软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(1):137−166 [doi: 10.13328/j.cnki.jos.006095]

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

SGX 应用支持技术研究进展*

董春涛[1,3,4], 沈晴霓[1,3,4], 罗 武[2,3,4], 吴鹏飞[1,3,4], 吴中海[1,2,3,4]

[1](北京大学 软件与微电子学院,北京 102600)

[2](北京大学 信息科学技术学院,北京 100871)

[3](北京大学 软件工程国家工程研究中心,北京 100871)

[4](高可信软件技术教育部重点实验室(北京大学),北京 100871)

通讯作者: 沈晴霓, E-mail: qingnishen@ss.pku.edu.cn

摘 要: 安全与可信是云计算中极为重要的需求,如何保护用户在云平台上托管的应用程序代码和数据的安全、防止云服务提供商和其他攻击者窃取用户机密数据,一直是个难题.2013 年,Intel 公司提出了新的处理器安全技术 SGX,能够在计算平台上提供一个用户空间的可信执行环境,保证用户关键代码及数据的机密性和完整性.SGX 技术自提出以来,已成为云计算安全问题的重要解决方案.如何有效地应用 SGX 技术来保护用户的应用程序,成为近年来的研究热点.介绍了 SGX 的相关机制和 SDK,概括了 SGX 应用所面临的安全问题、性能瓶颈问题、开发困难问题和功能局限性等问题,总结并归纳了 SGX 应用支持技术的研究进展,包括 SGX 应用安全防护技术、SGX 应用性能优化技术、SGX 应用辅助开发技术和 SGX 功能扩展技术,并展望了未来的发展方向.

关键词: 可信计算;SGX;系统安全;云安全;应用开发

中图法分类号: TP316

中文引用格式: 董春涛,沈晴霓,罗武,吴鹏飞,吴中海.SGX 应用支持技术研究进展.软件学报,2021,32(1):137−166. http://www.jos.org.cn/1000-9825/6095.htm

英文引用格式: Dong CT, Shen QN, Luo W, Wu PF, Wu ZH. Research progress of SGX application supporting techniques. Ruan Jian Xue Bao/Journal of Software, 2021,32(1):137−166 (in Chinese). http://www.jos.org.cn/1000-9825/6095.htm

Research Progress of SGX Application Supporting Techniques

DONG Chun-Tao[1,3,4], SHEN Qing-Ni[1,3,4], LUO Wu[2,3,4], WU Peng-Fei[1,3,4], WU Zhong-Hai[1,2,3,4]

[1](School of Software and Microelectronics, Peking University, Beijing 102600, China)

[2](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[3](National Engineering Research Center for Software Engineering, Peking University, Beijing 100871, China)

[4](Key Laboratory of High Confidence Software Technologies of Ministry of Education (Peking University), Beijing 100871, China)

Abstract: Security and trustworthiness are extremely important requirements for cloud computing. How to protect critical application code and data on the cloud platform and prevent cloud service providers and other attackers from stealing user's confidential data is a difficult problem. In 2013, Intel proposed a new processor security technology SGX, which can provide a trusted execution environment for user space on the computing platform to ensure the confidentiality and integrity of critical user code and data. Since SGX technology is proposed, it has gradually become an important solution to cloud computing security issues. How to effectively apply SGX technology to protect application has become research a hotspot in recent years. In this paper, the mechanisms and SDK of SGX are introduced, and the bottlenecks such as security issues, performance bottlenecks, development difficulties, and function limitations faced by SGX application are summarized. The research progress of SGX application supporting techniques are analyzed and summarized, including

* 基金项目: 国家自然科学基金(61672062, 61232005)
Foundation item: National Natural Science Foundation of China (61672062, 61232005)
收稿时间: 2019-11-24; 修改时间: 2020-04-26, 2020-05-14; 采用时间: 2020-06-01; jos 在线出版时间: 2020-07-27

SGX application security protection technique, SGX application performance optimization technique, SGX application assisted development technique, and SGX function extension technique. Finally, the development directions of SGX application supporting techniques are suggested.

Key words: trusted computing; SGX; system security; cloud security; application development

随着云计算与大数据技术的快速发展,越来越多的应用程序被部署到第三方数据中心和云平台中,如 Amazon AWS[1]和 Microsoft Azure[2].但是,这也带来了一系列安全问题,例如:应用程序在云平台以明文方式处理敏感数据,容易被云服务提供商或其他攻击者攻击.因此,应用程序必须能够抵御云服务提供商和其他攻击者的攻击.传统手段中,基于加密的保护技术[3−5]只能支持有限的运算操作,全同态加密[6]可以支持任意操作,但增加了大量的开销.当前,基于 CPU 提供的可信执行机制是不可信环境中保护应用程序的重要手段,例如 Intel 推出的 SGX(software guard extensions)指令集扩展[7].

