基于 AMD 硬件内存加密机制的关键数据保护方案

吴宇明, 刘宇涛, 陈海波

上海交通大学并行与分布式系统研究所, 上海 中国 200240

摘要 长期以来, 保护应用程序关键数据(如加密密钥、用户隐私信息等)的安全一直是个重要问题, 操作系统本身巨大的可信计算基使其不可避免的具有许多漏洞, 而这些漏洞则会被攻击者利用进而威胁到应用程序的关键数据安全。虚拟化技术的出现为解决此类问题提供了一定程度的帮助, 虚拟化场景下虚拟机监控器实际管理物理内存, 可以通过拦截虚拟机的关键操作为应用程序提供保护, 而硬件内存加密机制则能够解决应用程序在运行时内存中明文数据被泄露的问题。本文基于虚拟化技术和 AMD 的硬件内存加密机制, 提出了一套高效的关键数据保护方案, 并通过应用解耦和技术将关键数据与代码与其余的正常数据与代码分离并置于隔离的安全环境中运行从而达到保护关键数据的目的。测试显示, 软件带来的系统性能开销小于 1%, 关键部分的性能开销小于 6%, 常见应用的延迟在接受范围内。系统能够成功保护应用程序如私钥等关键数据免受恶意操作系统的读取与 Bus Snooping、Cold Boot 等物理攻击。

关键词 硬件内存加密; 数据保护; 内存泄露; 虚拟化
中图法分类号 TP309 DOI 号 10.19363/j.cnki.cn10-1380/tn.2018.01.003

Elimination of Memory Disclosure Attacks based on AMD Memory Encryption

WU Yuming, LIU Yutao, CHEN Haibo

Institution of Parallel and Distributed Systems, Shanghai Jiao Tong University, Shanghai 200240, China

Abstract For a long time, the security of critical data like encryption keys and private information has been an important concern. The huge trust computing base (TCB) of the operating system makes it vulnerable to various of attacks which are leveraged by malicious attackers to stealing the critical data from the applications. The virtualization technology can resolve some of these problems. Since the virtual machine monitor (VMM) Runs at the highest privilege level, it is responsible for managing the physical hardware resources and can easily intervene the selected critical operations of running OS and applications, to enforce pre-defined security policies. Recently, hardware memory encryption technology can also mediate some of these problems from the hardware level by encrypting memory data via dedicated hardware during runtime. Combined with virtualization technology and the newly proposed AMD memory encryption hardware. This paper presents a novel solution to protect the critical application data from the compromised OS in an efficient and fine-grained manner. Through application decomposition mechanism, it can separate the *critical compartments* from the other parts of the application and put them into the isolated environment. Evaluations show that the system performance overhead is less than 1% and the performance slowdown of the secure runtime environment is less than 6%. The latency of common applications is in an acceptable range. Security analysis shows that the system can successfully protect critical application data against a compromised operating system stealing as well as physical attacks including bus snooping and cold-boot attacks.

Key words Memory encryption; privacy protection; memory disclosure; virtualization

1 引言

长期以来, 如何保护应用程序关键数据(如加密密钥、用户隐私信息等)的安全一直是一个重要的问题。例如 2014 年肆虐全球的 Heartbleed[1]心脏滴血攻击就利用了 OpenSSL 中的指针越界错误, 使程序能够向攻击者返回当前内存中 64KB 的数据, 并借此偷取如密钥等隐私信息, 造成了极为恶劣的影响。除了

通讯作者: 陈海波, 博士, 教授, haibochen@sjtu.edu.cn。
本课题得到国家重点研发计划 No. 2016YFB1000104 支持。
收稿日期: 2017-09-15; 修改日期: 2017-11-17; 定稿日期: 2017-12-07

应用程序本身的漏洞, 操作系统(Operating System, OS)本身由于巨大的可信计算基不可避免的也具有一些漏洞, 这些漏洞同样会被攻击者利用从而造成十分恶劣的影响。如表 1 所示, 仅 2016 年一年各主要OS均被发现上百个漏洞, OS本身的可信程度令人担忧。而除了软件中暴露的问题以外, 攻击者还可能利用硬件的缺陷实施对应用程序关键数据的恶意读取, 如 Bus Snooping[2]和 Cold Boot[3]攻击。在大量的漏洞与成功实施的攻击下, OS 的健壮性也不禁会引起人们的质疑。

对于软件层面的攻击, 一种解决方案是建立一个相对独立的可信环境, 在可信环境内部执行应用程序从而避免外部而来的攻击。但是由于 OS 往往处于最高权限级, 因此在这之上建立的安全环境往往会受到来自 OS 自身漏洞的影响, 而在同级建立这样的可信环境往往较为困难。Dautenhahn Nathan[4]等人利用CR0中的WP位, 通过将OS的页表设置为只读, 保证所有对页表的修改必须要经过可信环境 Nested Kernel, 从而建立一个与 OS 本身同级别的可信环境, 将其余大量的 OS 代码移出了可信基。这种方法虽然可以阻止恶意的修改, 但是无法阻止攻击者对应用程序私密数据的恶意读取; 而如 Bus Snooping、Cold Boot 等物理攻击则可以完全绕开软件层面的解决方案。这本质上是由于应用程序在运行时内存中的数据均是明文, 攻击者只要知道应用程序私密数据的物理地址便可直接读取应用程序在内存中的明文数据。

表 1 2016 年主要操作系统 CVE 数目

Table 1 Number of CVEs in mainstream OS (2016)

系统名称	CVE 数目	系统名称	CVE 数目
Android	523	Debian Linux	319
Ubuntu	278	Leap OS	259
OpenSUSE	228	Linux Kernel	217
MacOS	215	Windows 10	172
IOS	161	Windows Server 2012	156

虚拟机监控器(Virtual Machine Monitor, VMM)的出现提供了一种新的途径。由于 VMM 运行在高于客户虚拟机(Guest Virtual Machine, Guest VM)中OS(Guest OS)的特权级并实际管理物理内存, 因此可以利用 VMM 进行介入, 对应用程序中的关键数据进行保护。但是这类防御措施仍然无法阻止如 Bus Snooping 或 Cold Boot 等物理攻击。

为了更彻底地解决应用程序数据泄露问题, AMD 提出了硬件内存加密机制: 安全内存加密(Secure Memory Encryption, SME)与安全加密虚拟化(Secure Encrypted Virtualization, SEV)[5]。SME 主要用于保护宿主机(Host)防止物理攻击, SEV 则主要用于虚拟化场景下 Guest OS 的安全。两者均使用可信硬件加密内存, 并通过页表进行设置来高效完成对加密页的管理。但是现有的 SME 与 SEV 均无法保护运行在 Guest OS 中的应用程序的数据安全。

本文基于AMD SEV硬件, 利用现有的虚拟化环境 Xen[6], 对其进行了软件扩展, 提出了一套解决方案 Sedora, 在不可信的 Guest OS 环境下保护应用程序关键数据的安全。Sedora 建立一个独立的安全运行环境, 在其中执行与应用程序关键数据相关的代码; 而针对包含物理攻击在内的恶意内存读取, Sedora 利用 SEV 的内存加密特性, 对安全运行环境进行内存加密。Sedora 将应用程序进行解耦和并建立不可被篡改的通信机制与应用程序直接通信, 将与关键数据相关的代码逻辑置于安全运行环境来避免潜在的攻击。安全分析和性能评估证明了 Sedora 是切实有效可行的。

本文的主要贡献如下:

1) 利用虚拟化环境下VMM负责模拟CPUID的特性, 对 VMM 进行扩展, 通过创建不可被篡改的通信机制, 使得 VMM 可以与应用程序直接安全地进行通信。

2) 对 AMD SEV 与 SME 进行全面分析, 以及利用 SEV 与 SME 机制提出保护应用程序在不可信 Guest OS 中关键数据安全的解决方案。

3) 基于 Xen 的软件扩展方案 Sedora, 通过建立安全的可信环境, 并利用不可被篡改的通信机制来解决应用程序的数据安全问题。

4) 基于 Xen 实现的可用原型系统, 以及对应的安全和性能评估。

本文结构如下: 第 2 章介绍相关背景技术; 第 3 章介绍 Sedora 的设计与实现; 第 4 章介绍 Sedora 的应用场景; 第 5 章是系统的安全分析; 第 6 章展示系统的性能评估; 在第 7 章介绍与本文相关的工作; 随后在第 8 章围绕系统进行了简单的讨论和对系统局限的分析, 并介绍了相应的后续工作; 最后第 9 章总结全文。

2 背景

利用虚拟化技术在脆弱的 Guest OS 上保护应用程序的关键数据安全已经在学术界被大量讨论, 如 SeCage[7]、OverShadow[8]等。但是与已有的保护技术相比, 硬件加密内存具有无可比拟的优点。Intel 和 AMD 都在近年提高了对安全的重视程度并在主流的

处理器中加入了硬件内存加密机制[5, 9], 基于硬件的内存加密将会在普通用户中大量部署; 其次, 加密后的内存不但能够抵御传统的基于软件的数据泄露攻击, 而且能够抵御如 Bus Snooping、Code Boot 这样的物理攻击; 最后, 基于硬件的防御手段对原有系统的修改要求极小, 在现有软件栈的基础上只需少量的修改即可部署。

Intel 曾提出了软件防护扩展(Software Guard Extensions, SGX)技术[9], 该技术能够通过可信硬件加密应用程序内存来抵御数据泄露攻击。而与明确保护应用程序关键数据安全的 SGX 不同, AMD 的 SME 与 SEV 机制旨在保护操作系统免受来自硬件的物理攻击, 因此不能直接参考 SGX 用于保护应用程序的数据安全。在本节, 本文将首先介绍 AMD 硬件内存加密相关机制与局限, 之后会简单介绍与本文相关的 Xen 虚拟化技术。

2.1 AMD 硬件内存加密机制

AMD 于 2016 年提出硬件内存加密机制 SME 与 SEV[5]并将其与现有的 AMD-V 虚拟化技术[10]进行了结合。如图 1 所示, 在原有的 CPU 中引入了新的片上芯片系统并将其与原有的内存控制器进行了结合以支持 SEV 和 SME 的实现。所有内存加密密钥的生成和管理均由一个专有安全处理器来负责, 上层的软件通过相应的内存映射寄存器对安全处理器发送请求来生成、安装、更换或撤销密钥。密钥被安装在内存控制器上, 当 CPU 经由内存控制器读写内存时, 内存控制器利用密钥对数据进行解加密。

