。通过重写违规函数，可以使用先前程序调用堆栈状态从调用违规函数之前的点恢复执行。显著地，由于经重写的函数现在包括边界检查，因此将阻止导致初始违例的畸形数据引起另一违例。因此，存储器损坏在恢复的执行中被阻止，并且攻击者不能完成溢出漏洞。因此，不是简单地停止执行以及突然地停止应用的执行，而是可以修改和恢复应用，从而从攻击中恢复，同时还阻止在该应用中在该点处的未来错误或问题。

[0015] 现在参照图1，示例计算机系统100包括计算机装置110。计算机装置110可以例如是任何移动或固定计算机装置，包括但不限于台式计算机或膝上型计算机或平板计算机、蜂窝式电话、游戏装置、混合现实装置或虚拟现实装置、音乐装置、电视、导航系统、摄像机、个人数字助理(PDA)、手持式装置、电子控制单元、具有与一个或更多个其他装置的有线和/或无线连接能力的任何其他计算机装置、或者能够生成视频输出信号的任何其他类型的计算机化装置。

[0016] 计算机装置110可以包括执行存储在存储器116中的指令的中央处理单元(CPU)114。例如，CPU 114可以执行操作系统140以及一个或更多个应用130。操作系统140和应用130可以各自与下述一个或更多个过程相关联：所述一个或更多个过程可以被分配有过程标识符并且被分配有存储器116的一部分。

[0017] 存储器116可以被配置用于存储数据和/或计算机可执行指令。计算机可执行指令可以定义操作系统140和/或应用130以及/或者与操作系统140和/或应用130相关联。CPU 114可以执行操作系统140和/或应用130。存储器116可以表示计算机装置110可访问的一个或更多个硬件存储器装置。存储器116的示例可以包括但不限于可以由计算机使用的类型的存储器，例如，随机存取存储器(RAM)、只读存储器(ROM)、磁带、磁盘、光盘、易失性存储器、非易失性存储器以及以上存储器的任何组合。存储器116可以存储正在由CPU 114执行的应用的本地版本。在示出的示例中，存储器116包括RAM 120、页面缓存122、硬盘驱动器124以及网络接口126。RAM 120可以是硬件部件例如一个或更多个双列直插式存储器模块(DIMM)。页面缓存122可以是RAM 120的下述部分：该部分用于存储源于辅助存储装置如硬盘驱动器124的页面。硬盘驱动器124可以表示任何辅助存储装置。硬盘驱动器124相比于RAM 120可以具有更大的容量但是更慢的存取时间。网络接口126也可以用作辅助存储装置例如用作网络驱动器。

[0018] CPU 114可以包括用于执行指令的一个或更多个处理器。CPU 114的示例可以包括但不限于本文所描述的专门编程的任何处理器，处理器包括控制器、微控制器、专用集成电路(ASIC)、现场可编程门阵列(FPGA)、图形处理单元(GPU)、片上系统(SoC)或者其他可编程逻辑机或状态机。CPU 114可以包括其他处理部件，例如算术逻辑单元(ALU)、寄存器以及控制单元。CPU 114可以包括多个核并且可能能够使用多个核执行多个线程来同时处理不同的指令组和/或数据组。

[0019] 操作系统140可以包括存储在存储器116中并且可以由CPU 114执行的指令(例如应用130)。操作系统140可以包括用于将存储器分配给过程的存储器管理器142。例如，存储器管理器142可以实现虚拟存储器系统。存储器116可以包括有限量的RAM 120。由CPU 114执行的处理可以请求比RAM 120的可用量更多的存储器。然而，所请求的存储器的大部分可能在相当量的时间内保持空闲。存储器管理器142可以通过将虚拟存储器地址146分配给过

程来使用虚拟存储器以满足对存储器的请求。然后，虚拟存储器地址146可以分别与RAM 120或页面中的物理存储器地址相关联，所述物理存储器地址可以存储在其他逻辑部件和物理部件例如压缩存储器164或硬盘驱动器124中。在实现方式中，虚拟存储器可以包括存储用于每个虚拟存储器地址146的存储器内容(例如指针)的位置的页面表144。在实现方式中，页面表144还可以存储用于检测和校正存储器错误或损坏的元数据148。例如，元数据148的组可以与页面表144中的每个虚拟存储器地址146相关联。

