软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(8):2207−2222 [doi: 10.3724/SP.J.1001.2012.04131]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于硬件虚拟化的单向隔离执行模型*

李小庆[1,2], 赵晓东[1,2], 曾庆凯[1,2+]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

[2](南京大学 计算机科学与技术系,江苏 南京 210093)

One-Way Isolation Execution Model Based on Hardware Virtualization

LI Xiao-Qing[1,2], ZHAO Xiao-Dong[1,2], ZENG Qing-Kai[1,2+]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

+ Corresponding author: E-mail: zqk@nju.edu.cn

Li XQ, Zhao XD, Zeng QK. One-Way isolation execution model based on hardware virtualization. *Journal of Software*, 2012,23(8):2207−2222 (in Chinese). http://www.jos.org.cn/1000-9825/4131.htm

Abstract: A one-way isolation execution model based on hardware virtualization is proposed. In this model, the security application based on self-requirements can be divided into two parts: host process and security sensitive module (SSM). Isolated execution manager named SSMVisor, as the core component of isolation execution model, provides a one-way isolation execution environment for SSMs, not only to ensure security, but also to allow SSMs to call outside functions. As security application's trusted computing base (TCB) only includes SSMs and SSMVisor, without operating system and the security unrelated module of the applications, the size of security application's TCB is reduced effectively. A prototype system is not only compatible with the original operating system, but also light-weight. Experimental results show that the performance overhead of prototype system is very low, about 6.5%.

Key words: isolation execution; hardware virtualization; security sensitive module; TCB (trusted computing base)

摘 要: 提出了一种基于硬件虚拟化技术的单向隔离执行模型.在该模型中,安全相关的应用程序可以根据自身需求分离成宿主进程(host process)和安全敏感模块(security sensitive module,简称 SSM)两部分.隔离执行器(SSMVisor)作为模型的核心部件,为 SSM 提供了一个单向隔离的执行环境.既保证了安全性,又允许 SSM 以函数调用的方式与外部进行交互.安全应用程序的可信计算基(trusted computing base,简称 TCB)仅由安全敏感模块和隔离执行器构成,不再包括应用程序中的安全无关模块和操作系统,有效地削减了 TCB 的规模.原型系统既保持了与原有操作系统环境的兼容性,又保证了实现的轻量级.实验结果表明,系统性能开销轻微,约为 6.5%.

关键词: 隔离执行;硬件虚拟化;安全敏感模块;可信计算基

中图法分类号: TP316 **文献标识码**: A

* 基金项目: 国家自然科学基金(61170070, 60773170, 90818022, 61021062); 国家科技支撑计划(2012BAK26B01); 高等学校博士学科点专项科研基金(200802840002); 江苏省科技支撑计划(BE2010032)

收稿时间: 2011-05-16; 修改时间: 2011-07-21; 定稿时间: 2011-09-30

操作系统作为上层应用程序 TCB(trusted computing base)的一部分,其安全性是可信执行环境的重要基础.一旦操作系统被攻击,运行在其上层的安全机制或系统就会被破坏或被绕过.但是,构造安全操作系统一直是个难题.操作系统的脆弱性和不可信赖主要是源于两个方面的特征[1]:庞大的代码规模和错误隔离机制的缺失.目前流行的操作系统代码量达到千万行.有关研究表明,每千行代码中约有 6~16 个漏洞[2]或 2~75 个漏洞[3],并且随着软件规模的增大,漏洞密度还会上升.据此推算,千万行级的系统中漏洞至少数以万计.在这些操作系统中,为上层用户进程提供基本安全保证的少量安全相关代码和大量安全无关代码混杂在一起.同时,由于缺乏有效的错误隔离机制,安全无关代码中的漏洞也影响着整个操作系统的安全性.

为了提高操作系统对应用软件的安全保护能力,研究者尝试在原有操作系统的基础上添加安全增强组件或安全执行环境.例如,利用虚拟机来隔离处理安全敏感数据[4−7]或提供隔离执行环境[8];直接使用或虚拟安全协处理器来处理安全相关应用[9,10];重新设计处理器结构,增加新的硬件保护机制来直接保证应用程序和数据的安全性[11−14];基于微内核结构分离应用程序减小 TCB 大小[15,16]或对系统安全性作形式化[17]证明等等.

然而,这些研究都需要对原有的操作系统、应用程序执行环境、应用程序编写方式甚至硬件体系结构进行大规模的修改.基于现有的主流操作系统,已经形成了成熟而稳定的 API、系统服务以及大量以此为基础的应用软件.而上述解决方案复用现有的软件且运行环境十分困难,最终导致相关安全系统不能被广泛使用.

构造安全操作系统非常困难,而构造与原有软件环境兼容的安全操作系统更加困难.另一方面,安全应用程序一般仅需要使用操作系统提供的服务,而不需要操作系统对用户进程拥有不受限制的操作特权,例如随意访问、修改进程地址空间.因此,Overshadow[18],SP3[19],CHAOS[20]等绕过操作系统内核,直接为用户进程提供隔离或者加密保护,使得操作系统内核不能随意访问用户进程使用的内存和寄存器.进一步地,Flicker[21],Trustvisor[22]使用应用程序逻辑片段 PAL(piece of application logic)的概念将安全应用程序中的安全相关代码封装起来,并为其提供隔离的执行环境.虽然 PAL 在运行时属于安全应用进程(PAL 的宿主进程)地址空间的一部分,但是包括宿主进程在内的外部代码与 PAL 交互的唯一方式是调用 PAL 提供的安全服务接口,外部代码不能通过任何方式访问 PAL 所属内存和寄存器内容.

Flicker,Trustvisor 虽然为 PAL 提供了隔离的执行环境,但是存在着诸多限制,主要表现在:(1) PAL 不能调用外部代码,使用外部地址空间中的内存;(2) PAL 被调用后需一次完成所有工作,运行期间不能被中断.前者限制 PAL 不能利用外部的库函数、系统调用等,其所使用的功能均需由自身实现(这会使被保护对象规模增大),不利于 PAL 与外部频繁的协作;后者使得 PAL 不便进行费时的操作,因为这将阻塞操作系统,影响整个系统的实时性和可用性.

因此,本文提出了一种单向隔离模型(以下也称为 SSM(security sensitive module)隔离执行模型).该模型中的 SSMVisor 部件为安全敏感模块 SSM 提供了一个安全的执行环境.安全敏感模块 SSM 由安全应用程序中的安全相关代码和数据构成,属于安全应用程序的一部分,在运行时占用宿主进程的一部分地址空间.单向隔离指禁止外部对 SSM 的访问和篡改,但是允许 SSM 调用其宿主进程的代码,使用宿主进程的内存.这意味着 SSM 可以直接调用宿主进程的功能或间接调用库函数来利用系统调用,使用外部地址空间内存与外部交互.同时,允许 SSM 运行时被操作系统中断,这样 SSM 可以进行费时操作,而不会对系统响应性和实时性产生影响.SSMVisor 作为一种基于硬件虚拟化技术的 VMM,无需对原有操作系统进行任何修改,保证了系统的轻量级和高效性.

本文第 1 节介绍单向隔离执行模型.第 2 节给出系统设计.第 3 节介绍关键技术.第 4 节进行性能测试、分析和比较.第 5 节给出小结.

1 单向隔离执行模型

1.1 隔离执行相关组件

SSM 单向隔离执行模型构成的相关组件如图 1 所示.其定义如下:

定义 1(安全敏感模块 SSM). SSM 封装了安全应用程序中安全敏感的相关代码,以及其运行时需要使用的数据段和工作栈.对于应用程序,SSM 不是必须的,由应用程序来选择是否使用 SSM 单向隔离机制.同一个安全

程序可以包含多个 SSM.SSM 之间相互不可信,并视对方为宿主进程中的一部分.