SGX 是一种基于内存加密和证明的硬件机制,可以为应用程序提供用户空间的可信执行环境 enclave[8],并将程序执行和状态与底层操作系统(operating system,简称 OS)、虚拟机管理程序、固件、I/O 设备等隔离开来,保障用户关键代码和数据的机密性与完整性不受恶意软件的破坏.SGX 的可信计算基(trusted computing base,简称 TCB)仅包括 CPU,避免了基于软件的 TCB 自身存在安全漏洞的风险,提升了系统的安全性.同时,SGX 可以保障运行时的可信执行环境,抵御恶意代码对程序被保护内容的访问与篡改,进一步增强了系统的安全性.SGX 强大的安全保障能力与性能保证,使其得到了广泛的应用与发展.但是 SGX 在应用中也暴漏出诸多的瓶颈问题,如何解决这些瓶颈问题也成为了学术界和工业界的重要研究方向.本文通过整理和归纳相关研究工作,总结了目前 SGX 应用面临的瓶颈问题,主要包括应用安全问题、应用性能瓶颈、应用开发的困难性和应用功能的局限性,并对这 4 类问题的解决方案分别进行了分析和总结.

安全性是 SGX 最基本和最重要的目标.SGX 可以为代码和数据提供机密性和完整性保证,但其在实际应用中依然会面临诸多的安全风险和威胁,例如过大的 TCB、暴露过多的对外接口和敏感代码的安全划分与验证以及侧信道攻击[9−11]等.针对上述安全风险与安全威胁,学术界已经提出了多种方案,本文将其归纳为 TCB 最小化、对外接口最少化、敏感代码的自动化生成与安全检测以及潜在的安全威胁分析与防护这 4 种类型.

性能开销是目前制约 SGX 应用推广的重要因素.SGX 为代码和数据提供机密性和完整性保证,同时也带来了较高的性能开销.SGX 性能开销主要包括进入和退出 enclave 导致的 enclave 切换开销、内存加解密开销和页替换(paging)开销这 3 个方面.针对上述性能开销的原因,本文总结和梳理了目前研究工作中的性能优化方案,将 SGX 性能优化归纳为模式(enclave)切换性能优化、内存加密优化和内存页替换这 3 种技术.

应用开发困难也是制约 SGX 发展和推广的重要原因.Intel 公司为用户提供了软件开发包(software development kit,简称 SDK),但是使用该 SDK 开发 SGX 应用程序却并不容易,甚至错误的使用方式会导致安全风险.目前,SGX 应用开发的困难问题主要包括敏感代码划分工作量大且无法直接支持旧有的应用程序、缺少安全检测与性能测试等辅助开发工具、支持的语言类型有限和缺少操作系统库(library OS)支持.本文总结了相关研究工作提出的 SGX 应用辅助开发技术,主要包括安全开发语言、辅助开发工具和通用系统库等.

云环境是 SGX 技术应用的主要场景,但是目前 SGX 技术自身缺乏对虚拟机(vitual machine,简称 VM)迁移和容器(container)等云平台常用功能的支持.很多研究工作针对 SGX 缺乏的功能支持,对其功能进行了扩展,主要包括虚拟机迁移支持技术和容器支持技术.

本文第 1 节对 SGX 的相关概念进行概述,并介绍 SGX 的关键机制和 SDK.第 2 节对 SGX 的应用现状以及瓶颈问题进行总结.第 3 节对 SGX 应用的安全风险以及已有的安全防护技术进行总结.第 4 节分析 SGX 性能开销的原因,并对相关性能优化技术进行归纳.第 5 节总结 SGX 应用开发的困难性问题以及已有的 SGX 应用辅助开发技术.第 6 节总结目前已有的 SGX 应用功能扩展技术.最后对全文进行总结,阐述 SGX 应用支持技术仍然需要解决的问题并展望未来的研究方向.

1 SGX 概述

SGX 是在原有 Intel 架构上扩展了一组新的指令集和内存访问机制[11],这些扩展能够在计算平台上提供一个可信的隔离空间,允许应用程序使用 enclave[7]保障用户关键代码和数据的机密性和完整性,不受特殊权限恶意软件的破坏[12].SGX 整体架构如图 1 所示.SGX 扩展指令集包括 17 条新指令,其中 5 条指令是用户指令,包括进入 enclave(EENTER)、退出 enclave(EEXIT)、恢复 enclave(ERESUME)、获取密钥(EGETKEY)和证明时获取 enclave 度量报告(EREPORT).其他 12 条指令是系统指令,用户通过操作系统的 SGX 驱动(driver)调用执行,主要包括 enclave 创建(ECREATE,EADD,EEXTEND,EINIT)、enclave 异步退出(AEX)、enclave 销毁(EREMOVE)、资源管理(EPA,ELDB/U,EWB,EBLOCK,ETRACK).因为 enclave 的创建必须在内核空间中进行,而 enclave 的交互仅限于用户空间应用程序,特权代码无法进入 enclave,非特权代码无法创建 enclave.

所有的 enclave 代码和数据都存储在 EPC(enclave page cache)中,EPC 是一块用来存放 enclave 和 SGX 数据结构[7]的被保护的物理内存区域,EPC 内存以页为单位进行管理,对 EPC 页进行访问控制的元数据信息保存在硬件结构 EPCM(enclave page cache metadata)里.SXG 应用开发需要将应用程序分为可信和不可信两个部分,两部分之间通过用户定义的 Ecall 和 Ocall 代码进行调用.