SME 主要用于抵御物理攻击, 原有页表项中的某一高位被识别为加密位 C-bit, OS 可以通过配置加密位 C-bit 来设置对应的物理页是否开启 SME 支持, 从而达到页粒度的管理。SEV 则结合新增的指令集与 Guest 页表, 以类似 SME 的方式实现对 Guest VM 内存的加密保护。当一个 Guest VM 以 SEV 方式启动时, 内存加密固件会初始化 Guest VM 的 SEV 上下文, 将其保存到片上系统并返回一个 *Handle* 作为标识, 每一个 *Handle* 对应于一个 Guest VM 的地址空间 ID(Address Space ID, ASID)。当 VM 被调度的时候, VMM 会向安全处理器发送 *ACTIVATE* 指令并通过命令缓冲区传入对应 Guest VM 的 ASID 和 *Handle*, 安全处理器则会将对应 VM SEV 上下文中的密钥装入内存控制器并用 ASID 进行标记。如图 1 所示, Xen VMM 除原有的 Domain0 外运行了三个 Guest VM, 其中 VM1 与 VM2 分别使用 Key1 和 Key2 对物理内存进行加密, 通过配置 guest 页表中的 C-bit, VM1 与 VM2 可以以页粒度管理物理内存加密, 而未开启 SEV 的 VMn 则和原来一样。需要注意的是, VMM 自身也可以通过设置自身页表或嵌套页表(Nested Page Table, NPT)中的 C-bit 来开启 SME 支持以进一步抵御物理攻击, 当同一块物理页开启了两种加密支持时, 内存控制器会优先采用与 SEV 相关的 C-bit 并使用对应的密钥对内存进行加解密。

2.2 AMD 内存加密的局限

AMD 内存加密机制包含 SME 与 SEV 两种, SME 的目的在于保护 OS 抵御来自硬件的物理攻击。如我

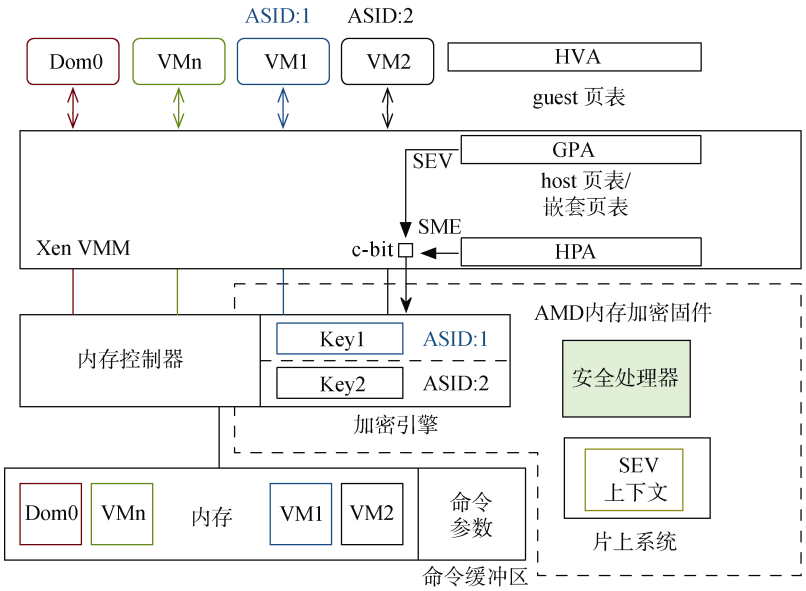

图 1 AMD 硬件内存加密架构

Figure 1 AMD memory encryption architecture

们在 2.1 节所讨论的, OS 可以通过配置页表项中的 C-bit 来管理加密页, VMM 则可以进一步通过配置 NPT 中的页表项来启动这一机制。SME 的对上透明性可以减轻部署的工作量, 但是对于 OS 或 VMM 本身来说, 读写内存本身不会受到任何影响。这也就意味着在这之上的应用程序或虚拟机也同样不会受到内存加密的影响, 因此无法借助 SME 来实现对应用程序关键数据的保护。

与 SME 不同, SEV 的设计目的在于保护 VM 自身, 页表的配置由 Guest OS 完成, 因此它能够保证 VM 在运行时的内存为密文, 从而防止来自其他 VM 甚至来自 VMM 的恶意读取。但是, SEV 之于 Guest OS 就如 SME 之于 Host OS 一样, SEV 对于运行在 Guest OS 中的应用程序来说是透明的, 因而 SEV 也不能直接用来保护应用程序关键数据的安全。

2.3 Xen 虚拟化技术

Xen[6]曾因其提出的半虚拟化概念而一夜成名, 通过对 Guest OS 进行显式的修改, 使 Guest OS 能够利用 Hypercall、影子页表和半虚拟化 I/O 来高效替换在传统虚拟化中的软件模拟。而随着 Intel VT-x[11]与 AMD-V[10]的发布, 当前的硬件已经能高效支持 CPU 与内存的虚拟化, 而 I/O 虚拟化则依旧沿用了 Xen 中设计的半虚拟化方案。

CPU 虚拟化: AMD-V 将 CPU 的运行模式划分为 guest 模式与 host 模式, VMM 与负责管理 VM 的 Domain0 运行在 host 模式中, 普通的 Guest VM 运行在 guest 模式中。两种模式都具有特权级 ring3 与 ring0 以分别运行应用进程与内核进程。在 Guest VM 启动之前, VMM 会为每一个虚拟 CPU(vCPU)初始化一个控制块(Virtual Machine Control Block, VMCB), VMCB 中存储着 Guest VM 当前 vCPU 运行时的各种状态。当 VMCB 被初始化完成后, VMM 可以将 VMCB 的地址作为参数, 通过 *VMRUN* 指令将控制流跳转到 guest 模式中 VM 的启动代码并开始执行, 硬件则负责将 VMCB 中的运行状态同步到当前的 CPU 中。当 Guest VM 遇到特权指令或触发在 VMCB 中配置过的下陷条件时, 控制流会下陷到 VMM 中, 同时硬件会再次将 CPU 的状态同步至对应的 VMCB 中。之后 VMM 负责读取 VMCB 中的退出原因并进行处理, 随后再次通过 *VMRUN* 继续执行 Guest VM 的代码。

虚拟化下的 CPUID: 作为一条特殊指令, CPUID 的目的在于发现 CPU 所具有的各项功能特性。查询时对应的 ID 会作为参数存入 EAX, 在调用 CPUID 后, 通过读取 EAX 到 EDX 中的返回值, 系统便可以知道当前物理 CPU 所支持的功能。在虚拟化环境下, 由于 Guest VM 无法直接获取硬件的信息, 因此 CPUID 也会作为特权指令下陷到 VMM 中, 由 VMM 负责填写对应的信息以供 Guest OS 读取。CPUID 的特别之处在于它可以被运行在 ring3 级的应用程序直接调用, 控制流则会直接下陷到 VMM 从而绕过 Guest OS。

嵌套页表: AMD 使用嵌套页表(Nested Page Table, NPT)[12]来完成对内存虚拟化的支持。如今的主流 OS 均采用页表系统来管理虚拟地址到物理地址的映射, 这样可以提高内存的使用率并且易于管理, 并且所有的应用程序均可运行在相同的虚拟地址空间中, 从而规范了程序的设计与开发。负责完成地址映射的数据结构则是页表(Page Table, PT), 在虚拟化场景下, 页表负责完成客户机虚拟地址(Guest Virtual Address, GVA)到客户机物理地址(Guest Physical Address, GPA)的映射, 而 NPT 则负责完成 GPA 到宿主机物理地址(Host Physical Address, HPA)的映射。Guest 通过 guest Control Register 3(gCR3)寄存器来管理自己的页表, NPT 则由 VMM 通过 nested CR3(nCR3)寄存器来管理。因此内存访问会经过 GVA 到 GPA 再到 HPA 的两个翻译过程。当 NPT 发生缺页错误时, Guest 会下陷到 VMM, VMM 则根据 VMCB 中的错误地址与原因, 分配新页并填写 NPT 来修复缺页错误。

Xen 内存共享: 由于 Guest VM 都运行在各自的地址空间中, 因此 VMM 必须提供一套供 VM 之间进行内存共享的机制, Xen 通过授权表满足 VM 对内存共享的需求。Xen 在启动 Guest VM 时会为每一个 VM 创建一张授权表, 当 VM 准备共享自己的内存时, 它会首先利用相关 Hypercall 在自己的授权表中添加一条包含授权目标 ID、将要共享的页地址和授予权限的相关信息, Xen 在处理 Hypercall 时会通过共享页的 GPA 查询到 HPA 并填到目标 ID VM 的 NPT 中, 同时根据授权信息将其映射为对应的权限。这样 VM 间就可以通过授权表来进行安全的内存共享。

2.4 CIL 静态分析

CIL(C Intermediate Language)[13]是一个用于 C 语言程序分析和转化的前端工具。CIL 能够分析源代码并进行类型检查, 同时还能将程序进行等价的简化以减少分析的复杂度。CIL 最终的结果包含三种基本元素: 表达式、指令和语句, 以及这些基本元素的前继与后继。借助这些信息, 使用者可以进行污染标记并发现所有访问被污染对象的程序部分; 也可以直接根据已有信息建立程序的控制流图(CFG, Control

Flow Graph)或抽象语法树(AST, Abstract Syntax Tree)从而为下一步的工作做准备。

2.5 威胁模型

Sedora 的目标是保护应用程序中关键数据 (如加密密钥、用户隐私信息等) 的安全, 防止其被攻击者恶意读取。对于应用程序, 我们假定攻击者在系统部署完成后能够通过网络远程以及在 Guest OS 本地阻塞或截取应用程序的数据。攻击者可以利用 Guest OS 本身的信息泄露漏洞, 结合已有的成熟攻击手段从而达到对 Guest OS 内存数据的任意读取, 进而恶意窃取应用程序中的关键数据; 攻击者也可以借助于其他应用程序利用 Guest OS 与被保护程序共享同样的内存从而对共享内存中的数据进行修改。同时, Sedora 还考虑 Cold Boot 与 Bus Snooping 这样的物理攻击。