定义 2(宿主进程,host process). 宿主进程是指使用 SSM 单向隔离机制的安全应用程序进程.SSM 是其中的一个特殊部分.

定义 3(普通进程,general process). 普通进程是指不涉及安全敏感服务的应用程序进程.

定义 4(客户操作系统,guest OS). Guest OS 是指为上层的进程提供运行环境的操作系统.

定义 5(SSM 隔离执行器,SSMVisor). SSMVisor 是 SSM 单向隔离执行模型的核心组件,为 SSM 提供单向隔离执行环境,相当于一个轻量级 VMM.它通过对 SSM 的全程管理,支持和保证 SSM 的单向隔离执行.

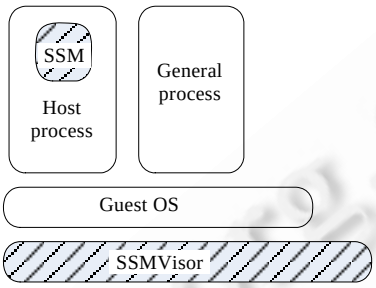


Fig.1 Component structure of SSM one-way isolation execution model

图 1 SSM 单向隔离执行模型组件结构

1.2 系统假设与安全属性

SSM 隔离执行模型的设计目标是能够为 SSM 提供阻止操作系统内核级别恶意攻击的安全保护,减少安全相关应用程序的 TCB,使其仅包含 SSM 和 SSMVisor 两部分.对于 SSM 隔离执行模型,我们给出以下假定:

假定 1. 操作系统 Guest OS、宿主进程(SSM 之外部分)和其他进程都不可信.

模型假定操作系统、宿主进程(除去 SSM 的部分)和其他进程都不属于安全应用程序的 TCB.不可信组件中可能存在漏洞并被攻击者利用,或者操作系统 Guest OS 本身就是一个恶意组件.因此,SSM 要能够在这样的环境中安全执行,必须为其提供隔离保护.

假定 2. 隔离执行器(SSMVisor)是可信的,SSM 对其自身也是可信的.

在系统中,假定 SSMVisor 是可信的,并且在操作系统启动前 SSMVisor 已经正确安装到系统中.SubVirt[23], BluePill[24]等利用虚拟机进行攻击的恶意软件在嵌套虚拟机环境中难以实施,所以我们不考虑针对虚拟机监控器进行攻击的情况.同时,假设 SSM 对自身是可信的,而不同的 SSM(包括属于同一个进程的 SSM)相互之间是不可信的.

一般来说,可信计算基(TCB)所包含软硬组件的规模与系统所要保证的安全属性有关,系统所要保证的安全属性越多,TCB 通常会越大.常用安全属性包括[25]:保密性、完整性、可恢复性和可用性.我们的目标是最小化应用程序的 TCB,在为其提供灵活的隔离执行环境的同时,要保证 SSM 的保密性和执行完整性,暂时不考虑可用性,不能防止拒绝服务攻击(DoS 攻击).SSM 隔离执行模型要保证 SSM 的以下安全属性:

安全属性 1(数据保密性). SSM 的内部数据不会被外部不可信组件访问.

安全属性 2(数据完整性). SSM 的内部数据不会被外部不可信组件破坏.

SSM 执行过程中数据的主要载体是寄存器和内存等,因此,SSMVisor 需要对 SSM 使用的寄存器和内存进行保护.而与外部交互使用的文件 I/O 和网络 I/O 数据,SSM 使用加密机制对其中的保密信息进行保护.

安全属性 3(执行完整性). SSM 按照预先设定的方式运行,其代码和执行流程不会被外部组件破坏.

1.3 单向隔离执行

SSM 单向隔离执行模型的主要目的是为 SSM 提供一个更为灵活的隔离执行环境,其单向性体现在 SSM 与

宿主进程 Host Process、操作系统 Guest OS 的关系上:SSM 可以读写宿主进程的代码和数据,而宿主进程和 Guest OS 不能读写 SSM 的内容和状态.SSM 对于外界组件功能的使用由其自身发起,外界组件无法影响到 SSM 自身的安全属性.总的来说,隔离执行环境提供以下 3 项主要功能:

- 功能 1(安全敏感服务):安全敏感服务是 SSM 完成自身功能所做的操作,宿主进程只能通过 SSM 指定入口点调用 SSM 提供的安全敏感服务.
- 功能 2(调用宿主进程):SSM 在运行时可以随时调用宿主进程的外部代码,使用外部数据.
- 功能 3(支持中断):为了支持 SSM 进行费时操作, SSM 运行时允许被 Guest OS 中断.

如图 2 所示,宿主进程 Host Process 通过指定接口调用 SSM 提供的安全服务,并在 SSM 完成任务后返回(流程 1 和流程 6).SSM 在执行过程中可以调用 Host Process 中的代码(流程 2 和流程 3),也可以被 Guest OS 中断(流程 4 和流程 5).其中,流程 2~流程 5 是本文提出的新特性.

这里给出的图 2 省略了 SSMVisor 的作用.实际上,由于 SSMVisor 处于软件层的最高特权级别,SSM 执行过程的路径控制转移会触发页面异常.该异常将被 SSMVisor 捕获处理.下文对此有详细论述.

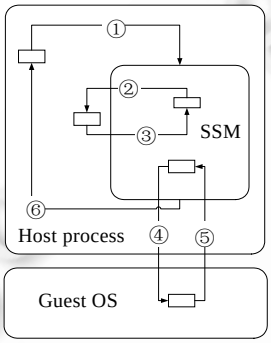


Fig.2 Work flow of one way isolation execution

图 2 单向隔离执行工作流程

1.4 内存操作权限

为了满足 SSM 隔离执行模型的安全目标,系统中各个软件组件切换运行时,SSMVisor 不断修改各个组件对应的物理内存操作权限,以保证要求的安全属性,完成单向隔离执行模型的功能.

内存操作权限包括对各相关组件的可执行权限和读写权限:

(1) 当 SSM 运行时,外部不可信组件和其他 SSM 的代码没有可执行权限.

(2) 当外部不可信组件运行时,SSM 的代码也没有可执行权限.

(3) 通过控制读写权限对 SSM 的内存信息进行保护.

(4) 保证只有宿主进程可调用安全服务接口.

定义 6(组件权限). 组件权限是指组件在运行时,对各个相关组件的代码和数据的内存所具备的操作权限.

当系统运行时,组件权限对应的物理内存操作权限的具体情况如图 3 所示.

具体组件权限见表 1.

Table 1 Permissions between the corresponding components

表 1 组件对应权限关系

组件\权限	SSM	Host process	Guest OS	SSMVisor
SSM	Trust	RWX	/ / /	/ / /
Host process	/ / /	Trust	/ / /	/ / /
Guest OS	/ / /	RWX	Trust	/ / /
SSMVisor	RWX	RWX	RWX	Trust

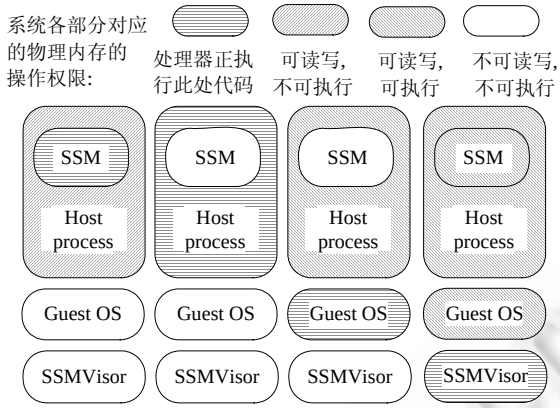


Fig.3 Memory operation permission of SSM isolation execution model

图 3 SSM 隔离执行模型内存操作权限

2 单向隔离执行系统

2.1 系统结构

为了保证 SSM 的单向隔离执行,SSMVisor 对 SSM 执行的出入实施控制.当处理器转入、转出 SSM 时,都会经由 SSMVisor 控制.SSMVisor 通过截获控制转移事件,针对不同的事件实施控制相应处理,以保证 SSM 的安全属性不被破坏.图 4 描述了 SSM 隔离执行模型的基本结构和工作原理.