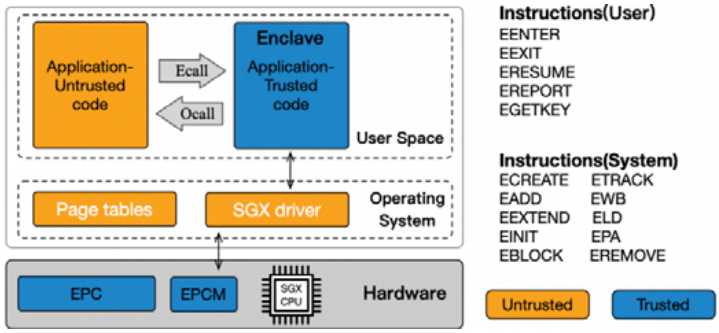

Fig.1 Architecture description of SGX

图 1 SGX 整体架构

1.1 SGX安全机制

SGX 旨在保护 enclave 内计算的机密性和完整性,抵御恶意软件攻击和物理攻击.SGX 技术主要包括隔离执行、认证(attestation)和密封(sealing)这 3 个核心机制.本小节对 SGX 的相关核心安全机制进行介绍.

1.1.1 隔离执行

SGX 的核心概念就是 enclave,它是一个被保护的安全容器,用于存储应用程序的敏感数据和代码[13,14].Enclave 内除代码和数据以外,还包括元数据和 TCS(thread control structure),TCS 用于保存进入或退出 enclave 时恢复 enclave 线程的特殊信息(如图 2 所示).EPC 是 PRM(processor reserved memory)的一部分,PRM 则是内存的一部分,BIOS 通过配置一组范围寄存器分配 PRM,系统软件或外围设备无法访问.

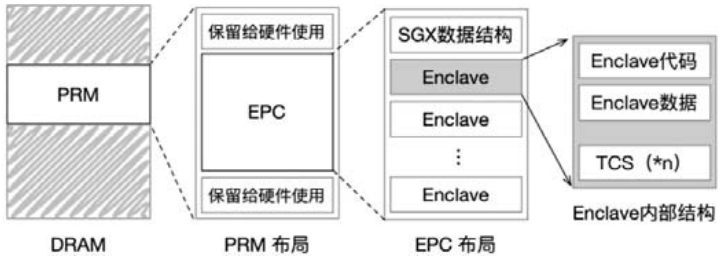

Fig.2 SGX sample layout of PRM and EPC

图 2 PRM 和 EPC 布局示意图

图 2 是一个 PRM 和 EPC 布局示意图.EPC 内存以页为单位进行管理,对 EPC 页进行访问控制的元数据信息保存在硬件结构 EPCM 里,一个 enclave 页面对应一个 EPCM 表项.通过处理器的访问控制,可以保证每个 EPC 页面只能映射到特定的虚拟地址,确保每一个 EPC 页只允许与其相关联的 enclave 访问.

Enclave 中的代码、数据和 TCS 等在运行过程中始终被加密存储在内存中.处理器缓存(cache)中的数据在写到内存之前,处理器中的内存加密单元(memory encryption engine,简称 MEE)会对其进行加密,因此操作系统无法直接获取 EPC 中的内容,保证了应用程序代码和数据在运行过程中不受特权系统软件的攻击.SGX 的内存保护机制可以保证 enclave 外部的应用程序不能访问 enclave 内存,enclave 内部的代码在 EPC 范围内只能访问属于自己的内存,不能访问其他 enclave 的内存.这样的内存保护机制防止了 enclave 内部运行的程序被其他恶意软件盗取和篡改隐私信息;同时,保证了在物理上锁住 EPC 内存区域,将外部的访问请求视为访问了不存在的内存,使得外部的实体(直接存储器访问等)无法访问.

应用程序在申请创建一个 enclave 时,需要进行页面分配(ECREATE)、复制程序代码与数据(EADD)和度量操作(EEXTEND).Enclave 被加载以后,需要进行完整性验证,以判断特权软件在创建 enclave 过程中是否篡改了程序和数据.在 enclave 初始化之后(EINIT),不可信任的代码(在 enclave 外)执行可信代码(在 enclave 内)的唯一方法是调用 EENTER 指令.EENTER 执行上下文切换到 enclave,保存不可信代码的状态,并恢复可信代码的最后已知状态.该上下文切换在概念上类似于用于英特尔 VTX 技术中的虚拟机上下文切换的 VMENTER 和 VMEXIT[15].

SGX SDK 提供了封装代码,称为 ecall[16],用于准备执行环境并调用 EENTER 指令.由于 enclave 以用户级权限运行可信代码,因此 enclave 无法访问硬件或其他系统资源.为了访问外部资源,例如文件系统、网络或时钟,enclave 必须退出到不可信代码,它可以通过 EEXIT 指令完成.EEXIT 执行反向上下文切换并切换回不可信的代码.执行此操作的 SDK 封装代码称为 ocall[16].为了避免泄漏隐私数据,正在执行 enclave 代码的 CPU 不直接处理中断、故障(例如页面错误)或虚拟机退出.相反,CPU 首先执行 AEX(asynchronous enclave exit)从 enclave 代码切换到非 enclave 代码,然后才处理中断故障或 VM 退出.

1.1.2 认 证

在 enclave 初始化之前,代码和数据是公开的,敌手可窃取和篡改.SGX 提供一个证明指令 EREPORT 来抵御敌手窃取和篡改初始化之前的代码和数据.具体地,enclave 通过调用 EREPORT 产生一个硬件签名的 enclave 度量值报告,并发送给验证方进行证明.SGX 支持本地认证和远程认证两种类型的身份认证方式[7,17]:本地认证用于平台内部 enclave 之间,用来认证报告的 enclave 和自己是否运行在同一个平台上;远程认证则是用于平台之间,用于远程认证者认证 enclave 的身份信息.