本文假设虚拟机监控器 VMM 是可信的, 利用 VMM 本身的漏洞从 VM 层面攻击 VMM 或攻击其他 VM, 或使 Guest OS 与 VMM 配合攻击应用程序等不在本文的讨论范围内。此外, 本文也相信 AMD 内存加密的相关硬件, 侧信道攻击[14]则不在本文的讨论范围内。

3 Sedora 设计与实现

3.1 概述

Sedora 是基于 AMD 硬件内存加密机制, 对 Guest VM 中应用程序关键数据进行保护的一套系统设计方案。其主要是通过分离应用程序为关键部分与正常部分, 通过建立隔离的安全运行环境来执行与关键数据相关的关键部分, 并通过一套由安全通信机制组成的安全调用来完成与正常部分的通信, 在通信过程中 Sedora 负责维护安全运行环境的机密性与安全调用部分的完整性。

Sedora 主体分为两部分: 运行在正常环境下的正常部分和运行在安全运行环境中的关键部分。如图 2 所示, 整个系统分为四个阶段: 1) 需要被保护的应用程序在系统初始化前被解耦和为包含关键数据和相关代码的关键部分, 和其余不受影响的正常部分; 2) 在应用初始化阶段, 应用程序会首先与 Sedora 建立安全通信, 之后与安全运行环境建立共享内存; 3) Sedora 会将关键部分加载到安全运行环境中, 关键部分将作为单独的调度单位在需要的时候被调度; 4) 应用程序正常部分通过安全调用, 利用共享内存向关键部分发送请求, 而关键部分则负责执行关键代码并将结果通过共享内存返回给正常部分, 返回数据的完整性由 Sedora 进行保证。

应用解耦和: 应用解耦和的目的在于找到所有与关键数据相关的代码。仅仅将密钥或隐私信息本身放入隔离的安全运行环境是不够的, 密钥会被使用或转存成为其他数据形式从而被带出安全运行环境, 因此 Sedora 需要将所有与关键数据相关的部分全部放入安全运行环境。

本文通过 CIL[13]静态污点追踪分析技术, 找到所有与关键数据相关的代码部分, 并借此分离出与关键数据相关的代码从而对其进行隔离保护。

安全通信: 由于应用程序与虚拟机监控器之间需要进行必要的通信, 因此 Sedora 会在应用初始化阶段与应用程序建立轻量级安全通信机制。本文利用虚拟化场景下 CPUID 会直接从用户态下陷到 VMM 的特点来绕过 Guest OS, 在满足 CPUID 的调用条件下, 应用程序可以通过寄存器直接获取 VMM 的反馈。与安全运行环境的通信则由共享内存负责, 两者一起为正常部分提供安全调用功能, 最终完成对关键数据的保护。

安全运行环境: 安全运行环境是保护关键数据的核心之一, 其本身基于 VMM 内存隔离机制运行在另一个地址空间中, 从而达到与原 Guest VM 的相互隔离。为了减少攻击面, 安全运行环境相对封闭, 仅能通过共享内存与外部进行通信, 并采用直连方式使用专有磁盘来存储关键数据。为了抵御来自攻击者的物理攻击, Sedora 利用 AMD 的 SEV 和 SME 机制对安全运行环境进行加密, 保证内存中的关键数据和代码在运行时均为密文。

数据处理与安全控制分离: Sedora 将应用程序的正常数据与关键数据进行了分离, 同时保持原有的数据处理流不变, 仅在关键数据和正常数据的边界, 即共享内存部分进行完整性校验。校验结果会通过轻量级通信机制传送给运行在 Guest OS 的应用程序, 最终保证从安全运行环境中返回的结果不会被 Guest OS 恶意篡改, 在保护关键数据不泄露的同时保证应用程序的正常执行。本章节将详细介绍使用的相关技术设计和实现。

3.2 应用解耦和

图 3 展示了应用解耦和的过程, 在进行应用解耦和时, 用户需要定义关键数据的范围, 如隐私数据的定义等。本文利用 CIL 静态代码分析工具来追踪程序的数据流, 找到关键数据的传递路径并对相关联的数据进行污染标记, 同时找出所有读取过污染标记的代码。CIL 本身具有强大的功能, 包含宏观的程序结构信息与具体的控制流信息, 由于本文旨在进行较为粗粒度的解耦和, 因此只需找出被标记数

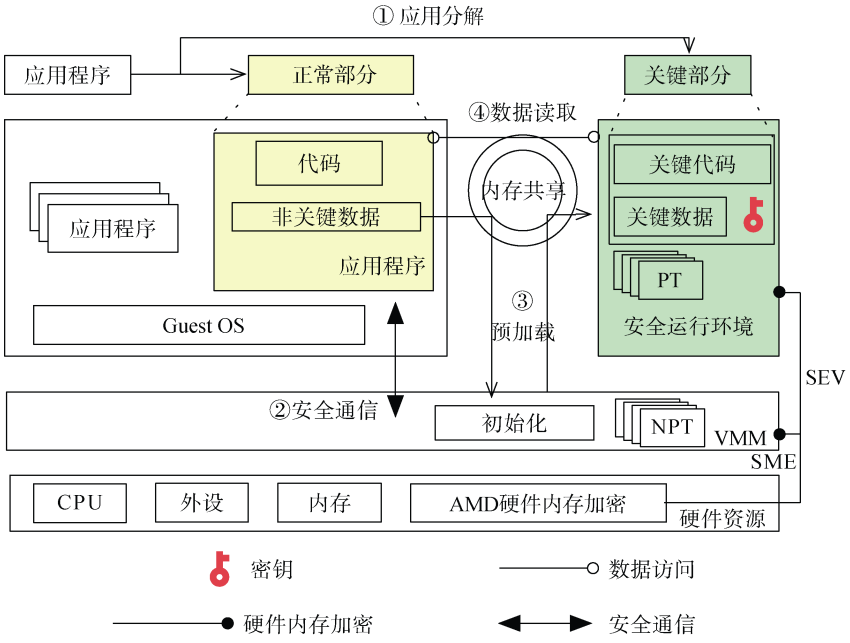

图 2　Sedora 系统架构

Figure 2　Architecture overview of Sedora

据相关的程序部分进行分离即可。而 CIL 本身的抽象语法树、程序转译等功能则被本文抛弃使用。

图中①根据用户定义, 应用程序中的关键数据被进行了标记, 需要注意的是, 标记可能不止一处, 但所有被标记的数据均被认为是污染源。在②的过程中, 本文使用 CIL 对代码进行静态分析, 传播标记, 找出所有与关键数据相关的代码和其他数据。需要注意的是静态分析的结果会得到所有可能的路径, 因此被标记的代码和数据相较于动态分析会比较多, 但是本文建立的安全运行环境与原有的正常环境完全隔离, 且安全执行环境本身是个封闭空间, 因此即使将大部分代码作为关键部分置入安全运行环境, 在面对恶意的 Guest OS 时 Sedora 仍能保证关键数据及相关代码的安全。动态分析虽然可以更高效, 但是无法覆盖所有可能路径, 因此会留下些许隐患, 比如与关键数据相关的代码可能被遗留在正常部分。同时, 本文发现应用程序对于关键数据(如密钥)的访问往往具有固定的模式, 这也表明静态分析足以满足本文对应用程序解耦和的需要。

在得到分析结果后, 所有被标记的代码和数据均被划分为关键部分, 而其余代码和数据则被划归为正常部分。在分析过程中, 本文发现对于关键数据的使用一般有两种情况: 1) 关键部分的代码与数据最终会返回一个与关键数据本身无关的返回值。这种情况一般发生在验证过程, 比如用户登陆时的密码验证, 验证过程作为关键部分被分离, 最终关键部分代码会返回一个成功或失败的结果, 在这种情况下关键部分可以与正常部分很好的分离。2) 正常部分利用关键部分进行数据加工, 并最终要求返回加工后的数据。这种情况一般发生在加密过程, 正常部分将明文传送给关键部分, 而后要求返回密文。由于密文在与关键数据(如密钥)作用的过程中会被标记从而划归为关键部分, 而安全运行环境由于自身封闭, 因此最终无法返回给正常部分。在这种情况下, 应用程序应当提前给出最终返回结果的定义, Sedora 会将关键部分中满足用户定义的结果进行返回, 从而满足应用程序对此类情况的需求。

在成功对应用程序进行划分后, 关键部分会在系统初始化时被预加载到安全运行环境中, 并被 Sedora 保护起来。如图 2 所示, 被预加载的应用程序关键部分与正常部分完全分离, 通过共享内存进行通信。本文将应用程序的关键数据部分替换为安全调用。如图 3 (4)中所示, 原有的应用程序在执行到需要访问安全数据的部分会使用安全调用, 并将参数通过共享内存传递给关键部分, 安全运行环境中的关键部分负责处理来自正常部分的请求, 在执行完成后将结果存入共享内存。由于共享内存会被 Guest OS 截取并可能被恶意修改, 因此在关键部分执行完成返回时, Sedora 会对共享内存中的数据进行完整性校验, 当控制流回到正常部分的安全调用时, 安全调用会通过 Sedora 实现的轻量级安全通信, 利用 CPUID 绕过 Guest OS 取回校验码, 同时完成对共享内存中结果的完整性检查, 以此来保证结果没有被恶意的 Guest OS 修改。