[0020] 通知应用编程接口(API)154或者其他类型的软件可以允许过程与存储器管理器142通信以针对过程配置存储器管理的某些特征。在实现方式中，存储器管理器142可以向过程或函数提供关于损坏的存储器的通知。例如，当损坏的存储器被检测到时，存储器管理器142可以生成通知。例如，存储器管理器142可以抛出指示损坏的存储器已经被检测到或者CFI违例已经出现的异常。过程可以指示过程是否正在处理异常。如果存储器管理器142未接收到过程正在处理异常的指示，则存储器管理器142可以确定如何处理异常(例如，尝试恢复、终止过程、使系统崩溃等)。作为通知的另一示例，当损坏的数据被校正时，存储器管理器142可以生成通知。类似于检测损坏的存储器，一些过程可以以特定方式处理经校正的存储器。例如，过程可能能够重新生成数据，而不是相信存储器管理器142已经成功地校正了损坏的存储器。在其他情况下，当损坏的存储器被校正时，过程可以选择不接收通知。

[0021] 错误检测器158可以是用于评估函数调用是否已经变得损坏的软件或硬件。例如，错误检测器158可以通过检查缓冲区存取是否超过存储器116中的缓冲区容量来识别缓冲区错误。当检测到错误时，错误检测器158可以向通知API 154提供指示。错误检测器158可以被配置成当通过过程请求函数调用时对函数调用进行评估。因此，错误检测器158可以确保过程正在接收经验证的数据。另外地或可替选地，错误检测器158可以周期性地评估函数调用以确定函数调用是否已经变得损坏。例如，错误检测器158可以使用空闲过程来检查错误，或者可以选择已经处于存储器中相对长时间的函数。例如，针对正在后台运行的应用，错误检测器158可以通过应用的类型来识别应用，并且确定该应用的正常运行时间(例如，运行时间阈值)。如果超过正常运行时间，则错误检测器158可以确定应用中可能存在错误。例如，如果预期某个应用仅运行一分钟，并且该应用运行了超过一分钟的时间，则错误检测器158可以确定函数已经变得损坏。在其他示例中，可以利用看门狗(watchdog)确定当被监视时应用是否未接收到期望的响应或“心跳”。

[0022] 当由错误检测器158检测到损坏的函数调用时，错误校正器159可以尝试对损坏的函数调用进行校正。错误校正器159可以确定性地修改函数调用的代码。例如，错误校正器159可以顺序地修改代码以测试错误。例如，错误校正器159可以分析每一行代码(或一组代码行)以修改并且相应地测试每个修改后的代码行的错误。然后，错误校正器159可以使用错误检测器158评估修改后的函数调用是否与原始函数调用匹配。

[0023] 现在参照图2，示出了根据实现方式的示例计算机装置110，包括与图1相比较的另外的部件细节。在一个示例中，计算机装置110可以包括用于执行与本文描述的一个或更多个部件和功能相关联的处理功能的处理器48。处理器48可以包括单组或多组处理器或多核处理器。此外，处理器48可以被实现为集成处理系统和/或分布式处理系统。在实现方式中，例如，处理器48可以包括CPU 114。

[0024] 在示例中，计算机装置110可以包括用于存储指令的存储器50，所述指令可以由用

于执行本文所描述的功能的处理器48执行。在实现方式中，例如，存储器50可以包括存储器116。CPU 114还可以包括用于执行所描述的功能的专用指令(例如，通过调用这些专用指令来检查函数调用以检查控制流转移的有效性/正确性)。这些检查还可以由作为错误检测器158的一部分而被包括的引用监视器以及被编译到操作系统140或应用130中的仪器来执行。

[0025] 此外，计算机装置110可以包括通信部件52，该通信部件52提供利用如本文所描述的硬件、软件以及服务来建立和保持与一方或更多方的通信。通信部件52可以执行计算机装置110上的部件之间的通信，以及计算机装置110与外部装置(例如，跨通信网络定位的装置和/或串行或本地连接至计算机装置110的装置)之间的通信。例如，通信部件52可以包括一个或更多个总线，并且还可以包括能够操作用于与外部装置交互的分别与发送器和接收器相关联的发送链部件和接收链部件。