当 enclave 向平台上其他 enclave 报告身份时,首先要获取自身的身份信息和属性以及平台硬件 TCB 信息,附加上用户希望交互的数据,生成报告结构(report structure).然后获取目标 enclave 的报告密钥,对报告结构生成一个消息认证码(message authentication code,简称 MAC)标签,形成最终的报告结构,传递给目标 enclave,由目标 enclave 验证请求报告身份的 enclave 与自己是否运行于同一平台.远程认证则需要引入一个引用(quoting) enclave,由引用 enclave 创建用于平台认证的签名密钥 EPID(enhanced privacy identification).被认证 enclave 首先执行 EREPORT 指令,将身份和附加信息生成报告结构,并利用引用 enclave 的报告密钥生成一个 MAC 标签,一起发给引用 enclave.引用 enclave 通过该结构验证被认证 enclave 是否运行于同一平台,然后将它封装为一个引用结构体 QUOTE,并使用 EPID 进行签名,将 QUOTE 和签名一同发送给远程认证者.远程认证者验证 enclave 报告是否由可靠的 Intel 处理器生成.

1.1.3 密 封

密封是指 enclave 使用 EGETKEY 指令获取一个硬件产生的封装密钥对 enclave 数据进行加密,并将加密数据存储在 enclave 之外不可信存储中的过程[17].Enclave 数据根据 enclave 的标识(MRENCLAVE)或签名标识(MRSIGNER)进行密封,对 MRENCLAVE 加密的数据只能由相同的 enclave 解密,而对 MRSIGNER 加密的数据可以由同一开发人员签名的任何 enclave 解密.在这两种情况下,密封密钥都来自特定于 CPU 的密钥,因此数据

只能在它被密封的同一台物理机器上打开.SGX 密封保证了密封数据的机密性和完整性,但不会自动提供回放(roll-back)保护.如果需要,enclave 开发人员必须检查已提供的密封数据的版本是否正确,通常通过使用安全单调计数器(monotonic counter)来实现.

1.2 SGX管理机制

1.2.1 SGX 内存管理机制

Enclave 与主机程序共用一片虚拟地址空间,部分 enclave 中映射到 EPC 页的虚拟内存称为 enclave 线性地址空间(enclave linear address range,简称 ELRANGE).这段地址空间在 EPC 分配终止后,对于主机进程来说是不能访问的.任何试图读写 ELRANGE 地址空间的 enclave 外部代码都会产生一个未定义行为的错误.

Enclave 执行总是在受保护模式下,并由 OS 内核或 hypervisor 进行地址转换.SGX 为了不影响传统 OS 对平台资源的管理和分配,实现 enclave 中的页到 EPC 中帧转换的页表(page table)仍由 OS 管理[7].OS 和 hypervisor 完全控制页表和扩展页表(extended page table,简称 EPT),enclave 代码使用与其主机应用程序相同的地址转换过程和页表.由于操作系统或 hypervisor 是不可信的,因此 CPU 必须要防止对 enclave 的地址转换攻击[10].当从 EPC 页地址转换到物理地址时,CPU 使用在 SGX enclave 控制结构(SGX enclave control structure,简称 SECS)中存储的初始分配信息来保证传递给地址转换过程的 EPC 页虚拟地址与 EPCM 中保留的 EPC 入口地址相匹配.以此来防止操作系统将 ELRANGE 地址映射到不受保护的内存,使得 ELRANGE 内存对 enclave 之外的软件不可见.

1.2.2 EPC 页驱逐(eviction)

操作系统可以将 EPC 页面驱逐到不可信的 DRAM 中,并将其加载回来,如图 3 所示.系统软件使用 EWB 指令驱逐 EPC 页面,该指令产生 ELDU 指令恢复被驱逐页面所需的所有数据[7].SGX 通过加密(EGETKEY)来保证被驱逐的 EPC 页面存储在不可信内存中时的机密性和完整性.EWB 和 ELDU/ELDB 指令使用一个 ID 来识别拥有被换出页面的 enclave.EWB 驱逐 EPC 的内容时,会创建一个 8 字节的随机数,称为页面版本(page version).SGX 的新鲜度(freshness)保证则建立在安全存储页面版本的假设之上,因此,EWB 将其存储在一个版本数组(version array,简称 VA)中,VA 页面也可以驱逐.被驱逐的页面依赖于存储其页面版本的 VA 页面,在 VA 页面被重新加载之前无法被重加载到 EPC 中.基于这种依赖关系创建的关系图称为驱逐树(eviction tree).驱逐树将 EPC 页面作为叶子,将 VA 页面作为内部节点.页面的父节点是存储其页面版本的 VA 页面[7].