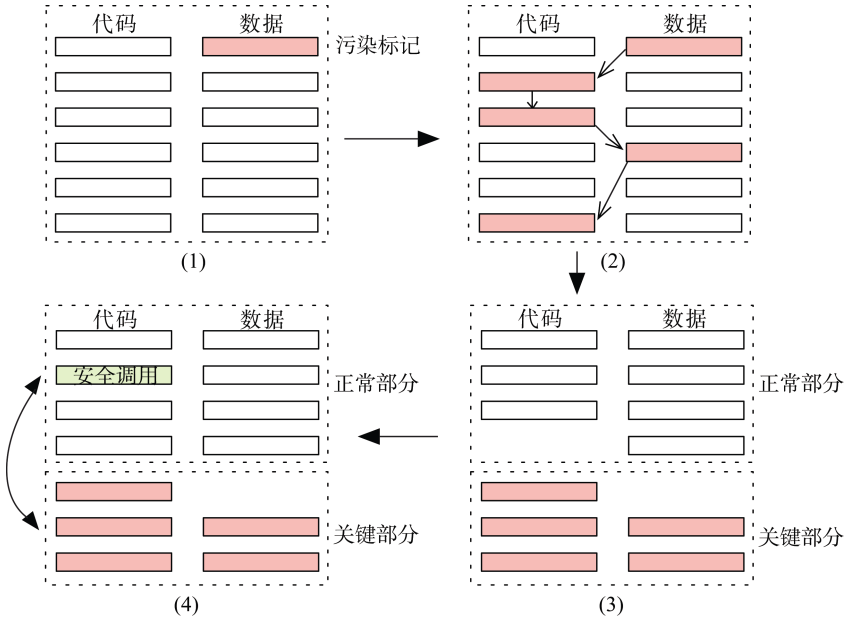

图 3　应用解耦和过程

Figure 3　Decomposition of application

3.3　安全通信

安全通信机制是实现安全调用的核心, 分为轻量级安全通信和共享内存通信。轻量级通信是 Sedora 为了绕过 Guest OS 与应用程序直接通信的机制; 而共享内存通信则是由 Sedora 为了满足关键部分和正常部分通信而实现的 VM 间通信, 共享内存通信的完整性安全又由轻量级安全通信保证。

3.3.1　轻量级安全通信

轻量级安全通信的实现依赖于虚拟化场景下特权指令的下陷模拟。由于 Xen 负责管理所有的硬件资源, 因此当 Guest OS 需要进行硬件相关处理, 如 CPUID、硬件中断等场景就需要下陷到 VMM 中由 VMM 负责处理。CPUID 作为一条特殊指令, 当 Guest VM 执行时, CPU 会产生 VMEXIT 并下陷到 VMM 中, VMM 会根据下陷原因取出 EAX 中的 ID 并实际完成 Guest VM 所需的工作, 最终将结果返回。

Sedora 选择 CPUID 作为轻量级安全通信具体实现的原因在于 CPUID 可以在用户态被调用。由于应用程序往往会根据 CPU 的特性从而产生不同的行为, 因此为了便于应用程序开发者, 从 Guest OS 的视角来看, CPUID 是一条非特权指令, 可以被应用程序直接调用。但是 CPUID 由于与 CPU 硬件的特性相关, Guest OS 无法直接返回这样的信息, 为了实现的高效, CPUID 会直接下陷到 VMM 并由 VMM 来负责处理。这样一来, 应用程序便可通过 CPUID 绕过 Guest OS 直接将数据通过寄存器传递给 VMM, 而 VMM 亦可通过寄存器将数据绕过 Guest OS 直接传递给应用程序, CPUID 的这一特性能够保证通信过程中的数据安全。

在 Sedora 的实现中, 由于不同的应用程序可能会使用同一块共享内存, 存在共享内存中数据被恶意修改的可能性, 因此轻量级通信主要保证共享内存数据的完整性。Sedora 实现了新的 CPUID, 当应用程序调用该 CPUID 时, Sedora 会计算指定的共享内存中的校验码并通过寄存器返回给应用程序。在目前的实现中, Sedora 为每一个 Guest OS 分配了 16 组 CPUID 号让应用程序用于完整性检验, 其中一组 CPUID 有 *CPUID_set* 与 *CPUID_get* 两种, 分别负责计算共享内存的校验值并将结果存入如表 2 所示的校验表, 以及从校验表中获取校验码。每个应用程序会只会使用一组 CPUID 号, 不同的运行中的应用程序不能共享同一组 CPUID 号。由于 CPUID 的数据传递只能使用 EAX 到 EDX 四个 32 位寄存器, 因此根据校验码的实现不同, 应用程序可能需要调用不止一次 CPUID 以获取所有的校验码。考虑到 MD5 校验码长度恰好为 128 位, 能够在一次 CPUID 中传输完毕, 且 MD5 的使用非常普及, 表现出相对强的可靠性, 因此 Sedora 将其作为目前的默认校验码。

正常部分与关键部分需要传输大量数据时, 作为轻量级安全通信实现的 CPUID 无法满足需求, 因此 Sedora 提供了与安全运行环境间的共享内存通信作为大量数据的传输方式。

3.3.2　共享内存通信

共享内存的实现是为了满足虚拟机间的通信需

要, 在 Xen 中共享内存使用最多的情况属于半虚拟化 I/O。Guest VM 中会运行一个前端驱动, 负责处理来自 Guest VM 自身的 I/O 请求。而 Domain0 中在创建 Guest VM 时会创建对应的后端驱动, 并初始化与前端内存的共享内存。当前端驱动需要读写数据时, 会首先将数据写入共享内存, 之后通过 Xen 中的 Event 通知后端驱动将共享内存中的数据写入实际的设备中。

Sedora 实现了类似的通信机制, 由于轻量级安全通信能传输的数据量十分有限, 因此当需要传递大量数据时, 安全运行环境会通过共享内存与 Guest VM 中的应用程序通信。如图 6, 当应用程序在 1 步调用 *CPUID_test* 并根据返回值确定 Sedora 被正确安装后, 应用程序会通过已有的接口 *shmget* 和 *shmat* 请求 Guest OS 帮助其建立共享内存。第 2 步 *shmget* 会使 Guest OS 分配一个页而第 3 步 *shmat* 会使 Guest OS 在应用程序的地址空间中映射这个页, 之后通过调用 Sedora 新实现的 syscall *sys_setenv* 并传入共享页号和对应权限后, Guest OS 会在第 5 步通过 hypercall *hypervisor_grant_op* 将该页共享给安全运行环境, 根据需要, 应用程序可以分别建立只读与可写的共享页。Sedora 在处理 *hypervisor_grant_op* 时, 通过查询授权表获取共享页的权限, 并修改安全运行环境的 NPT 对该共享页进行映射, 同时 Sedora 会将该共享内存与该 CPUID 号进行绑定, 记录在校验表中。如表 2 所示, CPUID 3 就被绑定了对应的共享页。至此, 安全运行环境就与 Guest VM 中的应用程序建立了共享内存通信机制。

虽然共享内存中的数据不属于关键数据, 因此不用担心其机密性的问题。然而, 由于建立共享内存的步骤依赖 Guest OS, 并且在之后的数据传输过程中 Guest OS 随时可以进行介入, 所以如何保证其完整性则是 Sedora 需要考虑的问题。如图 6 所示, 在建立了共享内存后, 应用程序需要如第 7 步向共享内存中写入随机测试数据, 并在第 8 步使用绑定的 *CPUID_set* 下陷到 VMM 中。Sedora 会对共享内存中的数据进行 MD5 校验, 并将校验码通过通过寄存器返回给应用程序。最后, 应用程序会将 Sedora 计算的校验码与自己所得的校验码进行对比以确保共享内存确实被建立。共享内存同样是程序运行期间正常部分与安全部分通信的唯一方式。

需要注意的是, 在调用 CPUID 后, Sedora 会计算指定的共享内存的校验码, 因此每个应用程序只能在初始化时建立与安全运行环境的共享内存, 当共享内存一但被建立, Sedora 会将其与该应用程序所使用的 CPUID 组号进行绑定, 如表 2 所示。在之后的完整性校验中, Sedora 会根据 CPUID 号直接计算与之绑定的共享内存的校验信息。

表 2 校验表

Table 2 Checksum table

Guest VM ASID		
CPUID 1	共享页号#n1	MD5 校验码
CPUID 2	无	无
CPUID 3	共享页号#n2	无
…	…	…

3.3.3 安全调用

安全调用是 Sedora 使用安全通信机制对应用程序提供的封装。由于应用程序被解耦和为正常部分和关键部分, 为了将两部分联系起来, Sedora 会在原来的应用程序中插入安全调用。

安全调用的过程如图 4, 当应用需要运行关键代码时, 首先将参数写入共享内存, 之后调用与应用程序绑定的 *CPUID_set* 下陷到 Sedora 中, Sedora 会校验共享内存中的数据并将校验值填到校验表中, 同时返回结果给应用程序检查。应用程序确认参数无误后便可发送 Event 以通知安全运行环境。在第 6 步结束后, 安全运行环境被调度, 关键代码会从共享内存中读取数据, 并通过 *CPUID_get* 获取校验表中的校验码进行完整性校验。当校验通过后, 关键代码会开始运行并将最终结果写入共享内存, 之后的过程与正常部分的过程一样。安全调用执行完毕后会将共享内存的数据返回给应用程序的正常部分。

安全调用的主要目的在于利于安全通信保证共享内存的中的数据完整性并隐藏其实现细节。使用者不需要关注下层的具体实现, 只需将所定义的关键数据和污染分析所得的关键部分作为参数交由关键调用即可。

3.4 安全运行环境

安全运行环境是一个被保护的虚拟机, 用以支持其上运行的安全部分。如图 5 所示, 安全运行环境作为特殊的虚拟机运行在 Sedora 之上, 其中运行了一个必要的 OS。需要注意的是, 在 Sedora 的实现中为了简化工作量将一个成熟的 OS 作为安全环境的 OS 来运行, 但是实际情况中根据需要可以运行一个仅仅包含支持库的运行时, 以减小安全环境的资源占用。由于安全运行环境本身仅在被调度时运行, 且关键部分代码相对较小, 因此不会占用过多资源。