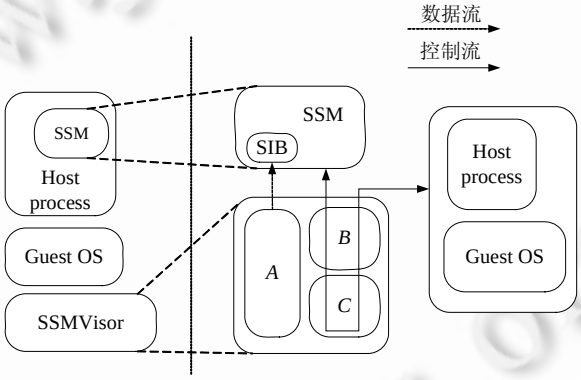


Fig.4 Architecture of SSM isolation execution model

图 4 SSM 隔离执行模型体系结构

SSMVisor 主要包括以下 3 个模块:

(1) SSM 管理(图中 *A* 模块):按照宿主进程发出的请求,处理 SSM 的注册、注销.每个 SSM 包含一个超级信息块(SIB),在注册时,根据 SSM 的特征信息(代码段和数据段内容以及 SSM 安全接口的入口位置),为其生成唯一密钥,并存放于 SSM 的 SIB 中,用于 SSM 和 SSMVisor 间的信息传递.

(2) SSM 控制转移(图中 *B* 模块):监视、截获 SSM 的控制转移事件,分析控制转移事件的类型,并将该事件交由 SSM 状态管理模块处理.

(3) SSM 状态管理(图中 *C* 模块):管理 SSM 的状态,处理状态间的转移.在状态转移的过程中,根据状态转移的方向,对 SSM 和其他不可信模块的状态进行修改,提供对 SSM 安全属性的保护.

2.2 SSM管理

SSM 注册和注销是由 SSM 宿主进程触发的.SSM 宿主进程在注册 SSM 时,需要提供 SSM 位置信息(宿主进程地址空间的某个区段)和 SSM 的服务接口信息.

定义 7(SSM 标识). SSM 标识是 SSM 在其生存期内信息的唯一标识.

在注册 SSM 时,SSMVisor 根据 SSM 的位置信息对该 SSM 进行标识,并且根据位置信息对 SSM 所属的物理内存(存放着 SSM 的代码和数据)进行保护.需要说明的是,SSM 的位置信息和 SSM 使用的物理内存的对应关系在 SSM 整个生存期是固定的.若被不可信 Guest OS 修改,则会产生对 SSM 的拒绝服务攻击,但不会影响 SSM 的安全属性.

此外,SSMVisor 需要根据 SSM 的特征信息(SSM 包含的代码段、数据段内容和提供的服务接口信息等)为其生成唯一的密钥.密钥的生成需要满足以下两个条件:

- 条件 1:对于同一个 SSM, SSMVisor 每次都会为其生成同样的密钥.
- 条件 2:攻击者无法通过 SSM 的特征信息推算出该 SSM 的对应密钥.

生成的密钥被放置到 SIB 中,供 SSM 用来加密 I/O 数据,或用于其他目的.

当宿主进程请求注销 SSM 时,SSMVisor 将清除 SSM 使用的内存,将这些内存交还给 Guest OS 管理使用.

2.3 SSM控制转移

控制转移事件是指处理器转入、转出 SSM 的事件.SSMVisor 通过页表机制来限制各软件组件所属内存的可执行权限,图 3 中描述了物理内存的执行和读写权限.当处理器执行的代码所在物理内存不具有可执行权限时,处理器会触发页面错误,SSMVisor 通过页面错误截获所发生的控制转移事件.

当 SSMVisor 截获到一个因可执行权限引起的缺页异常时,它根据该异常发生的位置找出与此事件相关的 SSM.同时,SSMVisor 分析引发此次页面错误的原因,控制 SSM 的状态转移.

与 SSM 相关的转移事件包括以下几类:

- 事件 1(调用外部事件):调用外部事件是指 SSM 调用外部函数和功能时引起的页面权限错误.
- 事件 2(中断事件):中断事件是指 Guest OS 中断正在运行 SSM 而引起的页面权限错误.
- 事件 3(SSM 服务事件):SSM 服务事件是指外部代码调用 SSM 服务接口引起的页面权限错误.
- 事件 4(SSM 返回事件):SSM 服务事件是指外部组件返回 SSM 时引起的页面权限错误.SSM 调用外部函数或被操作系统中断后,外部组件在正常情况下做完相应处理后将返回 SSM.

在获得相关的 SSM 和引发页面错误的原因后,将根据事件信息进行状态管理处理.若允许此次控制转移,则将对相应软件组件的读写和可执行权限进行修改,以允许处理器继续执行转移后的代码,并为下一次控制转移事件的捕获做准备.

2.4 SSM状态管理

SSM 状态管理是 SSMVisor 的核心工作,通过对 SSM 状态的维护和管理,实现 SSM 的单向隔离执行.其主要作用可以概括为:跟踪 SSM 的状态变化;保证 SSM 的安全属性不被破坏;保证 SSM 的信息在 SSM 不可控的情况下不会泄漏给外部不可信的进程、操作系统和外部设备.

2.4.1 SSM 状态

SSM 的生存期从注册到注销,共有 4 个 SSM 状态:

- 状态 1(空闲状态,IDLE):SSM 在未接收任何服务请求的情况下处于空闲状态.
- 状态 2(运行状态,RUNNING):SSM 接受了一个服务请求,并且正在运行时处于运行状态.
- 状态 3(等待状态,WAITING):当 SSM 主动调用宿主进程中的外部函数时,将进入等待状态,等待该函数完成.
- 状态 4(中断状态,INTERRUPTED):当 SSM 运行时,若被系统中断或异常事件中断,则进入中断状态.

4 种状态的转移关系如图 5 所示,其中,箭头上的数字仅用于标注转移过程,并不代表状态转移顺序.

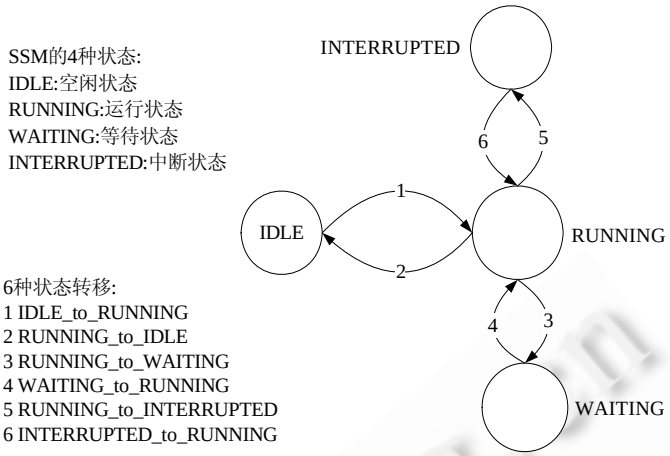


Fig.5 State of SSM and state transition

图 5 SSM 状态及状态转移关系

在注册成功之后,SSM 处于空闲状态.空闲状态下的 SSM 通过服务接口为外部提供安全服务,外部可以通过函数调用的方式调用 SSM 的服务接口.在 SSM 完成一个服务请求之前,将处于其他 3 个状态之一,并且不再接受其他外部的服务请求.完成服务后,SSM 将再次进入空闲状态,等待下一次服务请求.