[0026] 另外，计算机装置110可以包括数据存储装置54，数据存储装置54可以是提供关于本文所描述的实现方式采用的信息、数据库和程序的大容量存储的硬件和/或软件的任何适当的组合。例如，数据存储装置54可以是用于操作系统140和/或应用130的数据储存库。数据存储装置可以包括存储器116。

[0027] 计算机装置110还可以包括用户接口部件56，用户接口部件56能够操作用于接收来自计算机装置110的用户的输入并且还能够操作用于生成用于向用户呈现的输出。用户接口部件56可以包括一个或更多个输入装置，包括但不限于触觉输入、键盘、数字键盘、鼠标、触敏显示器、数字化仪、导航键、功能键、麦克风、语音识别部件、能够接收来自用户的输入的任何其他机件或以上的任何组合。此外，用户接口部件56可以包括一个或更多个输出装置，包括但不限于显示器、扬声器、触觉反馈机件、打印机、能够向用户呈现输出的任何其他机件或以上的任何组合。

[0028] 在实现方式中，用户接口部件56可以发送和/或接收与操作系统140和/或应用130的操作相对应的消息。此外，处理器48可以执行操作系统140和/或应用130，并且存储器50或数据存储装置54可以存储操作系统140和/或应用130。

[0029] 计算机装置110可以位于交通工具内部。例如，计算机装置110可以是包括导航系统、音频主机单元、交通工具安全系统以及其他交通工具部件的交通工具计算机系统的一部分。在这样的示例中，计算机装置110可以利用交通工具通信网络如交通工具总线(例如，控制器局域网络(CAN)总线等)与其他交通工具部件交互。

[0030] 图3是根据本实施方式的实现方式的CFI实施的示例方法的流程图300。在示例中，可以通过计算机装置110将应用130或其他计算机代码指令加载进存储器116中以用于由CPU 114执行来执行示例方法。

[0031] 在操作301处，计算机装置110执行程序指令。程序指令可以是目标为由计算机装置110的CPU 114执行的任何类型的编译后的代码。在说明性示例中，程序指令包括由对于存储器的读写缺乏数据验证检查的低级语言(例如，C或C++)来编译的指令。计算机装置110可以在计算机装置110启动时或者在加载新应用时执行任何给定的程序。计算机装置110可以基于代码或指令自动地执行代码。在另一示例中，可以基于用户手动地运行代码来执行代码。这样的示例包括双击程序、键入指令以执行代码、触碰触摸界面上的应用图标、基于任何输入装置激活图标、激活交通工具功能等。还可以利用其他输入接口执行程序。

[0032] 在操作303处，正在被执行的代码的程序流到达控制流转移。控制流转移可以是将对程序的控制从一个代码位置转移至另一个代码位置的程序指令的机器代码指令。这样的指令的示例包括：将程序流转移至子例程位置(并且还将调用后的指令地址保存在堆栈上以供返回指令稍后使用)的调用指令；在子例程的执行完成时允许执行恢复回到堆栈上的存储位置的返回指令；基于满足一个或更多个测试条件来可选地将控制转移至另一指令位置的条件跳转指令；以及不管条件如何均无条件地将控制转移至目标位置的无条件跳转指令等。

[0033] 在操作305处，系统确定控制流转移的目的地指令是否是有效的目标代码指令。如果控制流转移正在尝试(或引起)与对程序的预期一致的程序流的转移，则可以认为目标代码指令有效。故障可以由无辜或自然原因引起，包括环境辐射、随机噪声、信号错误，或者由人为原因引起。或者，故障可以由故意行为——例如，由用户提供的对恶意程序输入的操作——引起。在一个实施方式中，为了识别目标是否有效，系统可以将钩子(hook)包括到要经历CFI实施的各种函数调用中。系统还可以存储CFG模型，CFG模型表示根据正在计算机装置110上运行的代码或程序指令得到的正常程序流。错误检测器158可以利用钩子截获函数调用并且执行对由于执行待执行的(pending)控制流转移而引起的程序流与CFG模型的正常程序流的比较。然而，如果目标无效，则控制传递至操作307。在一个示例中，如果目标与CFG模型一致，则目标被认为是有效的，并且控制返回至操作301处以继续进行控制流转移指令的执行。例如，在程序执行期间，无论何时机器代码指令转移控制，它都以由提前创建的CFG确定的有效目的地为目标。如果程序流与CFG模型不匹配，则错误检测器158或系统可以基于控制流转移的到达来确定是否出现了故障。