安全环境自身通过配置 PT 运行在 SEV 的保护之下, 所有内存中的数据均为密文从而保证关键数

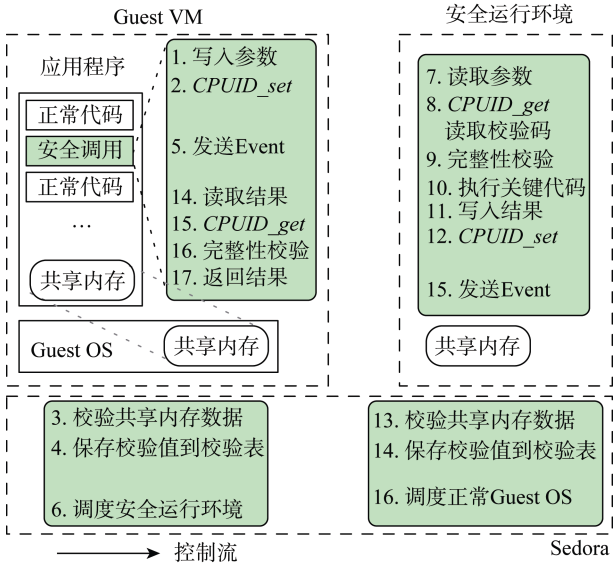

图 4　安全调用过程

Figure 4　Secure call procedure

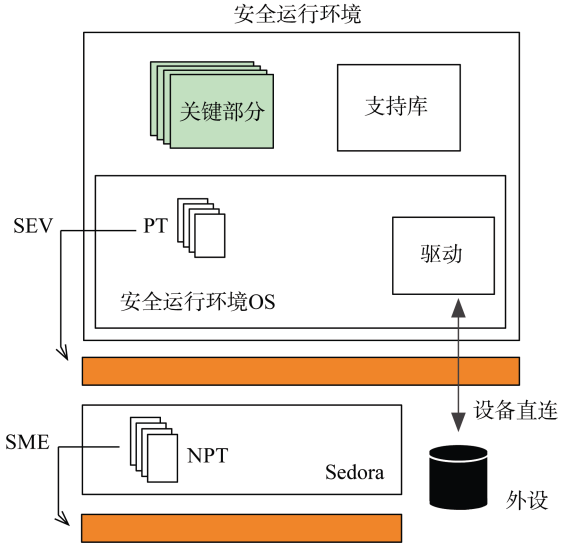

图 5　安全运行环境

Figure 5　Secure runtime environment

据的安全。同时, Sedora 自身会通过 NPT 和自身的 PT 开启 SME 保护, 以防止攻击者通过物理攻击读取 NPT 中的数据进而破坏安全运行环境的内存隔离。SEV 与 SME 由于对应用层透明, 因此关键部分代码不会收到任何影响。

安全运行环境自身运行在一个独立的地址空间, Sedora 通过 NPT 保证安全运行环境与其余 Guest VM 的隔离。运行环境自身相对封闭, 仅能通过在 Sedora 控制下的共享内存与外界通信, 并且关键部分的预加载也需要经过 Sedora 的校验。根据安全需要, Sedora 可以实施更进一步的安全扩展, 在预加载的校验阶段拦截执行流, 并执行扩展的安全检查以保证关键部分代码不是恶意的。正常 Guest VM 的 I/O 操作由 3.3.2 中所述, 需要使用 Xen 自身的半虚拟化 I/O, 但这样会涉及与 Domain0 的交互从而会打破安全运行环境的封闭性。为了解决这个问题, Sedora 采用设备直连的方式将专有的存储外设直接连接到安全运行环境, 在其中可以存放安全运行环境 OS 镜像、对于关键部分的支持库等。在特殊条件下, 可以通过配置, 将部分应用程序数据如加密密钥等存放于专有外部存储中。

在目前 Sedora 的实现中, 一个安全运行环境只与一个 Guest VM 相对应, 由于 SEV 的限制, 系统目前最多仅支持 16 个 Guest VM 同时使用该保护。为了支持关键部分的执行, 在预加载时支持库会与关键部分一起被加载, 在相应的初始化完成之后安全环境会进入阻塞态, 释放 CPU 资源并等待来自应用程序的安全调用。最终, 应用程序的关键数据得以安全的运行其中。

3.5　数据处理与安全控制分离

Sedora 的设计目的在于保护应用程序的关键数据安全。如果每次正常部分与关键部分进行交互时 Sedora 都介入检查, 频繁下陷会带来极大的性能开销。因此, Sedora 采用了安全控制与数据处理相分离的思想, 将主要的数据共享通过共享内存交由应用程序自身完成, 数据传输过程中不需要 Sedora 的介入。而在数据传输的边界(开始传输和结束传输), Sedora 对传输的数据进行完整性检查, 并将校验码存放在校验表中以备应用程序检查。整个传输过程与传统的数据传输并无二致, 依旧在 Guest OS 的管理之下, 但是 Sedora 通过轻量级安全通信和完整性检查保证了数据不会被恶意的 Guest OS 所篡改。

同时, Sedora 使用安全调用的方式将关键数据和代码均放到了安全运行环境中, 安全控制部分交由安全运行环境执行, 应用程序原有的处理逻辑保持不变, 这样在正常数据处理中 Sedora 和安全运行环境不需要进行任何干预, 也减少了相应的开销和应用程序开发者的解耦和工作。受安全检查影响而带来的少量开销也只会影响占比较小的关键部分。

4　应用场景

表 3 表示的是 Sedora 新定义的 3 种新 CPUID, 1 个新的 syscall 与 1 个新的 hypercall, 通过调用这些指令 Sedora 可以为关键数据提供安全保护。整个应用程序的生命周期如图 6 所示, 当含有保护需求的应用程序所在 Guest VM 启动时, Sedora 便一并创建对应的安全运行环境。在这个过程中, Sedora 主要负责开启安全运行环境的 SEV 保护, 使用 NPT 将其隔离

表 3　Sedora 中定义的 CPUID、syscall 与 hypercall

Table 3　New designed CPUID, syscall and hypercall in Sedora

名称	说明
CPUID_set_n	与 Guest VM ASID 绑定的 *CPUID_set*, 用于计算共享内存校验码并写入校验表并将校验码返回
CPUID_get_n	与 Guest VM ASID 绑定的 *CPUID_get*, 用于读取对应校验表中的校验码
CPUID_test	用于检测当前平台是否为 Sedora, Sedora 会返回一个固定签名值
syscall_set_secenv	请求 Guest OS 为自己配置共享页
hypervisor_grant_op(bind, ...)	在校验表中绑定共享页与对应的 CPUID

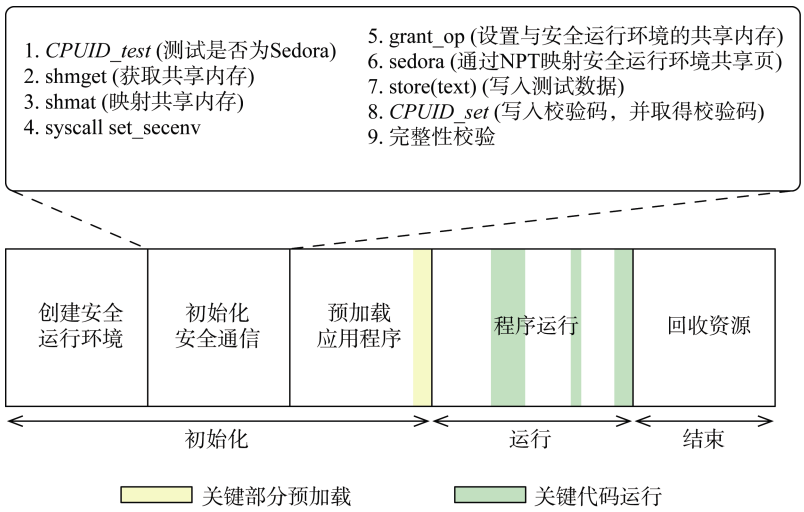

图 6　应用程序生命周期

Figure 6　Lifecycle of protected application

并将专有存储直接连接到环境中。在应用程序预加载过程中, 由 Guest OS 辅助完成共享内存的建立, 并将关键部分传给安全运行环境。在预加载的过程中应用程序会利用安全通信检查系统是否为 Sedora 并完成自检, 之后在正常运行中使用安全调用完成功能。最后当应用程序结束运行时, 由 Sedora 回收安全环境中的资源; 当 Guest VM 停止运行时, Sedora 也会一并撤回安全运行环境。

4.1　利用 Sedora 保护 OpenSSH

OpenSSH 使用 RSA 加密算法进行验证, 在 OpenSSH 服务器的验证阶段, 客户端使用服务器的公钥来加密数据并使用自己的私钥对密文进行签名, 服务器端使用客户端的公钥来验证客户端身份, 并使用自己的私钥来解密获取会话密钥。具体实现为两个大质数 p 和 q 会被用于生成两个值与密钥相关的值 e 和 d, 而 p 和 q 的乘积 N 会被公开($N = p \times q$)。当一个客户端连接到服务器端时, 服务器端会使用私钥*(d, N)*来加密数据, 客户端则可以使用对应的公钥*(e, N)*来解密数据。由于 N 和质数的长度是公开的, 因此在一个恶意的 Guest OS 场景下(如 Guest OS 被安装了 rootkit), 当攻击者想获取私钥时, 只需要在内存中遍历搜索这个长度的数据, 然后看是否能整除 N 即可快速找出该质数, 当找到其中一个质数时, 另一个只需通过整除 N 即可获得, 之后便可自行生成 e 和 d 从而获取密钥。

本文使用的 OpenSSH 为 openssh-portable-7.5p1, OpenSSH 会将指数质数 d 存放在一个 *BIGNUM* 数据类型中的指针 d 中, 生成密钥的上下文则由一个 *BN_CTX* 数据结构维护。在这个案例中, 我们定义验证过程中所使用的私钥为关键数据, 通过应用解耦和, 使用私钥进行验证的过程放入安全运行环境中。

我们通过手动安装 rootkit 来模拟 Guest OS 被攻击从而变为恶意的场景, 攻击者借由 rootkit 达到提权并能够读取 Guest OS 的所有内存。由于服务器端的私钥验证过程都在安全运行环境, 因此攻击者无法通过扫描 Guest OS 的内存来找到密钥。如果攻击者能够物理接触服务器并实施 Cold Boot 或 Bus Snooping 等物理攻击, 由于内存中的数据均是加过密的, 攻击者依旧无法读取其中的数据。

4.2　利用 Sedora 保护 GnuPG

GnuPG 是一款加密软件, 其作用是加密文件以保护文件中的数据隐私。在此例中 GnuPG 使用 AES

进行加密, AES 是一种对称加密算法, 与公钥加密系统不同的是, 对称加密加密和解密使用的是同样的密钥, 一旦丢失那么攻击者可以利用密钥将密文全部解密。AES 加密的加密过程一般是将明文 t 通过一系列与密钥 s 的变换(异或、移位等), 最终生成密文 e。AES 的加密模式有多种, 其中 ECB 密码本模式最为简单, CTR 计数器模式较为可靠, 因此我们使用这种加密模式来进行测试。通常使用 GnuPG 是为了保证 I/O 的数据是密文从而防止攻击者读取 I/O 中属于用户自己的隐私数据, 但是 GnuPG 加密所使用的密钥存放在内存中, 因此当攻击者获取密钥后不但可以读取到本次加密的所有数据, 所有使用该密钥加密的文本均可以被攻击者所读取。