2.4.2 SSM 状态转移

SSMVisor 根据 SSM 状态和发生的页面异常事件,处理状态之间的转移.不同状态转移有不同的处理过程.

- 状态转移 1(IDLE_to_RUNNING)

当外部代码通过 SSM 指定的接口地址调用 SSM 的服务时,SSM 将从空闲状态转入运行状态.处理如下:

(1) 验证状态转移合法性;

(2) 初始化 SSM 的运行环境(包括了 SSM 使用的工作栈、SSM 服务接口使用的参数信息等).

- 状态转移 2(RUNNING_to_IDLE)

当 SSM 完成任务、返回外部调用点时,SSM 将从运行状态转移到空闲状态.处理如下:

(1) 必要的寄存器清除工作,以免泄露保密信息;

(2) 恢复外部代码使用的工作栈(在 SSM 运行时使用的是 SSM 自身的工作栈).

- 状态转移 3(RUNNING_to_INTERRUPTED)

当 SSM 处于运行状态时,可能发生中断或异常事件,打断 SSM 的正常执行.当发生此类事件时,处理如下:

(1) 保护 SSM 使用的寄存器和内存;

(2) 记录 SSM 被中断位置(再次返回 SSM 时,需对返回执行的地址进行验证,并从此处开始执行);

(3) 为操作系统内核构造临时使用的用户空间工作栈.

- 状态转移 4(INTERRUPTED_to_RUNNING)

在内核处理完中断或异常事件后,控制流返回 SSM 被中断点开始执行.在控制流返回 SSM 前,处理如下:

(1) 检查中断返回的执行位置;

(2) 恢复事先保存的寄存器状态;

(3) 恢复 SSM 使用的工作栈.

对于状态转移 3 和状态转移 4,SSM 自身对中断的发生是不可知的.如果 SSM 中的程序错误触发了异常,在 Host Process 的信号处理函数中将对该异常进行处理,具体操作就是注销触发异常的 SSM.

- 状态转移 5(RUNNING_to_WAITING)

当 SSM 调用外部函数时, SSM 的状态将从运行态转移到等待态.处理如下:

(1) 若 SSM 需要使用堆栈来传递参数,则在 SSM 工作栈和宿主进程工作栈间拷贝参数信息.因为禁止外部代码访问 SSM 内存,SSM 传递的参数中应该没有指向自身地址区的指针.

(2) 记录下 SSM 的转出位置(再次返回 SSM 时,需对返回执行地址进行验证,并从此处之后开始执行).

(3) 对 SSM 的寄存器内容进行保护.

- 状态转移 6(WAITING_to_RUNNING)

当 SSM 调用的外部函数完成工作并返回 SSM 时(返回到 SSM 服务调用点的下一条语句),SSM 状态将从等待态转移到运行态.处理如下:

(1) 检查 SSM 的返回执行地址;

(2) 恢复 SSM 的寄存器内容(如外部函数具有返回值,则保留 EAX 寄存器内容不变);

(3) 恢复 SSM 的工作栈.

状态转移中内存读写权限的修改和 SSM 内存映射关系不变性检查:

在上述所有转入、转出 SSM 的状态转移处理中,SSMVisor 除了需要对于每个转移进行以上所列出的操作以外,还需要根据下一步处理器将执行哪个组件代码,对相应组件所属物理内存的读写权限进行修改,使得 SSM 可以读写宿主进程内存,外部不可信组件不可读写 SSM 内存等,具体权限如图 3 所示.

所有转入 SSM 的状态转移处理,都要判断 SSM 的位置信息(逻辑地址空间区段)与其所属的物理内存的映射关系是否保持不变,以防止外部不可信组件通过修改页表,改变 SSM 逻辑地址空间与物理内存的映射,对 SSM 实施攻击.

非法状态转移的处理:

除了上述 6 种 SSM 的合法状态转移之外,其他的状态转移都将被视为非法状态转移.例如,若 SSM 中断或等待状态下返回 SSM 时,没有从 SSM 被中断点或等待点的下一条语句开始执行,则属非法转移事件(会破坏 SSM 的执行流完整性).非法转移事件将导致宿主进程不可恢复异常.

2.5 SSM状态的安全性

2.5.1 状态的安全性

定理 1(SSM 状态安全性). 4 种 SSM 状态都能保持 3 个安全属性.

证明:SSM 的 4 个状态分别是 IDLE,RUNNING,WAITING 和 INTERRUPTED.

在 RUNNING 状态下,处理器执行 SSM 的代码.在此情况下,SSM 不会破坏其自身的 3 个安全属性,这是由 SSM 自身和 SSMVisor 的可信性假设保证的.而不可信的部分包括操作系统和程序其他相关部分没有处于运行状态,即使这些部分被恶意破坏,也无法影响到 SSM 的正常执行.

SSM 处于非 RUNNING 状态时,处理器将执行不可信组件代码(在状态转移过程中,处理器会短暂执行 SSMVisor 代码,由于 SSMVisor 的可信性假设,使得此时 SSM 的安全属性不会受到破坏),因此,下面将对剩余 3 种状态下 SSM 安全属性的保持做统一论证.

1. SSM 安全属性 1(数据保密性)和安全属性 2(数据完整性)保持的论证

不可信代码可能破坏 SSM 安全属性 1 和安全属性 2 的方式:访问或篡改 SSM 使用的内存和寄存器内容.因此,要保证 SSM 的 3 个安全属性,必须保证 SSM 使用的内存和寄存器内容的保密性和完整性.

首先,对于 SSM 所属内存完整性和保密性的保持来自于 3 个条件:

(1) SSMVisor 对于内存操作权限的控制的不可绕过性,可以防止不可信代码对 SSM 所属内存进行非法操作;

(2) 在处理器转出 SSM 后、开始执行不可信代码前,SSMVisor 都能够截获此类事件以便进行对 SSM 的资源隔离保护处理,这需要由控制转移截获的完备性来保证;

(3) 为了防止不可信代码通过修改 SSM 逻辑地址到物理地址的映射关系,从而替换 SSM 使用的物理页面,SSMVisor 需要保证在 SSM 生命周期中,其逻辑地址到物理地址的映射关系不会发生变化.

下面分别论证这 3 个条件成立.

(1) 内存操作权限控制的不可绕过性:SSMVisor 对于物理内存操作权限的控制是直接的和最终的,这意味着 Guest OS 无法通过修改自身的页表权限破坏、绕过 SSMVisor 设置的物理内存操作权限,这一点是由 SSMVisor 使用的特殊硬件页表机制决定的(嵌套页表机制,见第 3.2 节).

(2) 控制转移的截获的完备性:对于 SSM 和不可信组件,都没有拥有对对方内存的可执行权限的最终修改控制能力,而 SSM 组件不能直接执行不可信组件.同样,不可信组件也没有 SSM 的执行权限,根据前面论证的内存操作权限控制的不可绕过性,使得 SSMVisor 能够截获所有转入、转出 SSM 的事件.

(3) SSM 内存映射关系不变性:第 2.4.2 节中的内存映射关系检查保证了 SSM 内存映射关系的不变性.