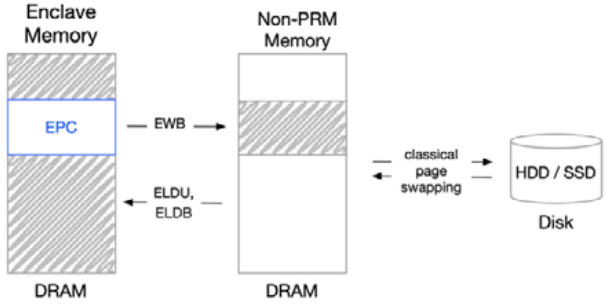

Fig.3 EPC pages evicted into non-PRM DRAM

图 3 EPC 页驱逐到非 PRM 内存

1.3 SGX SDK

为了简化 SGX 应用程序的开发,Intel 发布了 SDK[14].它向开发人员隐藏了 SGX 硬件细节信息,并引入了 ecall 和 ocall 的概念,分别用于进入和退出 enclave,如同普通的函数调用那样.通过使用 SGX SDK,开发人员可以创建加载到 enclave 的代码,并定义 enclave 代码与其他不可信的应用程序代码之间的 ecall 和 ocall 接口.

Intel 提供了一个名为 edger8r 的边缘函数创建工具,用于自动生成 ecall 和 ocall 的安全封装代码.要自动生

成 ecall 和 ocall 代码,开发人员必须使用 SGX 规定的语法,在 enclave 描述语言(enclave description language,简称 EDL)扩展文件中声明 ecall 和 ocall 函数.然后,edger8r 解析 EDL 文件生成封装代码,如图 4 所示.Ecall 和 ocall 被认为是边缘函数(edge function),因为对它们的执行会跨越安全边界.函数的参数需要编组(marshall),并从加密内存复制到明文内存,反之亦然.为了边界交互的安全,需要对调用的参数执行多项安全检查,特别是在参数是指针的情况下.Enclave 代码必须验证所访问数据的完整性,如 ecall 的参数、ocall 的返回值和从不可信内存读取的数据.

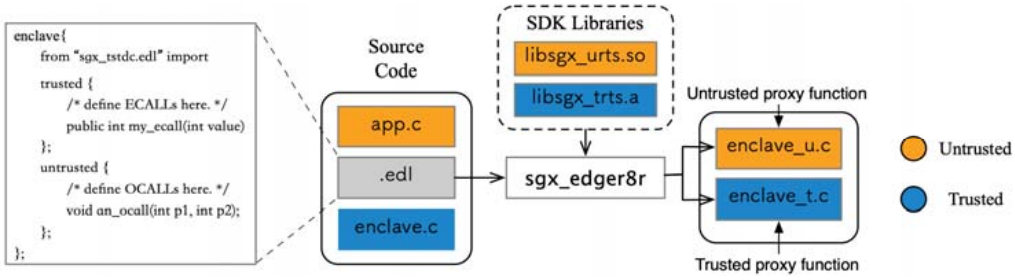

Fig.4 Process of edger8r generates wrapper code from EDL file

图 4 Edger8r 解析 EDL 文件生成封装代码的过程

Enclave 支持多线程,因此需要同步原语.由于在 enclave 内部无法进行睡眠等待,因此,SGX SDK 提供的 enclave 内部同步原语实现了通过 ocall 以在 enclave 外部进行睡眠等待.SDK 提供的互斥锁的工作方式如下:如果线程尝试锁定未锁定的互斥锁,则此操作成功完成而无需离开 enclave.每当线程尝试锁定已锁定的互斥锁时,它将进入队列并通过 ocall 退出 enclave 进入睡眠状态;然后,持有互斥锁的线程需要通过进入队列并通过 ocall 离开 enclave 来唤醒睡眠线程.因此,SGX SDK 提供的互斥锁会导致两次 ocall,开销比较大.

2 SGX 应用现状及瓶颈问题

SGX 技术推出以来,受到了学术界和工业界的极大关注,目前在很多领域已经得到了广泛的应用,如文献[18]重点讨论了 SGX 在保障应用与网络安全方面的应用研究进展.但是,SGX 在应用的同时也暴露出很多安全缺陷,如 Wang 等人重点讨论了针对 SGX 的侧信道攻击和防御方面的研究进展[19].事实上,除了侧信道问题,SGX 应用还暴露出了许多其他安全性问题、性能瓶颈、开发困难、功能局限等诸多应用方面的问题.本节首先介绍 SGX 目前的应用现状,然后总结 SGX 在应用中暴露出的诸多问题,最后对 SGX 应用支持技术的研究方向进行总结.

从应用的场景来看,SGX 可以用于构建安全的云应用[20−22],用于保护网络通信[23−27]以及增强本地应用的安全性[28−31].SGX 首先被应用在云计算安全领域,如构建云平台应用安全隔离执行环境[32]、构建云平台大数据安全可信计算环境[33,34]和对虚拟化技术的支持[35].例如,Hunt 等人提出的 Ryoan 系统[32],利用 enclave 实现了一个分布式沙箱来保护沙箱实例不受潜在恶意平台的攻击.Schuster 等人提出的 VC3 框架[33],基于 SGX 实现了保护代码和数据的机密性和完整性 mapreduce[36]框架.Brenner 等人提出的 Securekeeper 框架[21],通过设计存储数据的加解密方案和在 enclave 中对数据处理的方式,实现了对管理数据隐私性和完整性的保护.Arnautov 等人设计了一个用于 Docker[37]的安全容器环境 SCONE[35],利用 SGX 来保护 Docker 容器内进程免受外部攻击.微软与英特尔协作利用 SGX 以增强云中的安全性,帮助数据和代码在使用过程中得到更强的保护[38].IBM 利用英特尔 SGX 配合 IBM Cloud Data Shield 以帮助保护使用中的数据[39].