[0034] 在操作307处，系统展开程序调用堆栈。在示例中，系统可以基于以下假设从程序调用堆栈弹出当前堆栈帧：在当前违规函数被调用之前程序执行处于一致的稳定点，并且在执行中堆栈在该点处未损坏。可以对程序调用堆栈的违规函数执行机器代码反汇编，并且可以识别违规函数内的存储器操作符/操作数。如下文更详细地论述的，可以重写违规的函数代码以消除存储器损坏漏洞(即，缓冲区溢出)，而不是使违规的函数代码保持不变并且冒着重复出现故障状况的风险。例如，可以将应用或程序从机器语言反汇编成汇编语言。可以利用错误校正器159将机器语言反汇编成汇编语言。然后，错误校正器159可以分析代码行(例如，单独地或将它们集体分组地)以确定针对代码行的存储器操作和数据大小限制。

[0035] 在操作309处，系统分析违规函数代码以确定数据大小限制。例如，系统(例如，错误检测器158)可以对指令执行代码分析以识别存储器存取操作和操作数。系统(例如，错误检测器158)还可以对指令执行代码分析以识别变量的实例化，因为实例化可以指示正在被定义的变量的界限。作为一种可能性，可以通过对程序指令的静态、动态或符号分析来提取数据大小限制。在示例中，系统可以模拟函数的程序执行以识别导致存储器损坏的缓冲区。在另一示例中，系统可以在程序指令中搜索缓冲区初始化代码，该缓冲区初始化代码包括数据大小限制作为参数值、寄存器值或立即值。在一些实施方式中，为了执行代码分析，系统可以将违规函数的机器指令代码反汇编成更适用于高级分析的汇编表示或其他表示。

[0036] 在另一示例中，可以对违规函数以及违规函数内的识别的存储器操作符/操作数执行机器代码反汇编。将操作数跟踪到其实例化，从而确定其数据大小限制。边界检查由基

于那些确定的数据大小限制来实现大小比较的机器指令构成。通过移动现有指令或以其他方式重新定位现有指令来创建用于边界检查的空间以将这些机器指令与现有指令内联添加。重写代码的原因是为了消除存储器损坏漏洞(即,缓冲区溢出),而不是使代码保持不变并且冒着重复出现故障状况的风险。因此,可以在敌对性操作环境中提高总体系统可靠性和可用性。在一个示例中,可以利用错误校正器159重写代码以消除存储器损坏。

[0037] 在操作311处,系统启用对存储要被修改的代码指令的存储器的写入权限。在许多实现方式中,包括可执行代码的存储器段由存储器管理器142标记为只读以允许代码被读取但是阻止代码被更改。这通常是有利的,因为一旦将程序指令加载至存储器中,写访问限制就会阻止意外或故意篡改或更改程序指令。然而,对代码指令的写访问的这种限制可能会阻碍通过CFI实施方法识别的运行时间程序改进的实现。在一个示例中,分配的权限模型可以在计算装置操作期间实施写访问限制并且不允许修改权限。系统还可以指示存储器管理器142以既可读取又可写入的权限临时标记存储代码的存储器,从而允许在识别出违规函数的场景下重写代码。