上述 3 个条件的成立共同保证了 SSM 所属内存的完整性和保密性.

下面将论证 SSM 所属寄存器内容的完整性和保密性.

当处理器转出 SSM 时,SSM 的使用的寄存器内容会被 SSMVisor 保存、清空,这样,不可信代码将无法读取 SSM 使用的寄存器内容,从而保证了 SSM 寄存器内容的保密性;当处理器重新执行 SSM 代码前,SSM 使用的寄存器内容将被 SSMVisor 再次恢复,从而保证了 SSM 寄存器内容的完整性.

SSM 内存和寄存器内容的完整性和保密性共同保证了 SSM 的安全属性 1 和安全属性 2.

2. SSM 安全属性 3(执行完整性)保持的论证

不可信代码破坏 SSM 安全属性 3 的方式有两个:

- 破坏 SSM 的安全属性——篡改 SSM 的代码、工作栈和寄存器内容;
- SSM 使用被篡改的外部功能.

所以,SSM 单向隔离模型要求 SSMVisor 保证 SSM 的代码完整性、工作栈完整性和寄存器内容完整性.由于 SSM 的代码和工作栈都存放在 SSM 所属内存中,前面关于 SSM 内存完整性的证明保证了 SSM 的代码完整性和工作栈完整性.同时,论证也证明了寄存器内容完整性已经被保证.这里需要说明的是,对于其中的指令指针寄存器,SSMVisor 会根据具体情况(SSM 处于 IDLE,INTERRUPTED 或是 WAITING 下)来判断其完整性是否被破坏.

SSM 代码完整性、工作栈完整性和寄存器内容完整性共同保证了 SSM 的安全属性 3.

3. 外部功能或者中断处理被恶意篡改时 SSM 安全属性保持的论证

对于 SSM 使用了篡改的外部功能的情况:在前面的假设中,已经将外部组件视为不可信的,所以这种情况是可能发生的.但是对于外部组件的功能使用是由 SSM 自身发起的,SSM 可以控制对外部功能的使用,SSM 不能使用自身无法对返回结果正确性进行验证的外部功能.SSM 与外部组件交互时,比如 I/O 操作等,对其中的安全敏感数据和参数必须能过加密保护,防止数据的保密性被破坏.同时,SSM 在使用外部功能时,执行状态变为等待或者中断状态,其内存和寄存器中的内容被 SSMVisor 隔离保护,不可信组件将不能篡改或窃取 SSM 中的信息.外部不可信组件执行完成后,如果程序控制流不能返回到 SSM 中,就无法对 SSM 模块造成有效的攻击,影响的是不可信部分的控制流.如果不可信部分控制流返回到 SSM 中,则 SSMVisor 会对返回执行地址的合法性进行检查.对于非法操作,SSMVisor 直接将宿主进程以不可恢复异常结束,SSM 内部控制流和数据仍然不会受到影响.由此可知,被恶意篡改的外部组件无法对 SSM 的 3 个安全属性造成破坏. □

2.5.2 状态转移的安全性

定理 2(SSM 状态转移安全性). SSMVisor 对 SSM 的控制、处理过程是达到安全要求:

(1) 只能在 4 种合法状态间转换;

(2) 不会出现其他非法的状态.

证明:若要证明 SSM 只会处于 4 种合法状态之一,则需证明 SSM 在注册成功后处于合法状态中,并且此后发生的状态转移都是由合法转移事件触发的.

合法转移事件是指当该事件发生时,SSM 处于允许该事件发生的合法状态下,且该事件会触发 SSM 转移到下一个合法状态中.例如,当 SSM 处于 IDLE 状态时,只有宿主进程调用 SSM 服务接口触发的转移事件合法的,SSM 才转入 RUNNING 中;当 SSM 处于 INTERRUPTED 和 WAITING 状态时,只有返回到 SSM 被中断点的转

移事件是合法的,SSM 才转入 RUNNING 中.

第 2.3 节中定义的 4 种转移事件都是合法转移事件.由第 2.4.2 节给出的各种处理可知,在 SSM 使用外部功能或响应中断状态变为 INTERRUPTED 和 WAITING 状态时,SSMVisor 对于外部事件处理完后的合法返回地址进行了保存,同时,SSM 服务接口的合法地址在注册时也告知 SSMVisor.并且 SSMVisor 对于转移事件合法性的验证采取了白名单方式,也就是说,SSMVisor 只判断转移事件是否满足所有的合法特征,这使得转移事件的合法性验证不会出现错判的情况.因此,SSM 的状态转移都会由合法转移事件触发的.

由于 SSM 在注册成功后处于合法状态 IDLE 中,且 SSM 的状态转移都是由合法转移事件触发的,因此,SSM 在生命周期中只能在 4 种合法状态间切换.

由于转移事件的合法性验证不会出现错判情况,所以 SSMVisor 能够准确判断非法转移事件.非法转移事件不会导致 SSM 的状态转移,只会导致宿主进程运行时不可恢复异常.因此,SSM 不会处于非法状态下. □

3 实现关键技术

3.1 硬件虚拟化技术

SSM 隔离执行模型的实现基于硬件虚拟化技术,目前大多数 Intel 和 AMD 的处理器都支持硬件虚拟化.

SSM 隔离执行模型采用了 AMD SVM[26]技术.SVM 将 CPU 划分为两种操作模式:主模式和客户模式.VMM 运行在主模式下,而所有的客户操作系统都运行在客户模式下.VMM 和客户操作系统都运行在各自操作模式的最高特权级别,每个虚拟机都有一个虚拟机控制块 VMCB 与之对应,VMCB 中包含了虚拟机的执行状态信息.当需要执行一个虚拟机时,VMM 以对应的 VMCB 作为参数调用 VMRUN 指令.CPU 会根据 VMCB 的内容装载虚拟机的执行状态,并开始执行该客户操作系统代码.VMM 可以设置 VMCB 以便截获某些事件的发生,当这些事件发生时,CPU 将再次陷入 VMM,此时,VMM 可以进行相应的管理和调度工作.CPU 在陷入 VMM 之前,会将当前正在执行的虚拟机的执行状态保存在 VMCB 中,以便下次 VMM 重新运行该虚拟机时使用.

图 6 结合 AMD SVM 技术,以伪代码的形式描述了 SSM 隔离执行模型的工作原理细节.图中涉及了 SVM 中的用于初始化,启动、管理一个 VM 的扩展指令(VMLOAD,VMRUN 和 VMSAVE)和关键数据结构 VMCB. Intercepts 代表了一切引发处理器从客户模式陷入到主模式的事件,在 SSMVisor 中,主要是由控制流转入、转出 SSM 引起的页面异常事件.

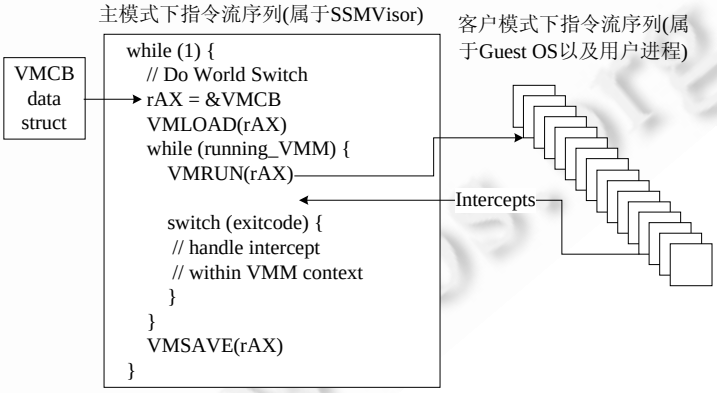


Fig.6 Operation details of SSM isolation execution model

图 6 SSM 隔离执行模型工作细节

3.2 内存保护

SSMVisor 对内存的保护主要分为以下两个方面:

(1) 对 SSM 所属的物理内存的可读写权限进行保护,以防止 SSM 的安全属性被破坏;

(2) 对系统中各软件组件所属的物理内存的可执行权限进行保护,以截获组件间的控制转移事件.