SGX 也被用于保护网络通信安全和本地应用安全.如在文献[23]中,作者提出一种保护方案(S-NFV),利用 SGX 对网络功能虚拟化(network function virtualization,简称 NFV)产生的状态进行安全隔离保护.Andrew 等人提出的 Haven 系统[30],通过修改 Drawbridge 沙箱[40]实现了 Windows 应用直接在沙箱中的 enclave 内安全运行,为 Windows 应用提供了安全保障.

除了应用于云计算、网络通信和本地应用以外,许多解决方案都受益于 SGX 提供的安全保护,包括多方计算[41]、机器学习[42,43]、区块链[44−46]、人工智能和生物识别技术保护等.例如,文献[44]介绍了一种基于 SGX 区块链资源挖掘框架 REM(resource-efficient mining),REM 利用 SGX 的安全保证实现了有效工作量证明(proof-of-useful-work,简称 PoUW),以提供可信的工作负载报告.UCloud 引入了 SGX 技术,为智能合约提供了区块链所不能提供的机密性,可拓展的计算节点也解决了区块链本身无法应对复杂应用场景的问题.在保留区块链去中心化、用户互信的基础之上,通过可信硬件执行机密性高、需要相对复杂计算能力的程序,并通过区块链对执行结果进行记录和验证,实现可追溯的特性[96].Ekiden 同样使用 SGX 保证智能合约的机密性,并利用 SGX 可信性提高区块链的共识效率[47].蚂蚁金服金融科技推出的可信计算云服务则是将蚂蚁区块链自主研发的虚拟机内嵌在 SGX 可信执行环境中,在数据保密的通用链下能够实现智能合约[48].

SGX 技术目前被广泛地应用到各个领域,同时,SGX 技术在应用中也暴露了诸多瓶颈问题.如何解决 SGX 在应用中的瓶颈问题,也成为学术界的研究热点.本文首先对目前 SGX 技术相关应用和研究工作进行了梳理和总结,将 SGX 技术应用的瓶颈总结为以下 3 个方面.

- 应用安全问题.SGX 在 TCB 大小、enclave 接口等 SGX 应用程序设计方面存在诸多安全隐患,同时也面临侧信道攻击、内存攻击和硬件攻击等传统的安全威胁;
- 应用性能瓶颈.SGX 在内存加密、执行模式切换(包括 ecall、ocall 和 AEX)和内存页替换等方面存在高昂的性能开销,这严重制约了 SGX 的应用和推广;
- 应用开发的困难性和功能的局限性.Intel 公司提供的 SDK 虽然简化了 SGX 的使用,但仍需程序员手动地将应用程序划分为可信和不可信两部分,不能直接无缝对接原有的应用程序,导致 SGX 应用程序的开发工作量很大.SGX 提供了强大的安全功能,但对虚拟机迁移和容器等云平台常用技术支持不足,难以满足当前云平台应用对 SGX 的需求.

针对 SGX 技术的上述应用瓶颈问题,工业界和学术界进行了具体的分析,并提出了相应的解决方案.本文总结了相关文章,主要包括 CCS、S&P、NDSS 等顶级会议,对这些工作进行了总结和分类,见表 1,主要可以分为三大类.

Table 1 Related researchs of SGX application supporting techniques

表 1 SGX 应用支持技术相关研究工作

工作分类	主要工作	主要方案类型	相关系统(工作)简称
安全增强	增强 SGX 技术的安全性	(1) TCB 最小化 (2) 对外接口最少化 (3) 敏感代码的自动生成与检测 (4) 潜在威胁的分析与防护	SecureKeeper[21],VC3[33],Panoply[49]等 Haven[30],SCONE[35],SGXKernel[50],Graphene[51]等 Glarming[52],Moat[53]等 T-SGX [49],SGXBOUNDS[50],Ref.[42,43]等
性能优化	优化 SGX 技术的性能开销	(1) 减少 enclave 上下文切换 (2) 减少页面换入换出 (3) 减少 enclave 内存页访问	Hotcall[56],Switchless[57],SGXKernel[50],Graphene[51]等 Eleos[58],VAULT[59],EActor[60],STANlite[20]等 SCONE[35]等
易用性改进	SGX 应用程序辅助开发技术	(1) 安全开发语言 (2) 辅助开发工具 (3) 通用操作系统库和函数库	Rust SDK[61],MesaPy[62],GOTEE[63]等 Glamdring[52],Moat[53],sgx-perf[64]等 Haven[30],SCONE[35],SGXKernel[50],Graphene[51]等
	SGX 功能扩展	(1) 支持虚拟机迁移 (2) 支持容器 (3) 构建可信外部路径	Ref.[65],Ref.[66],Ref.[67]等 SCONE[35],Ref.[68]等 SGXIO[69]等

- 安全增强:分析 SGX 应用存在的安全风险和面临的安全威胁,并提出增强 SGX 应用安全性的相关方案.目前,SGX 应用安全增强方案主要有 TCB 最小化、对外接口最少化、敏感代码安全生成与检测和潜在安全威胁的分析与防护等;
- 性能优化:分析 SGX 性能开销的原因,并提出相应的性能优化方案.目前,改进 SGX 应用性能的解决方案主要包括减少模式切换(enclave 切换)开销、减少页替换开销以及减少对 enclave 内存页的使用和访问等;
- 易用性改进:主要包括 SGX 应用辅助开发技术和功能扩展技术.应用程序辅助开发技术主要包括安全

开发语言、辅助开发工具以及通用系统库和函数库等;功能扩展则主要包括对虚拟机迁移和容器等云平台常用技术的支持方案等.