本文使用的 GnuPG 为 2.0 版本, 我们定义 GnuPG 加密所使用的密钥 s 为关键数据, 通过应用解耦和, 使用密钥进行加密的代码和密钥本身均被加载到安全运行环境中。此案例中对应于 3.2 中所描述的情况 2), 正常部分要求关键部分返回加工后的数据, 因此我们定义生成的密文为关键部分所返回的结果。当 Guest VM 在第一次运行 GnuPG 时, 通过调用在安全运行环境中的关键部分代码生成密钥, 并将密钥存储于专有外设中。通过手动安装 Rootkit, 我们模拟了攻击者成功攻陷 Guest OS 的情况, 由于密钥存储于安全环境中, 因此攻击者无法读取密钥的值。即使攻击者能够趁机读取部分明文, 但是已经被加密的数据则无法读取, 而在这段时间内, 应用程序拥有足够的时间来检测出攻击的发生并实施相应的对抗措施。

当正常部分需要加密数据时, 明文会通过共享内存被传给安全部分进行加密, 而后再将密文通过共享内存传回给正常部分。但是这么做会产生大量的数据拷贝, 从而导致过高的开销。所幸, 在 Xen 的半虚拟化 I/O 实现中, Guest VM 会将数据通过共享内存共享给 Domain0, 这部分在 Guest VM 创建时便被建立并且在 VM 运行期间保持不变, 这恰好与 Sedora 所要求的关键部分与正常部分的共享内存特性一致, 因此通过对半虚拟化 I/O 中的前端进行少量修改, 我们可以让应用程序所使用的共享内存与前端共享内存保持一致。当应用程序需要从通过 I/O 读取明文进行加密并再写回磁盘时, 可以通过这种复用共享的方式减少数据拷贝从而减少性能开销。

5 安全分析

抵御物理攻击: 硬件加密内存所带来的最大好处就是程序运行时内存中的数据均为密文, 应用程序可以利用这一特性来抵御如 bus snooping 和 cold-boot 攻击。这种基于硬件的攻击一般假设攻击者可以直接导出内存数据从而进行离线分析, 而在 SEV 和 SMED 的保护下, 攻击者能看到的只有密文。对于如 Rowhammer[15]这类 bit-flip 攻击, 由于内存中数据的实际排布经过加密后发生了变化, 因此获得物理上相邻的内存地址变得不再简单, 而精确改变某一位则更为困难。

关键数据机密性: 一个恶意的攻击者想要读取如密钥这类关键数据, 只能使用映射关键数据所在的物理页, 从而能够从内存中直接读取关键数据、通过网络远程攻击或使用物理攻击直接读取数据这几类方法。Sedora 通过 NPT 实现了安全运行环境的内存隔离, 攻击者无法直接映射 Sedora 所使用的物理内存从而无法对其进行映射; 安全运行环境本身封闭, 对外交互只能通过共享内存, 这使得攻击者无法通过网络等方式远程干涉关键代码的执行; 攻击者使用物理攻击则能够被 Sedora 直接抵御。通过这些方法, 关键数据的机密性可以得到保证。

共享内存完整性: 由于在 Xen 中, 共享内存的建立只在 Guest OS 的内核态, 而共享内存到应用程序之间的联系则完全由内核来负责。因而, 攻击者可能会利用漏洞达到对共享内存的映射, 从而冒充被保护程序来破坏其传递的参数或破坏其所得结果, 进而破坏程序的正确执行流程。Sedora 通过安全通信机制, 对共享内存的每次通信都进行完整性检查, 并利用轻量级安全通信绕过 Guest OS 将检查结果交给应用程序, 以此来避免与关键代码相关的结果和参数被恶意修改。

关闭保护: Sedora 提供的保护依赖于对 CPUID 的模拟和 AMD 硬件内存加密的开启。我们的安全调用过程不涉及到功能开启、关闭和更改, 因此攻击者无法在使用安全调用的过程中关闭保护机制。对于 CPUID 的模拟全部发生在 VMM Sedora 中, 因此 Guest OS 无法介入修改; 硬件内存加密的开启和关闭则只能由 VMM 在 host 模式下调用特权指令完成, 因此攻击者无法关闭加密保护; 而加密页的管理则是由 VMM 和安全运行环境自行维护, 攻击者无法介入管理。

I/O 数据窃取: 在原有系统中, Guest VM 的密钥需要通过 I/O 存放在磁盘中, 攻击者可以在之后读取磁盘进行窃取。Sedora 将需要离线保存的数据均存放于与安全运行环境绑定的专有外设中, 而应用程序运行过程中密钥只在安全运行环境中被使用, 因此攻击者无法通过 I/O 窃取作为关键数据的密钥。

6 性能评估

6.1 测试环境

本文测试所使用的 CPU 为 8 核(16 线程) Ryzen 1700X, 频率为 3.4GHz, 内存为 8G, 虚拟机监控器版本为 Xen 4.5.1, Domain0、安全运行环境与 Guest VM 所使用的 Linux 内核版本为 4.10.2, 每个 Guest VM 均被分配两个虚拟 CPU 与 2G 内存。

6.2 应用性能评估

对于 OpenSSH, 本文使用了一个脚本在同一个客户端中连续使用 ssh 客户端向运行在 Guest VM 中的服务器端发送验证请求 100, 000 次并对比了服务器端正常运行与应用 Sedora 方案后的运行情况。为了降低网络延迟带来的误差, 测试时选择的客户端为同一物理机的 Domain0, 测试使用的 OpenSSH 版本为 7.5p1, 结果显示在 Sedora 上运行的服务器端延迟增加了 8.68%。

对于 GnuPG, 本文将加密部分放入安全运行环境并进行大量的文件读写加解密, 测试使用了修改后的前端驱动以减少数据拷贝带来的性能开销。结果如表 4 所示, 程序执行的速度与文件在磁盘中的分布有关。当文件为顺序分布时, 程序执行时的瓶颈在于加解密所花的时间, 其中解密的开销为 17.59%, 加密的开销为 6.71%。经过分析, 本文认为造成解密开销远高于加密的原因在于加密后的数据可以首先写入缓冲区, 之后批量写入磁盘, 而解密所需的数据测必须每次从磁盘上读取, 因而无法如加密写入那样大批量执行。而当文件为离散分布时, 可以发现性能的主要的开销在于不在于软件而在于磁盘自身的调度, 在这种情况下 Sedora 所带来的开销仅有解密时的 2.32%与加密时的 0.91%。

表 4 GnuPG 性能测试

Table 4 Performance evaluation of GnuPG

操作	顺序读 (解密)	顺序写 (加密)	随机读 (解密)	随机写 (加密)
性能开销/%	17.59	6.71	2.32	0.91

6.3 系统性能评估

Sedora 自身由于包含安全调用时的完整性验证、参数拷贝与安全环境调度等开销, 同时由于使用了 AMD 的 SEV 技术, 因此本文将评估系统本身软件方面与 SEV 硬件加密分别带来的开销。本文分别在运行普通部分的 Guest VM 与运行关键部分的安全运行环境 (Secure Env) 中运行了 SPECCPU 2006 与 PARSEC 3.0 包含的基准测试程序, 以未修改的 Xen 4.5.1 环境下的相同 Guest VM 中的运行结果作为基准, 从而评估 Sedora 系统软件本身与 SEV 硬件加密分别造成的性能开销。其中 SPECCPU 2006 主要用来测试 CPU 单个线程的计算能力; PARSEC 3.0 则是综合测试, 包含多线程与大量内存读写的测试程序。

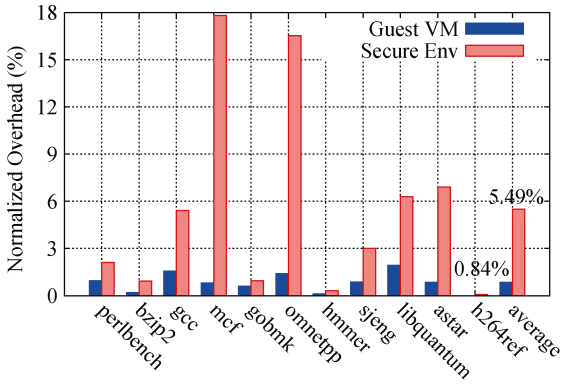

图 7 性能测试 SPECCPU2006

Figure 7 Evaluation of SPECCPU2006 benchmark

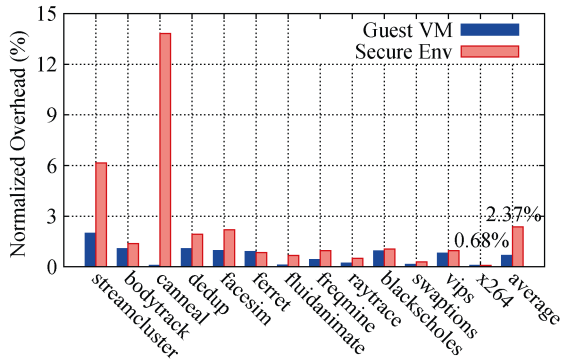

图 8 性能测试 PARSEC 3.0

Figure 8 Evaluation of PARSEC 3.0 benchmark

图 7 显示了 SPECCPU2006 测试集的运行结果, Sedora 系统本身所带来的性能开销平均不到 1%, 在误差范围内几乎可以将其忽略。安全运行环境的平均性能开销有 5.49%, 其结果是可以接受的。其中如 CPU 密集型的程序如 hmmer、h264ref 等几乎没有性能开销, 而造成性能降低最多的 mcf (17.82%)与 omnetpp(16.52%)均因其存在大量内存读写且测试数据在内存中分布较为离散。被访问数据分散会造成缓存利用率低, CPU 需要频繁访问内存以获取数据, 由于安全运行环境的内存读写需要经过加解密, 因此这样的访存中硬件加解密会是主要瓶颈。

图 8 显示了 PARSEC3.0 测试集的运行结果, Sedora 系统本身的软件性能开销只有 0.68%, 对于安全运行环境, 大部分情况下几乎没有性能开销, 而总的性能开销也仅有 2.37%。其中 canneal 测试的性能降低则也是由大量离散访存中内存加解密造成。