SSMVisor 使用嵌套页表机制(nest page table,简称 NPT)和 DEV 机制来达到上述目标.这两种机制都是 AMD SVM 提供用来管理物理内存的.具体而言,通过 NPT 来实现内存虚拟化工作(在软件虚拟化的实现方式中,内存虚拟化由影子页表机制来完成),并使用 NPT 页表项的 Present 位来控制相应物理内存的可读写权限(页面不存在也就意味着页面不可读写),使用页表项的 NX 位来控制相应物理内存的可执行权限(其中,使用 NX 位需要开启 PAE 机制).SSM 位置信息对应的 NPT 页表项需在注册阶段获得,并在其生命周期中保持不变.为了防止 SSM 所属物理内存被外围设备访问、修改,使用了 DEV 机制对 SSM 所属物理内存进行保护.

3.3 执行环境切换

执行环境切换包含了工作栈和寄存器内容两个方面的切换.这些工作都是在与 SSM 相关的控制转移发生时,由 SSMVisor 来完成的.

当 SSM 使用外部功能时,SSMVisor 需要为外部代码构造临时工作栈.若其中涉及函数调用包含参数,则需要将调用参数从 SSM 工作栈中拷贝到外部临时工作栈.当处理器返回 SSM 以后,SSMVisor 需要将工作栈设置回 SSM 自身的工作栈.如果外部功能调用结束后带有返回参数,则同样也需要进行工作栈参数拷贝工作.具体如图 7 所示.

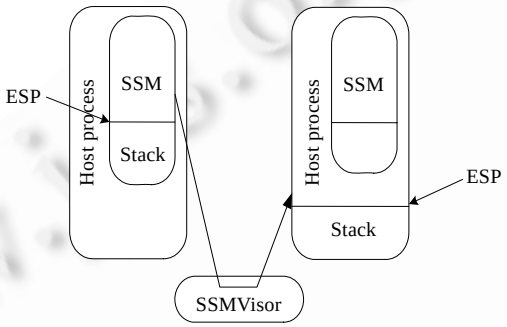


Fig.7 Stack switch triggered by control transfer

图 7 控制转移触发时堆栈的切换

寄存器内容切换包含 SSM 使用的寄存器内容进行的保存、清空和再次恢复工作,所涉及的寄存器包括通用寄存器和标识寄存器.对于其他控制寄存器和段寄存器,由于 SSM 不会直接使用该类寄存器,因此在执行环境切换时只保证这些寄存器内容不被修改,不会做清空工作.

3.4 密钥生成

目前,SSMVisor 仅为 SSM 提供密钥服务,SSM 可以利用 SSMVisor 提供的密钥进行加密处理.为了满足第 2.2 节提供的密钥生成的两个条件,SSMVisor 使用了摘要算法 SHA512.SSMVisor 首先将 SSM 特征信息和一个属于 SSMVisor 的密钥进行拼接,再使用 SHA512 摘要算法对拼接结果进行计算,计算结果就是属于该 SSM 的密钥.SHA512 算法的特性保证了 SSM 特征信息中的微小修改,都会产生差别极大的结果.由于使用了 SSMVisor 的密钥信息,攻击者仅通过 SSM 特征信息无法推知其密钥信息;同时,SHA512 算法的不可逆性能够保证,即使攻击者获得了 SSM 密钥和 SSM 特征信息,也无法推知 SSMVisor 的密钥信息.

4 系统评价

4.1 安全性分析

为了保证 SSM 隔离执行模型的安全性,在设计、实现 SSMVisor 时尽量保证其轻量级,除了对主机内存和 SSM 模块进行管理以外,基本上没有其他管理代码,因此代码量极少.在去除了注释和用于测试的代码以后,表 2

给出了 SSMVisor 各部分的代码量,其中,total 未包含头文件中的代码量.Core 为 SSMVisor 的主体部分,Lib 包含了一些字符串处理、内存管理和页表操作等工具类函数.

在此,与性能方面表现出色的 Trustvisor[22]作简单比较.可以看出,SSMVisor 代码量在 3 000 行左右.Trustvisor 代码量在 6 000 行左右,但是,其中包含 RSA 加密算法,导致 Trustvisor 代码量略多.两者都在千行代码级别.在排除 SSM 自身大小影响的情况下,SSMVisor 与 Trustvisor 对应用程序 TCB 的削减程度相当.

Table 2 Code distribution of SSMVisor

表 2 SSMVisor 代码分布表

Init	Runtime			Total	Header file
C+ASM	Core	SHA512	Lib	–	–
440+170	1 173	246	605	2 634	1 265

抗攻击能力分析:对于传统的攻击方式,例如 rootkit 攻击、木马攻击和其他类型的权限提升攻击,其目的都是获得高权限级别,突破操作系统的安全限制,从而非法窃取或破坏安全进程状态.这些攻击方式在 SSMVisor 中都失去了攻击能力.

原有的安全进程中的安全相关代码和数据都被集中在 SSM 中,由 SSMVisor 直接保护.首先,SSM 地址空间对应的物理内存,只有 SSM 自身和 SSMVisor 可以访问;其次,在控制流转出 SSM 时,SSMVisor 都会对 SSM 使用的通用寄存器内容进行保存和清除,既防止了外部代码通过寄存器内容窃取 SSM 的保密信息,又防止了外部代码通过破坏 SSM 寄存器的内容来破坏 SSM 内部状态;最后,SSM 使用的外部功能应该只是辅助安全操作的完成,与安全敏感操作无关,并且 SSM 对外部功能返回结果可以检查判定.此外,与外部功能交互的保密信息应该使用 SSMVisor 提供的密钥对其进行加密,经过加密的数据再交由宿主进程和客户操作系统处理.所以,不必担心外部功能被恶意篡改后窃取 SSM 中的保密信息或者返回错误的结果影响 SSM 的正常执行.因此,传统的攻击方式即使成功地攻破了操作系统内核,也无法破坏 SSM 的 3 种安全属性.

4.2 兼容性分析

由于使用了硬件虚拟化技术,SSMVisor 除了管理 SSM 和控制 SSM 的状态转移外不对操作系统的执行做多余的干涉,所以客户操作系统无需任何修改也无须重新编译即可运行在 SSMVisor 之上.同样,对于已有的普通应用程序也无须修改和重新编译.只有需要使用 SSM 隔离执行模型来提供安全保证的安全应用程序,才需要针对 SSM 部分做少量的修改.在实现的原型系统中,SSM 隔离执行模型提供了编译链接安全应用程序时使用的链接器脚本,应用程序开发人员只需在安全相关代码和数据的声明上添加安全模块段标识,重新编译链接程序即可完成对安全应用程序的修改,生成 SSM.因此,SSMVisor 具有较好的兼容性.

4.3 灵活性分析

灵活性是本文工作的特色.SSM 隔离执行模型的灵活性体现在允许外部组件和 SSM 间通过函数调用的方式进行交互以及 SSM 在执行过程中可以响应系统中断,并且允许 SSM 使用属于外部地址空间的内存.因此,程序员可以方便地使用外部空间中的库函数来实现自身的功能,比如调用 malloc 函数从外部地址空间中分配一段临时使用的内存块,或者使用网络相关库函数将加密信息发送到远程主机上.这里需要注意的是,不要将 SSM 保密信息以未加密的方式存放在外部地址空间中.