[0038] 在操作313处,系统根据数据大小限制重写函数以包括存储器操作边界检查。经重写的函数包括确保对函数的数据缓冲区的写入操作被限制于缓冲区的大小的边界检查。系统可以跟踪故障以确定数据缓冲区的大小。系统可以通过分析与经重写的函数调用相关联的存储器的数据大小限制来确保针对重写的函数调用减少了数据大小限制。为了帮助生成边界检查代码指令,系统(例如,错误校正器159)可以利用具有数据大小限制的代码模板和可以插入的存储器操作数位置来生成经重写的代码。然后,系统可以将经重写的代码注入到现有的机器代码中以阻止未来的CFI错误。例如,可以通过移动现有代码或以其他方式重新定位现有代码来创建用于新边界检查指令的空间以将新指令与现有指令内联添加。通过重写代码,可以消除存储器损坏漏洞(即,缓冲区溢出)。这避免了重复故障状况的可能性。因此,可以提高总体系统可靠性和可用性。可以从违规函数调用跟踪数据大小限制到初始化。例如,可以通过对程序指令的静态、动态或符号分析来提取数据大小限制。系统(例如,错误检测器158)可以确定违规函数调用并且反汇编程序指令或者模拟程序执行以识别导致存储器损坏的缓冲区。系统可以在存储器或程序指令中搜索包括数据大小限制作为参数值、寄存器值或立即值的缓冲区初始化代码。恢复的数据大小限制和存储器操作数位置被插入到代码模板中以生成修改后的代码。在另一示例中,系统(例如,错误校正器159)可以测试修改后的代码以验证修改后的指令相比于原始指令的功能等效性。系统可以将各种软件测试(例如,单元测试、回归测试、集成测试、形式化验证测试、硬件在环测试、编码规则测试等)应用于修改后的程序,在模拟环境下以良性或畸形的输入重新执行修改后的程序等,以便确定修改后的代码中的错误的存在。响应于任何错误,系统可以改变修改(例如,使用随机突变、遗传算法等)并且重复测试直到满足通过标准。系统可以确定违规函数调用并且反汇编程序指令或模拟程序执行以识别导致存储器损坏的缓冲区。系统可以在存储器或程序指令中搜索缓冲区初始化代码,该缓冲区初始化代码包括数据大小限制作为参数值、寄存器值或立即值。可以将恢复的数据大小限制和存储器操作数位置插入代码模板中以生成修改后的代码。系统可以在运行时间操作期间重写函数调用,而不是停止操作。

[0039] 在操作315处,系统禁用对存储代码指令的存储器的写入权限。这恢复了存储器管理器142对代码的写访问的阻止,从而再次阻止了加载至存储器中的代码被意外或恶意用

户写入。照此，禁用写入权限可以阻止由未经授权的用户修改代码的能力。当禁用了权限时，阻止了修改存储器或存储装置中的条目的能力，所述条目包括创建文件、删除文件、重命名文件等。

[0040] 在操作317处，系统将程序调用堆栈恢复至先前状态。例如，程序堆栈、程序计数器和其他CPU寄存器以及其他程序状态可以被重置为与执行违规函数之前的程序指令的执行一致的状态。在操作317之后，控制传递至操作301以恢复代码的执行。通过重写违规函数，执行可以使用先前的程序调用堆栈状态从调用违规函数之前的点恢复。由于经重写的函数现在包括边界检查，因此将阻止出现的违例引起另一违例。因此，在恢复的执行中阻止了存储器损坏。因此，可以修改应用并且使应用从恢复的状态恢复，而不是简单地停止执行以及突然地停止应用的执行，从而从攻击中恢复，同时还阻止在该应用中在该点处的未来错误或问题。照此，程序可以利用修改后的函数恢复执行，所述修改后的函数是在根据数据大小限制重写函数调用以包括存储器操作边界检查之后产生的。然后，程序可以从允许边界检查的开始处恢复执行。可以在恢复的状态下发生执行。

[0041] 尽管以上描述了示例性实施方式，但是并非意指这些实施方式描述了权利要求书所涵盖的所有可能形式。说明书中使用的词语是描述性而非限制性的词语，并且应当理解，在不脱离本公开内容的精神和范围的情况下可以做出各种改变。如先前所描述的，可以组合各种实施方式的特征以形成本发明的可能未明确描述或说明的另外的实施方式。虽然各种实施方式已经被描述为提供优点或者就一个或更多个期望特性而言相对于其他实施方式或现有技术实现是优选的，但是本领域的普通技术人员认识到，可以取决于特定应用和实现方式对一个或更多个特征或特性进行折中以实现期望的总体系统属性。这些属性可以包括但不限于成本、强度、耐用性、生命周期成本、可销售性、外观、包装、大小、适用性、重量、可制造性、易组装性等。照此，在任何实施方式被描述为就一个或多个特性方面而言相比于其他实施方式或现有技术实现方式不是太可取的程度上，这些实施方式不在本公开内容的范围之外，并且针对特定应用可以是可取的。