通过上述对 SGX 应用支持技术研究现状的总结和分类,可以将 SGX 应用程序支持技术的目标归纳为安全、性能和易用性这 3 个方面.这 3 个目标不是相互独立的,而是互相关联和影响的,如 TCB 大小就会关系到应用的安全和性能.

本节首先介绍了 SGX 技术的应用研究现状,分析了 SGX 应用中暴露出的安全性问题、性能瓶颈、开发困难、功能局限等诸多方面的问题,然后将 SGX 应用支持技术的研究方向概括为 SGX 应用安全增强技术、SGX 应用性能优化技术、SGX 应用程序辅助开发支持技术和 SGX 应用功能扩展技术这 4 个方向.本文第 3 节~第 6 节将分别从这 4 个方面进行详细的分析和总结.

3 SGX 应用安全防护技术

SGX 通过一系列技术来保护程序的安全,并且将系统的可信计算基缩小到 CPU,相比以往将整个操作系统或特权软件(如 hypervisor)视为可信计算基,可以避免更多的系统攻击带来的危害.但是 SGX 在实际应用中依然会面临诸多的安全风险,如过大的 TCB 和暴露过多的外部接口等.本文将 SGX 应用程序设计和开发时需要考虑的安全风险归纳为以下 4 个方面.

- TCB 的大小.有研究[70,71]表明,软件漏洞的数量以及潜在的安全漏洞会与代码大小成比例地增加.尽可能地减小 TCB,可以有效地减少 enclave 内部代码漏洞潜在的安全风险;
- 外部接口的多少.由于 enclave 需要与不可信区域进行交互,外部接口就成为不可信操作系统的攻击面,控制主机操作系统的攻击者可以使用此接口来破坏在 enclave 内运行的进程或者推测出敏感信息.因此,设计 enclave 程序时需要考虑到该隐患;
- 开发的工作量和可靠性.SGX 程序开发最重要的是需要将代码划分为敏感和非敏感两部分,SGX SDK 目前不支持现有的应用程序直接运行在 SGX 环境中,SGX 程序开发需要经验和时间,并且不合理的敏感代码划分可能会破坏 enclave 代码的安全性.因此,开发时需要充分考虑敏感代码划分的工作量和可靠性;
- 面临的安全威胁.有研究表明,SGX 面临着侧信道攻击[9−11]、内存攻击[55,72]等多种攻击的威胁.开发 SGX 应用程序时也需要考虑到这些安全威胁.比如,enclave 中的数据处理过程如果依赖于外部数据,攻击者可以通过一些不合法输入,发起对 enclave 的缓冲区溢出攻击,这些攻击可能会改变可信区中程序的执行流程以及获取可信区中的敏感信息.

针对 SGX 目前在应用中存在的上述安全问题,工业界和学术界提出了很多解决方案,本文对相关解决方案进行了梳理和总结,具体分类如下.

- TCB 最小化.SGX 应用程序的 TCB 由 enclave 代码和可信硬件组成.根据最小特权原则[73],只有应用程序中需要访问敏感数据的代码才应在 enclave 内执行.最小化 TCB,可以保证 enclave 内部代码面临尽可能少的威胁;
- 对外接口最少化.除了 TCB 大小外,enclave 对外接口的复杂性同样会影响 enclave 内部代码和数据的安全性.减小 enclave 和不可信操作系统之间的接口,可以减小暴露的攻击面,从而减小攻击者成功攻击的可能性[70];
- 敏感代码的安全生成与检测.利用自动化的开发工具对 SGX 应用程序的代码进行自动划分,可以极大地减少开发人员的工作量;通过对 enclave 代码进行安全检测,可以有效地增强敏感代码的安全性;
- 潜在的安全威胁防护.本文针对常见的潜在安全威胁进行了分析和总结,并给出了目前研究中已有的防护方案和相关的开发建议.

本节下面将对这 4 类方案进行详细的介绍和分析.

3.1 TCB最小化

3.1.1 TCB 安全风险分析

本小节首先来分析 TCB 大小对 SGX 应用程序安全性的影响.目前使用 SGX 的应用在 TCB 中 enclave 代码部分的大小设计方面有 3 种选择.

- 预定义的少量专用敏感代码:将尽可能少的代码放入 enclave 中,一般只将需要直接处理敏感数据的代码放入 enclave 中,如 VC3[33],只将处理数据的 map/reduce 函数放入 enclave,而将其他不完全可信的平台代码排除在 enclave 外,这样可以避免平台可能存在的安全漏洞威胁 enclave 内代码和数据安全;
- 完整的应用程序及其依赖的库:将应用程序和 enclave 内程序运行时所依赖的系统库和函数库等部署在 enclave 内,如 Haven[30]、SCONE[35]和 Graphene[51]等系统,采用的方案是,在 enclave 内部署函数库或系统库来支持执行完整的应用程序.安全敏感和不敏感的应用程序代码和数据都位于 enclave 内,提高了性能,但从 TCB 的角度来看,却增加了 enclave 内部代码和数据面临的安全风险;
- 从特定应用中划分出敏感代码:将应用程序中的敏感代码划分出来作为 TCB,这里的敏感代码除直接处理敏感数据的代码外,还可能包括其他间接处理敏感数据的代码以及与 enclave 内部代码交互频繁的代码等.这类方案需要程序开发人员手动划分应用程序,开发工作量大,并且划分敏感代码缺乏参考标准,如果敏感代码划分得不够合理,可能会破坏 enclave 内代码和数据的安全性.