SSM 隔离执行模型的灵活性还体现在 SSM 运行过程中允许被操作系统中断,这样,在 SSM 中可以进行一项长时间的操作时不会影响系统响应速度,同时,允许 SSM 与外部组件进行频繁的交互.在现阶段实现的原型中,对于 SSM 模块使用外部功能,主要还是由应用程序员自己检查和约束.在后续的工作中,我们研究和开发分析 SSM 模块使用外部功能合理性的脚本,帮助提示程序员 SSM 使用的外部功能是否涉及到安全相关操作、是否存在信息泄密等情况.

4.4 性能分析

为了测试 SSMVisor 原型系统的性能开销,我们进行了以下 3 个测试:

- 测试 1:Linux 和 SSMVisor+Linux 环境下,各系统调用和页面错误的处理时间.
- 测试 2:Linux 和 SSMVisor+Linux 环境下,各 I/O 的带宽.
- 测试 3:SSMVisor+Linux 环境下的 SSM 注册和注销、外部代码调用 SSM 安全接口以及 SSM 调用外部函数的开销.

测试使用的硬件环境:CPU 为 AMD Athlon™ II x4 640 3.00GHZ,内存为 2GB DDR3.客户机操作系统为 fedora13,使用 lmbenchmark 测试套件进行性能测试.测试中为 SSMVisor 分配 128MB 内存,其余全部分配给客户操作系统.并且,由于目前实现的 SSMVisor 原型只支持单核,所以客户机操作系统仅使用处理器的 1 个核.

测试 1 和测试 2 考察因使用隔离执行模型对操作系统基本操作的开销影响.

表 3 为测试 1 的绝对测试值结果,其中,simple syscall 指简单的系统调用,用于测试用户空间到内核空间的空间切换开销.

表 4 为测试 2 的绝对测试值结果,其中,因为文件写操作比读操作对性能影响更加明显,所以这里仅给出文件写的带宽.同时,为了排除外部网络环境对网络带宽的影响,socket 套接字使用本地地址.

Table 3 Processing time of system call and page fault in Linux, SSMVisor+Linux (ms)

表 3 Linux 和 SSMVisor+Linux 环境下,各系统调用和页面错误的处理时间 (ms)

	Fork	Exec	Simple syscall	Simple write	Simple read	Prot fault	PF
Linux	57.427	351.375	0.200	0.248	0.275	0.256	3
SSMVisor+Linux	61.305	367.400	0.201	0.251	0.281	0.264	3

Table 4 I/O bandwidth in Linux, SSMVisor+Linux (MB/s)

表 4 Linux 和 SSMVisor+Linux 环境下,各 I/O 带宽 (MB/s)

	File write bandwidth	Socket bandwidth	AF_UNIX sock bandwidth	Pipe bandwidth
Linux	30.047	1 906.070	3 225.310	2 495.660
SSMVisor+Linux	29.663	1 850.820	3 122.700	2 458.270

图 8 和图 9 分别为通过测试 1 和测试 2 得到的 SSM 隔离执行模型带来的性能开销的相对值.从图中可以直观地看出,由于使用了硬件虚拟化技术,以及 SSMVisor 简洁的设计,SSMVisor 带来的相对性能开销最大为 6.5%.虽然硬件虚拟化的 NPT 页表机制,客户机操作系统增加了一轮页表地址转换的过程,将会带来比较明显的开销,但是 SSMVisor 利用 AMD SVM 的 ASID 机制,降低了查询 NPT 页表的次数,使最终的性能开销很低.

而 Trustvisor[22]原型测试结果的性能开销最大为 7%.上述性能实验结果表明,SSMVisor 在增加了灵活性的情况下,与 Trustvisor 的性能开销相当.

表 5 为测试 3 的实验结果,即各 SSM 相关操作的性能开销.其中,所有实验结果都是 100 次同一类型操作时间的平均值,每次测试结果的波动不超过 5%.其中,call simple safe function 是指外部代码调用一个空的 SSM 安全服务,call simple outside function 是指 SSM 调用一个空的外部函数.由于 SSMVisor 需要对 SSM 所属的物理内存进行保护,SSM 相关操作的开销将与 SSM 的大小相关.因此,测试 3 中测试了不同大小的 SSM.

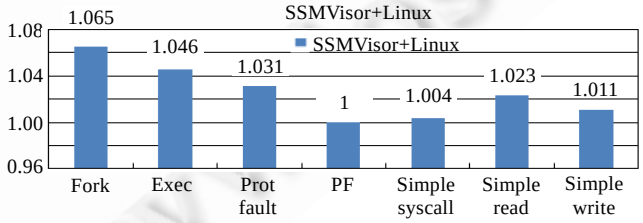


Fig.8 Overhead ratio of system call and page fault in SSMVisor+Linux comparing Linux

图 8 SSMVisor+Linux 环境相对于 Linux 环境中系统调用和页面失效处理的性能开销比例

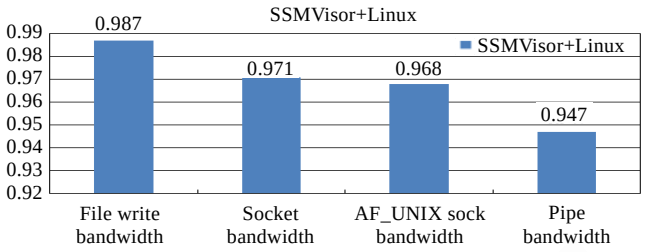


Fig.9 I/O bandwidth ratio in SSMVisor+Linux comparing Linux

图 9 SSMVisor+Linux 环境相对于 Linux 环境的各种 I/O 相对带宽比例

Table 5 Performance cost of operations about SSM (ms)

表 5 SSM 相关操作的性能开销 (ms)

SSM size (KB)	Register SSM	Unregister SSM	Call simple safe function	Call simple outside function
8	1.620	0.953	3.543	3.620
16	2.671	1.467	4.367	4.480
32	3.980	2.427	5.810	5.873
64	5.712	3.823	8.616	8.840
128	7.823	4.956	14.276	14.766
256	15.395	9.393	25.526	27.313

从表 5 中可以看出,对于 SSM 相关的单个操作,其性能开销与其大小基本成线性相关关系.其中,注册 SSM 的开销主要来源于 SSMVisor 对 SSM 所属页表项的权限修改和摘要哈希值的计算;注销 SSM 的开销主要来源于 SSMVisor 对 SSM 所属页表项的权限修改和 SSM 所属物理内存的清零工作;调用空的安全接口的开销主要来源于至少两次的 SSM 与外部之间的模式切换,每次切换都需要修改页表权限并构造临时工作栈或还原 SSM 工作栈操作;SSM 调用外部空函数的开销同上,所以两者开销基本一致.为了降低 SSM 相关操作的开销,可以尽量编写小的 SSM,这样可以在提高 SSM 安全性的同时减少调用外部函数的次数,提高模型的性能.

5 总 结

本文提出了一种单向隔离执行模型,为安全敏感代码提供灵活的、细粒度的、高效的安全运行环境.灵活性特点使得 SSM 与外部组件之间可以通过函数调用方式进行交互.SSM 可以使用外部地址空间和库函数等资源,有利于简化 SSM 自身的实现,压缩 SSM 的规模.在 SSM 运行时允许被操作系统中断,保证 SSM 可以处理费时服务而不会影响系统响应速度.细粒度地安全服务隔离执行,有助于缩小 TCB,提高安全保护效率.系统原型实现与测试表明,SSM 隔离执行模型在 TCB 规模和性能开销方面的表现与 Trustvisor 相当,同时为 SSM 提供了更加灵活的隔离执行环境.