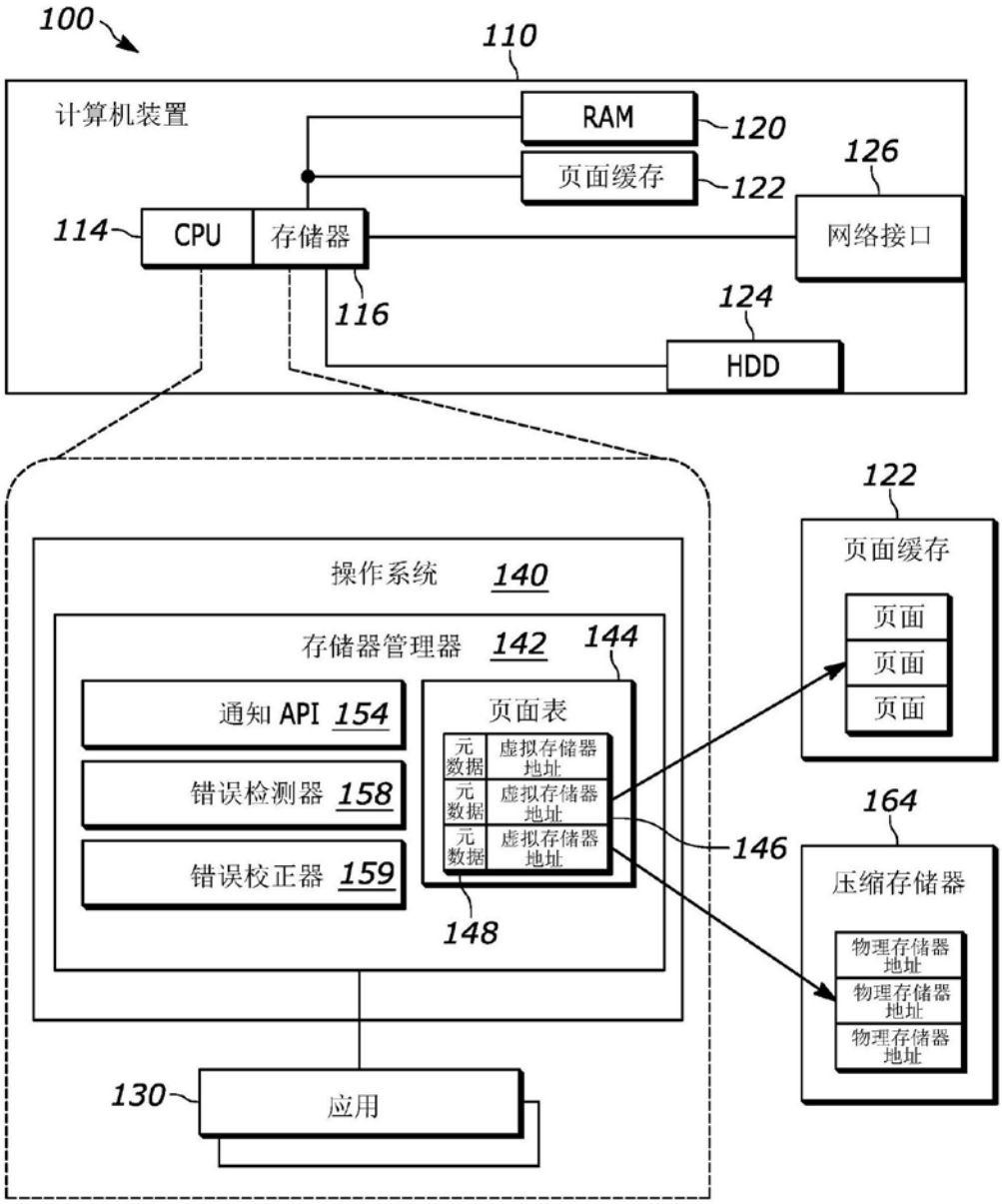

100
110
计算机装置
RAM
120
页面缓存
122
126
网络接口
114
CPU
存储器
116
124
HDD
操作系统 140
存储器管理器 142
144
通知 API 154
错误检测器 158
错误校正器 159
页面表
元数据
虚拟存储器地址
148
146
122
页面缓存
页面
页面
页面
164
压缩存储器
物理存储器地址
物理存储器地址
物理存储器地址
130
应用