6.4 性能分析

为了分析影响性能的因素, 本文还进行了一些相关指令及过程的微测试以分析造成性能下降的原因。其中 CPUID、MD5 校验、安全运行环境调度均在 VM 中执行了 100, 000 次并取平均值作为结果。安全运行环境的调度由于实际为一个时钟中断产生的 CPU 调度, 这与一个 Hypercall 的调用过程类似, 因此本文测量了一个空 Hypercall 所需的 Cycle 数作为替代。对于 MD5 校验, 本文以 256B 的数据校验作为基准。而对于硬件内存加密的访存开销, 本文通过拷贝连续 256M 的数据并取一个 4K 页的内存拷贝作为基准。

表 5 指令/过程微测试

Table 5 Instruction/Procedure micro-benchmark

指令/过程	CPUID	MD5 校验	安全运行环境调度	加密内存访存(4K)
Cycle 数	134	4908	787	1989

测试的结果如表 5 所示, 本文的安全调用会涉及 4 次 CPUID 的调用与至少 2 次的 MD5 校验检查和 2 次环境调度, 因此每次调度所需的 CPU 开销约为 12000 个 cycle, 而根据与安全运行环境共享页大小的不同, 校验所需的时间也会不同, 因此应用程序应当根据自己需要选择合适大小的共享页以提高系统的性能。

而运行在安全运行环境中程序的访存, 每次访问一个物理页大约需要 1989 个 cycle, 与没有加密时的情况相比, 其开销为 8.70%。因此为了提高性能, 一方面, 应用程序在解耦和时应谨慎划分关键部分, 减小安全运行环境中的数据和代码; 另一方面, 减少安全运行环境中的内存访问并提高缓存的命中率, 这样可以尽可能避免由内存加密所带来的性能下降。

7 相关工作

7.1 硬件内存加密

安全处理器过去的几年里在学术界已经被广泛研究和讨论, AEGIS[16]提出了一种单芯片处理器, 该处理器能提供一个防止修改、能够被验证的安全环境。XOMOS[17]利用其提出的安全处理器架构 XOM[17], 在现有的操作系统中支持共享库、进程间通信等等。基于硬件内存加密, 研究者们也不停改进内存加密与完整性验证。Brian 等人[19]利用与地址无关的种子加密(Address Independent Seed Encryption, AISE)来进一步优化计数加密模式并结合 Bonsai 默克尔树, 通过保护交换内存来提高基于默克尔树的内存完整性验证的性能和安全。基于安全处理器架构, 相应的出现了保护关键安全应用程序不受不可信操作系统和物理攻击影响的 Bastion[20]与 SecureME[21], 其中 SecureME 还结合了如内存隐身、系统调用保护等手段。HyperCoffer[22-23]则首先提出利用安全加密内存器, 在不可信的虚拟机监控器上运行虚拟机, 它同样基于高效的内存加密机制 AISE 和完整性验证技术 BMT, 通过在虚拟机中引入 shim 层来解决 VM 与 VMM 之间的语义断层, 最终能够支持不加修改的、包含商业软件在内的虚拟化技术。

在学术界对于硬件内存加密如火如荼地研究推动下, 各主流处理器厂商也推出了相应的具有内存加密支持的产品。Intel 推出了 SGX, 一种对 CPU 的硬件扩展和一系列在用户态可以使用的新指令, 它能够让应用程序在用户态直接申请被称为“飞地”的私有内存。其启动时, CPU 内生成加密密钥并用其来加密飞地内的数据, 数据仅能被创建它的应用程序自身访问, 即使是权限级更高的操作系统也无法进行读取和修改。基于 SGX, 有大量的工作来用其保护软件的安全[24, 29], 或者提高 SGX 本身的安全[29-30]。SCONE[24]利用 SGX 来提高容器机制存在的一些局限性; Haven[25]能够将未修改的应用程序的代码和数据置于 SGX 中, 以此来提防脆弱的操作系统; M2R[26]和 VC3[27]都使用了 SGX 来增强 MapReduce 框架从而能够保护分布式计算中隐私数据的安全; Ryoan[28]利用 SGX, 在面对不可信的服务提供商的情况下, 保护数据处理服务的隐私数据安全。另一方面, 为了增强 SGX 飞地本身的安全, 如 SGX-Shiled[29]被用于为运行在飞地内的应用程序提供地址空间随机化; T-SGX[30]则利用了硬件事物内存来根除针对于运行在飞地内应用程序、基于页错误的侧信道攻击。

正如本文在第 2.1 节所介绍的, AMD 也推出了具有硬件内存加密支持 SME 和 SEV 的处理器, 具有特权级的代码能够通过设置页表项中的 C-bit 来管理加密页, 虽然与已经现世几年的 SGX 相比, SME 与 SEV 还没有经过足够多的测试和验证, 支持 SEV 的对应硬件甚至还没有发售, 但是相比于 SGX 的复杂过程, SME 与 SEV 简单易用, 对上层透明的特点依旧具有极强的吸引力。Sedora 是首先提出利用 SEV 与 SME 来保护应用程序在不可信 Guest OS 中数据安全的解决方案, 并利用硬件加密机制测试了相应的性能开销。

7.2 应用数据保护

对于应用关键数据保护, 学术界已有诸多研究。SeCage[7]利用 Intel 推出的 VMFUNC 指令, 对 kvm 进

行安全扩展, 达到高效的敏感数据隔离, 但是 AMD 却没有 VMFUNC 的支持, 甚至实现嵌套页表的方法与 Intel 也有非常大的差异; Mimosa[31]将使用密钥的过程置于硬件事务内存中, 利用攻击者恶意读取密钥时产生的事务中断来发现攻击者; Shred[32]通过修改编译器并结合内核模块, 建立新的程序原语, 保证一个 Shred 内的内存仅能被 Shred 内的代码访问; DieHarder[33]提出了一个面向安全的内存分配器来抵御基于堆的内存攻击而 CRYPTON[34]则设计了一个新的数据抽象和读取原语来保证数据的隔离, 但是这两者都要依赖操作系统本身的庞大的库。

其他的一些软件的加密手段则是尝试将加密密钥仅存放在 CPU 中并通过加密来保证应用数据的机密性。Safekeeping[35]利用了 x86 的 SSE MMX 寄存器来保证密钥自始至终只存在于寄存器中, 在保证高效加密的同时也避免了密钥的泄露。类似的, Copker[36]在 CPU 中实现了非对称加密, 无论明文或密钥均不会出现在内存中。这些方法的核心是利用缓存只对 CPU 可见的特点, 将加解密过程置于缓存中以达到对密钥和明文的保护, 但是这些方法在编程时需要极为小心, 并且加密过程本身仍然依赖于操作系统本身。

由于 VMM 运行在更高的特权级, 因此有许多学者尝试利用虚拟化技术来实现对应用数据保护, 比如 CHAOS[37-38]、OverShadow[8]和 Appshield [39]等系统就利用了虚拟化技术提供的内存隔离并介入被保护程序和 OS 之间的切换来应用安全策略, 达到保护应用程序的目标; HyperCrypt[40]从 Hypervisor 层面直接对整个 Guest OS 进行加密来抵御针对内存的物理攻击。与 Sedora 相比, 这些方案均具有更大的 TCB, 且内存中的数据依旧为明文, 因此无法抵御如 Bus Snooping 等物理攻击。

8 讨论与局限

部署工作量: Sedora 的部署需要应用解耦和, 因此需要应用程序的使用者定义关键数据, 并将应用程序进行分离, 虽然这一过程还不能纯自动化实现, 但是一些现有半自动化工具如 CIL 能够一定程度上简化工作。与本文工作相比, SeCage[7]仅支持仅允许有限的少量代码作为隐秘部分, 而 Sedora 支持更加粗粒度的应用解耦, 并保证运行在安全环境中的数据和代码不会受到攻击者攻击, 但是应用程序需要对关键部分大小与性能开销进行权衡。而 OverShadow[8]与 Appshield[39]等基于虚拟化技术的方案则需要对系统做更多改动。

保护范围: Sedora 被设计用于保护应用程序的关键数据如密钥、与用户相关的隐私信息等, 相关的代码和数据被划分为关键部分运行在隔离的安全运行环境中。安全运行环境本身的封闭性以及硬件加密保护保证了其中的数据不会被攻击者窃取。但 Sedora 不提供关键部分返回结果的机密性保护, 也不保护通过网络等其他方式传送的数据。

恶意应用程序调用: 由于 Sedora 的安全运行环境为虚拟机粒度, 因而存在恶意的应用程序冒充被保护应用程序的 CPUID 号来发起计算请求, 从而获取计算后的结果。然而这些结果本身不属于关键数据, 并且攻击者即使获取这些结果对其本身而言意义不大。而且, 这个问题可以通过强制每个应用程序进程号与其共享内存、CPUID 唯一绑定来解决。

关键部分支持库: 在安全运行环境中运行的代码如果需要用到其他的库, 那么用户需要在预加载时同时提供这些库。因此 Sedora 不支持关键部分对正常部分的数据访问, 这在一定程度上限制了 Sedora 的应用场景。但是好的应用设计和解耦和能够解决这一问题, 同时相对隔离的系统设计对于安全方面也是更高的保护。

安全运行环境: 由于 AMD 硬件限制, SEV 目前仅支持最多 16 个 VM, 而安全运行环境与 VM 相对应, 因此 Sedora 目前也仅支持最多 16 个安全运行环境同时运行。但是通过更改设计, 可以实现多个 VM 共享一个安全运行环境以此来提高资源的利用率, 这也是本文的后续工作之一。除此之外, 随着更新一代 CPU 的发布, 随着 SEV 支持的 VM 数量上升, 这种情况也能得到一定程度的缓解。

应用程序支持: 在目前的实现中, 应用程序通过 CPUID 进行区分, 一个 VM 支持 16 个应用程序同时使用 Sedora 提供的安全保护, 虽然本文认为同一 VM 中需要被保护的应用程序数量往往有限, 但这一限制亦可通过配置更多的 CPUID 号进行缓解。本文发现目前保留的 CPUID 非常多, 可以被大量使用, 因此在后续工作中我们将借此提供一个用户可配置的参数以支持更多 CPUID 的需求。