References:

[1] Tanenbaum AS, Herd JN, Bos H. Can we make operating systems reliable and secure? IEEE Trans. on Computers, 1996,39(5): 44−51. [doi: 10.1109/MC.2006.156]

[2] Basili VR, Perricone BT. Software errors and complexity: An empirical investigation. Communications of the ACM, 1984,27(1): 42−52. [doi: 10.1145/69605.2085]

[3] Ostrand TJ, Weyuker EJ. The distribution of faults in a large industrial software system. ACM SIGSOFT Software Engineering Notes, 2002,27(4):55−64. [doi: 10.1145/566171.566181]

[4] England P, Lampson P, Manferdelli J, Peinado M, Willman B. A trusted open platform. IEEE Trans. on Computers, 2003,36(7): 55−62. [doi: 10.1109/MC.2003.1212691]

[5] Garfinkel T, Pfaff B, Chow J, Rosenblum M, Boneh D. Terra: A virtual machine-based platform for trusted computing. ACM SIGOPS Operating System Review, 2003,37(5):193−206. [doi: 10.1145/1165389.945464]

[6] Ta-Min R, Litty R, Lie D. Splitting interfaces: Making trust between applications and operating systems configurable. In: Proc. of the 7th Symp. on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2006. 279−292.

[7] Rosenblum NE, Cooksey G, Miller BP. Virtual machine-provided context sensitive page mappings. In: Proc. of the 4th ACM SIGPLAN/SIGOPS Conf. on Virtual Execution Environments. 2008. 81−90. [doi: 10.1145/1346256.1346268]

[8] Huai JP, Li Q, Hu CM. Research and design on hypervisor based virtual computing environment. Journal of Software, 2007,18(8): 2016−2026 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/2016.htm [doi: 10.1360/jos182016]

[9] Dyer JG, Lindemann M, Perez R, Sailer R, Doorn LV, Smith SW, Weingart S. Building the IBM 4758 secure coprocessor. IEEE Trans. on Computer, 2001,34(10):57−66. [doi: 10.1109/2.955100]

[10] Berger S, Cáceres R, Goldman KA, Perez R, Sailer R, Doorn LV. vTPM: Virtualizing the trusted platform module trusted platform module. In: Proc. of the 15th USENIX Security Symp. Berkeley: USENIX Association, 2006. 305−320.

[11] Dwoskin JS, Lee RB. Hardware-Rooted trust for secure key management and transient trust. In: Proc. of the 14th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2007. 389−400. [doi: 10.1145/1315245.1315294]

[12] Lee RB, Kwan PCS, McGregor JP, Dwoskin J, Wang ZH. Architecture for protecting critical secrets in microprocessors. In: Proc. of the 32nd Int'l Symp. on Computer Architecture. New York: ACM Press, 2005. 2−13. [doi: 10.1145/1080695.1069971]

[13] Lie D, Thekkath CA, Mitchell M, Lincoln P, Boneh D, Mitchell JC, Horowitz M. Architectural support for copy and tamper resistant software. ACM SIGPLAN Notices, 2000,35(11):168−177. [doi: 10.1145/356989.357005]

[14] Peinado M, Chen YQ, England P. NGSCB: A trusted open system. In: Proc. of the Information Security and Privacy. LNCS 31, Berlin: Springer-Verlag, 2004. 86–97. [doi: 10.1007/978-3-540-27800-9_8]

[15] Singaravalu L, Pu C, Härtig H, Helmuth C. Reducing TCB complexity for security-sensitive applications: Three case studies. In: Proc. of the 1st ACM SIGOPS/EuroSys European Conf. on Computer Systems. 2006. [doi:10.1145/1218063.1217951]

[16] Shapiro JS, Smith JM, Farber DJ. EROS: A fast capability system. In: Proc. of the ACM Symp. on Operating Systems Principles. New York: ACM Press, 1999. 21−32. [doi:10.1145/346152.346191]

[17] Klein G, Elphinstone K, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engelhardt K, Kolanski R, Norrish M, Sewell T, Tuch H, Winwood S. seL4: Formal verification of an OS kernel. In: Proc. of the 22nd ACM Symp. on Operating Systems Principles. New York: ACM Press, 2009. 207−220. [doi: 10.1145/1629575.1629596]

[18] Chen XX, Garfinkel T, Lewis EC, Subrahmanyam P, Waldspurger CA, Boneh D, Dwoskin J, Ports DRK. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. In: Proc. of the 42th ACM SIGOPS Operating System Review-EurosSys 2008, Vol.2. New York: ACM Press, 2008. 2−13. [doi: 10.1145/1346281.1346284]

[19] Yang J, Shin KG. Using hypervisor to provide data secrecy for user applications on a per-page basis. In: Proc. of the 4th ACM SIGPLAN/SIGOPS Conf. on Virtual Execution Environments. New York: ACM Press, 2008. 71–80. [doi: 10.1145/1346256.1346267]

[20] Chen HB, Zhang FZ, Chen C, Yang ZY, Chen R, Zang BY, Mao WB. Tamper-Resistant execution in an untrusted operating system using a VMM. Parallel Processing Institute Technical Report, FDUPPITR-2007-0801, Shanghai: Fudan University, 2007. 1−16.

[21] McCune JM, Parno BJ, Perrig A, Reiter MK, Isozaki H. Flicker: An execution infrastructure for TCB minimization. In: Proc. of the 42th ACM SIGOPS Operating System Review-EurosSys 2008, Vol.4. New York: ACM Press, 2008. 315−328. [doi: 10.1145/1357010.1352625]

[22] McCune JM, Li YL, Qu N, Zhou ZW, Datta A, Gligor V, Perrig A. Trustvisor: Efficient TCB reduction and attestation. In: Proc. of the 2010 IEEE Symp. on Security and Privacy. Washington: IEEE Compute Society, 2010. 143−158. [doi: 10.1109/SP.2010.17]

[23] King ST, Chen PM, Wang YM, Verbowski C, Wang HJ, Lorch JR. SubVirt: Implementing malware with virtual machines. In: Proc. of the 2006 IEEE Symp. on Security and Privacy. Washington: IEEE, 2006. 314−327. [doi: 10.1109/SP.2006.38]

[24] Rutkowska J. Subverting Vista kernel for fun and profit. In: Proc. of the Black Hat in Las Vegas. 2006. http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf

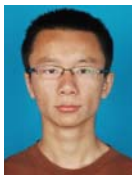
[25] Härtig H, Hohmuth M, Feske N, Helmuth C, Lackorzynski A, Mehnert F, Peter M. The Nizza secure-system architecture. In: Proc. of the 1st Int'l Conf. on Collaborative Computing. 2005. 102−111. [doi: 10.1109/COLCOM.2005.1651218]

[26] AMD. AMD64 virtualization codenamed “pacifica” technology: Secure virtual machine architecture reference manual. No.33047, Revision 3.01, Sunnyvale Advanced Micro Devices, 2005. 1−124. http://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf

附中文参考文献:

[8] 怀进鹏,李沁,胡春明.基于虚拟机的虚拟计算环境研究与设计.软件学报,2007,18(8):2016−2026. http://www.jos.org.cn/1000-9825/18/2016.htm [doi: 10.1360/jos182016]

李小庆(1986－),男,安徽太和人,硕士,主要研究领域为信息安全.



赵晓东(1987－),男,硕士,主要研究领域为信息安全.

曾庆凯(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信息安全,分布计算.