图1

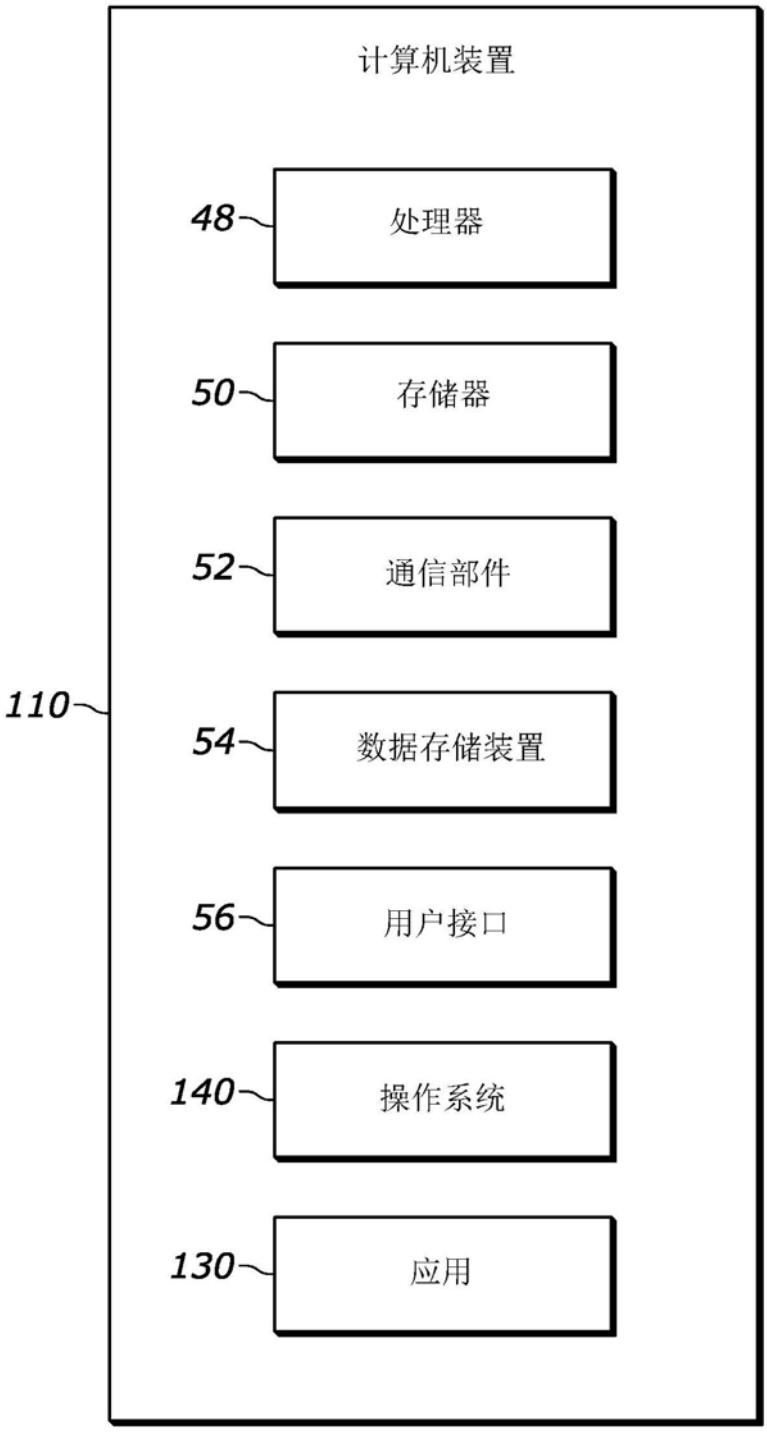

计算机装置
48
处理器
50
存储器
52
通信部件
110
54
数据存储装置
56
用户接口
140
操作系统
130
应用

图2

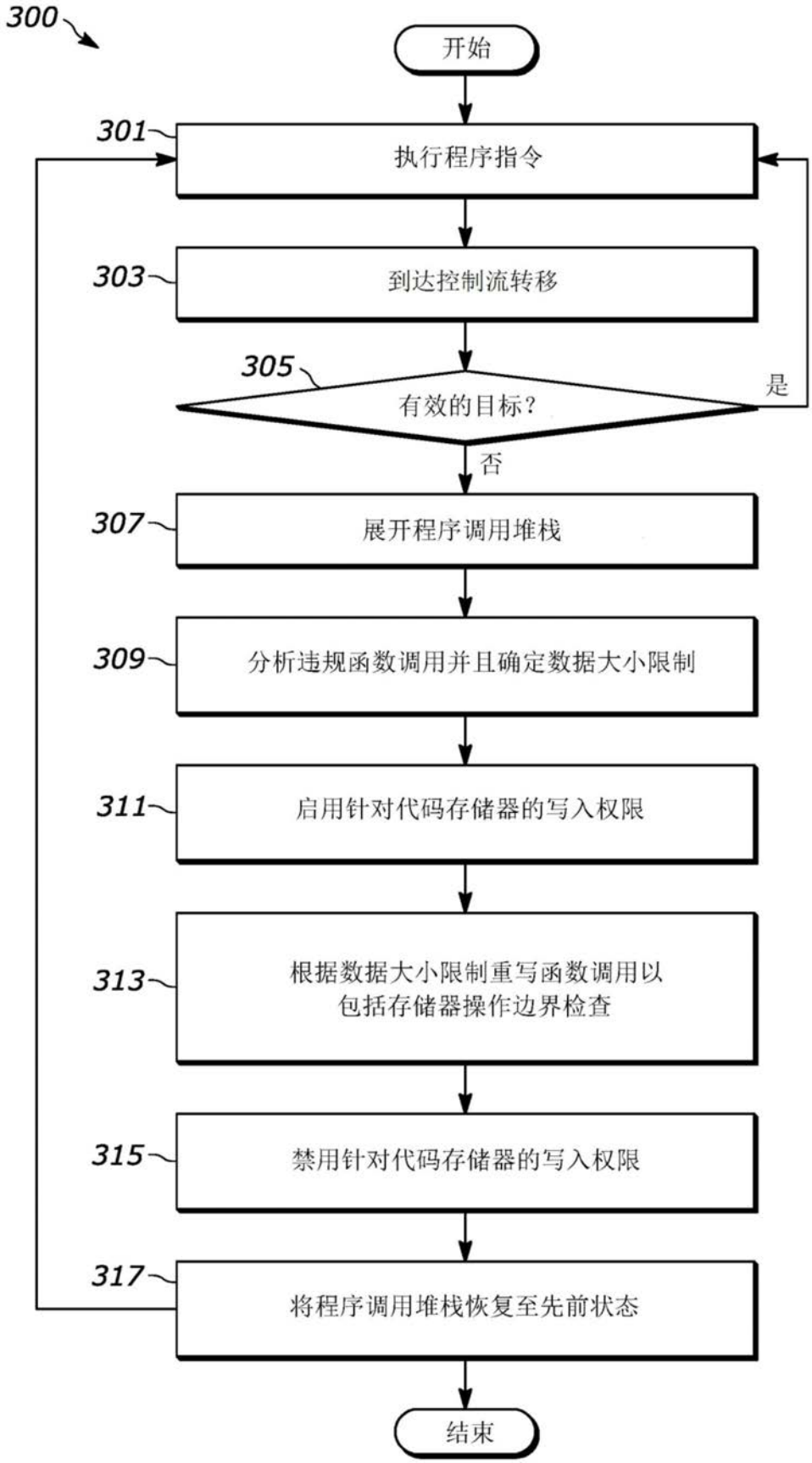

300
开始
301
执行程序指令
303
到达控制流转移
305
有效的目标?
是
否
307
展开程序调用堆栈
309
分析违规函数调用并且确定数据大小限制
311
启用针对代码存储器的写入权限
313
根据数据大小限制重写函数调用以
包括存储器操作边界检查
315
禁用针对代码存储器的写入权限
317
将程序调用堆栈恢复至先前状态
结束

图3