硬件内存加密: 由于硬件内存加密是在发生内存访问时实时进行的, 加密的复杂度会严重影响用户的使用体验, 因此目前 AMD 的硬件内存加密使用的是较为高效的 AES 加密。加密时硬件会将内存页的物理地址作为加密的相关信息纳入计算, 因此同样的数据在不同的物理页中得到的密文也不相同, 攻击者无法通过直接拷贝物理页来达到复制数据的目的。然而, 目前的硬件不支持对加密页的完整性保

护, 这意味着攻击者可以使用同一物理页进行重放攻击, 比如向服务器验证端发送同样的加密后的密码来攻击验证程序。不过, 物理内存页由虚拟机监控器管理, 攻击者不得不在获取 Guest OS 权限后进一步获取 VMM 权限才有实施这种攻击的可能, 这已超出了本文的讨论范围。

AMD 硬件内存加密的加密单位为 4K 页, 明文数据的变动仅影响该物理页对应的密文变动。因此通过频繁改写某一内存数据, 攻击者虽然无法定位明文的具体位置, 但仍可以精确到页粒度, 这在一定程度上降低了加密的安全度。不过, 与传统的内存相比, 由于任何数据的变动都会影响整个物理页的密文, 因此这在一定程度上也提高了基于内存数据位跳变的 Rowhammer[15]攻击, 和基于内存访问模式推测密钥的侧信道[14]攻击的门槛, 从而提升了系统对物理攻击的抵抗性。

目前, 硬件内存加密依赖于对 VMM 的信任, 但是 VMM 本身亦可能存在漏洞, 所以硬件内存加密的安全性仍有提升的空间。所幸, AMD 已经提出了下一代 SEV-ES, 它能够为一些重要的加密数据提供完整性保护, 并将更多的 VMM 代码排除在可信基之外, 因此本文相信硬件加密的安全性将越来越高。

对比 Intel SGX: 除了 AMD 的 SME 与 SEV 机制, Intel 在早先已经发布了安全防护扩展 SGX[9]。相比较 AMD 的硬件内存加密, SGX 能够在非虚拟化的情况下直接保护应用程序免受恶意操作系统的攻击。但是 SGX 的 API 众多, 使用繁复, 且需要应用程序重新设计编写以利用 SGX 提供的安全保护功能。在现有版本中, 大规模的应用 SGX 会对应用开发者提出过高的要求。而 SEV 与 SME 则将这部分工作转交给更加熟悉系统的系统开发人员, 从而减轻应用开发人员的工作难度。

SGX 通过通过将物理内存页标记为“飞地”并进行加密并提供完整性保护, 只有对应的应用程序可以进行访问, 以此来保证应用程序的数据安全。但是申请的内存数量有限, 因此 SGX 不能实现如 SEV 一般的全虚拟机保护, 这也反映出两者之间的侧重不同。

9 结论

本文设计并实现了一个在虚拟化环境下, 保护应用程序关键数据安全的方案 Sedora。该方案是第一个使用 AMD 硬件内存加密来保护应用程序关键数据安全的方案。通过将应用程序解耦和为正常部分与关键部分, 将关键部分部署在安全运行环境中从而其安全。该方案实现的安全运行环境相对封闭并由 AMD 的硬件内存加密机制提供保护, 通过系统实现的安全通信机制中的共享内存与正常部分进行通信, 其完整性受安全调用的保护。应用场景展示了该方案可以被应用于常见应用程序, 安全分析证明了该方案能够保护应用程序关键数据免受恶意操作系统和部分物理攻击的侵害, 性能评估则证明了该方案的可行性。

在后续工作中, 安全调用将被优化以支持更多的应用程序, 而安全运行环境则被重构为支持更多的虚拟机。对于应用解耦和过程, 后续也将考虑寻找更加自动化的工具对应用进行解耦和。

参考文献

[1] Durumeric, Zakir, et al. “The matter of heartbleed.” *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014.

[2] Ledbetter Jr, William B., and Russell A. Reininger. “Method for data bus snooping in a data processing system by selective concurrent read and invalidate cache operation.” U.S. Patent No. 5,119,485. 2 Jun. 1992.

[3] Halderman, J. Alex, et al. “Lest we remember: cold-boot attacks on encryption keys.” *Communications of the ACM* 52.5 (2009): 91-98.

[4] Dautenhahn, Nathan, et al. “Nested kernel: An operating system architecture for intra-kernel privilege separation.” ACM SIGPLAN Notices 50.4 (2015): 191-206.

[5] Kaplan, David, Jeremy Powell, and Tom Woller. “AMD memory encryption.” White paper, Apr (2016).

[6] Barham, Paul, et al. “Xen and the art of virtualization.” *ACM SIGOPS operating systems review*. Vol. 37. No. 5. ACM, 2003.

[7] Liu, Yutao, et al. “Thwarting memory disclosure with efficient hypervisor-enforced intra-domain isolation.” *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[8] Chen, Xiaoxin, et al. “Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems.” *ACM SIGARCH Computer Architecture News*. Vol. 36. No. 1. ACM, 2008.

[9] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innova- tive technology for CPU based attestation and sealing. *In Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13.

[10] Virtualization, AMD64. “Secure virtual machine architecture reference manual.” *AMD Publication* 33047 (2005).

[11] Uhlig, Rich, et al. “Intel virtualization technology.” *Computer* 38.5 (2005): 48-56.

[12] Virtualization, A. M. D. "Amd-v nested paging." White paper. [Online] Available: http://sites. amd. com/us/business/it-solutions/ virtualization/Pages/amd-v. aspx (2008).

[13] Necula, George, et al. "CIL: Intermediate language and tools for analysis and transformation of C programs." *Compiler Construction*. Springer Berlin/Heidelberg, 2002.

[14] Cherednichenko, Oksana, A. A. Baranov, and T. I. Morozova. "Side-channel Attack." (2013).

[15] Kim, Yoongu, et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors." *ACM SIGARCH Computer Architecture News*. Vol. 42. No. 3. IEEE Press, 2014.

[16] Suh, G. Edward, et al. "AEGIS: architecture for tamper-evident and tamper-resistant processing." *Proceedings of the 17th annual international conference on Supercomputing*. ACM, 2003.

[17] Lie, David, Chandramohan A. Thekkath, and Mark Horowitz. "Implementing an untrusted operating system on trusted hardware." ACM SIGOPS Operating Systems Review 37.5 (2003): 178-192.

[18] Lie, David, et al. "Architectural support for copy and tamper resistant software." ACM SIGPLAN Notices 35.11 (2000): 168-177.

[19] Rogers, Brian, et al. "Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly." Microarchitecture, 2007. MICRO 2007. *40th Annual IEEE/ACM International Symposium on*. IEEE, 2007.

[20] Champagne, David, and Ruby B. Lee. "Scalable architectural support for trusted software." *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010.

[21] Chhabra, Siddhartha, et al. "SecureME: a hardware-software approach to full system security." *Proceedings of the international conference on Supercomputing*. ACM, 2011.

[22] Xia, Yubin, Yutao Liu, and Haibo Chen. "Architecture support for guest-transparent VM protection from untrusted hypervisor and physical attacks." *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 2013.

[23] Xia, Yubin, et al. "Secure outsourcing of virtual appliance." *IEEE Transactions on Cloud Computing* (2015).

[24] Arnautov, Sergei, et al. "SCONE: Secure Linux Containers with Intel SGX." OSDI. 2016.

[25] Baumann, Andrew, Marcus Peinado, and Galen Hunt. "Shielding applications from an untrusted cloud with haven." ACM Transactions on Computer Systems (TOCS) 33.3 (2015): 8.

[26] Dinh, Tien Tuan Anh, et al. "M2R: Enabling Stronger Privacy in MapReduce Computation." *USENIX Security Symposium*. 2015.

[27] Schuster, Felix, et al. "VC3: Trustworthy data analytics in the cloud using SGX." *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015.

[28] Hunt, Tyler, et al. "Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data." OSDI. 2016.

[29] Seo, Jaebaek, et al. "SGX-Shield: Enabling address space layout randomization for SGX programs." *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA. 2017.

[30] Shih, Ming-Wei, et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs." *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA. 2017.

[31] Guan, Le, et al. "Protecting private keys against memory disclosure attacks using hardware transactional memory." *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015.

[32] Chen, Yaohui, et al. "Shreds: Fine-grained execution units with private memory." *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016.

[33] Novark, Gene, and Emery D. Berger. "DieHarder: securing the heap." *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010.

[34] Dong, Xinshu, et al. "Protecting sensitive web content from client-side vulnerabilities with CRYPTONS." *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013.

[35] Parker, T. Paul, and Shouhuai Xu. "A Method for Safekeeping Cryptographic Keys from Memory Disclosure Attacks." *INTRUST*. 2009.

[36] Guan, Le, et al. "Copker: Computing with Private Keys without RAM." *NDSS*. 2014.

[37] Chen, Haibo, et al. "Daonity–Grid security from two levels of virtualization." *information security technical report* 12.3 (2007): 123-138.

[38] Chen, Haibo, et al. "Tamper-resistant execution in an untrusted operating system using a virtual machine monitor." (2007).

[39] Cheng, Yueqiang, Xuhua Ding, and R. Deng. "Appshield: Protecting applications against untrusted operating system." *Singaport Management University Technical Report, SMU-SIS-13* 101 (2013).

[40] Götzfried, Johannes, et al. "HyperCrypt: Hypervisor-based Encryption of Kernel and User Space." *Availability, Reliability and Security (ARES), 2016 11th International Conference on*. IEEE, 2016.

吴宇明 于 2015 年在西安交通大学软件工程与日语专业获得工学、文学双学位。 现在上海交通大学软件工程专业攻读工学硕士学位。研究领域为虚拟化与系统安全。
Email: yumingwu233@gmail.com

陈海波 于 2009 年在复旦大学计算机系统结构专业专业获得工学博士学位。现任上海交通大学教授、博士生导师。主要研究方向为系统软件、系统结构与系统安全。
Email: haibochen @sjtu.edu.cn

刘宇涛 于 2017 年在上海交通大学大学软件工程专业获得工学博士学位。现任华为公司高级工程师。研究领域为操作系统、虚拟化安全和手机安全。Email: mctrain016@gmail.com