另外,目前 SGX v1 版本支持的 enclave 最大为 128M(应用程序实际能使用约 93M),一旦 TCB 大小超过 93M,就会触发频繁的页替换,增加了安全风险和性能开销.SGX v2 有可能进一步扩展 SGX 的可用物理内存,但目前 Intel 官方还未发布相关资料.

3.1.2 最小化 TCB 的安全方案

最小特权原则下,应该最小化 TCB,从而减少 enclave 内部漏洞造成的威胁.最小化 TCB 的相关安全方案一般仅把支持功能,或者敏感部分放入 enclave.应用程序必须遵守预定义的受限制的 enclave 接口[21,33,49].这些工作的核心是使用 SGX 提供的 enclave 保证运算的机密性,同时保证 enclave 尽可能地小.

Schuster 等人提出的 VC3 系统[33]就是采用了最小化 TCB 的设计方案.VC3 使用分布式系统 Hadoop[36]中计算节点提供的 enclave 来保护 map/reduce 任务的代码和数据,并严格限制 map/reduc 任务仅通过特定的接口与不可信的环境进行交互.VC3 将未经修改的 Hadoop 和操作系统均排除在 TCB 之外,最小化了 TCB,从而减少了 enclave 内部代码和数据面临的安全风险.因此,即使 Hadoop 和操作系统受到损害,也可以保证 map/reduce 任务代码和数据的机密性和完整性.并且,VC3 通过部署协议以保护分布式 MapReduce 计算结果的安全性和可验证性.VC3 在 TCB 最小的情况下,保证了数据和代码的完整性和机密性,并保证了计算结果的可验证性,但是由于 map/ reduce 任务只能通过特定的接口与不可信环境交互,因此其支持的功能非常有限.

Brenner 等人提出的保证所管理数据的隐私性和完整性的 ZooKeeper[74]框架 Securekeeper[21]同样采用最小化 TCB 的设计方案.SecureKeeper 系统通过设计所存储数据的加解密方案和在 enclave 中对所管理数据进行处理的方式,将轻量修改的 ZooKeeper 和操作系统排除在 TCB 之外,将系统的 TCB 保持在很小的水平,从而使在不可信环境下数据的隐私性和完整性得到了有效的保证.

Shinde 等人提出的 PANOPLY 系统[49]旨在最小化 SGX enclave 应用程序 TCB,同时又能为 enclace 代码提供丰富的操作系统抽象.PANOPLY 提出了一种称为微容器(micro-container or micron)的新概念,微容器是运行在 enclave 中应用程序的逻辑单元,微容器将标准的 Linux 抽象提供给应用程序.例如,除支持标准 Linux 系统调用外,启用微容器的应用程序还可以使用多进程、多线程、事件注册/回调(例如信号).PANOPLY 可以将应用程序分成一个或多个微容器,或者通过导入微容器库来增强安全性,如图 5 所示.同时,PANOPLY 保证 enclave 间交互的完整性,确保即使操作系统出现异常,应用程序的执行也仍然遵循合法的控制和数据流.

PANOPLY 将最小化 TCB 优先于性能作为目标.与以前的系统(例如操作系统库解决方案 SGXKernel[50]、Graphene[51])不同,PANOPLY 使用了一种简单的委托而不是模拟的设计理念,将操作系统抽象的实现委托给底层操作系统,而不是在 enclave 内进行仿真.PANOPLY 微容器只需要通过执行少量检查来抵御操作系统的恶意

响应.这种设计消除了在 enclave 内模拟基础操作系统的大量工作.通过这种设计,PANOPLY 提供的应用程序 TCB 比以前的操作系统库系统的数量级小 2 个数量级.

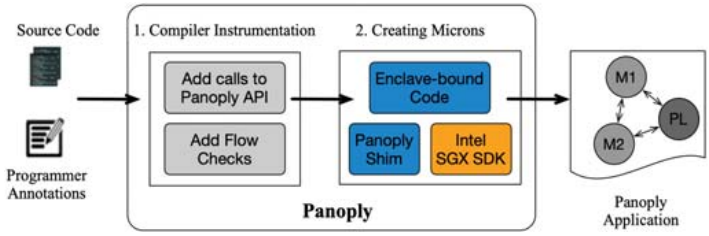

Fig.5 Panoply system overview

图 5 Panoply 系统概况

3.2 对外接口最少化

3.2.1 Enclave 接口的安全风险分析

由于 enclave 处于用户态,自身无法执行系统调用,需要与不可信区域进行交互,enclave 必须向主机操作系统公开外部接口.外部接口则会成为不可信操作系统的攻击面,控制主机操作系统的攻击者可以使用此接口来破坏在 enclave 内运行的进程并推测敏感信息,增大了安全风险和性能开销.如 Iago 攻击[75],这是来自不受信任的操作系统对应用程序的语义攻击,其中未经检查的系统调用返回值或许会影响应用程序.Iago 攻击难以进行全面检测,至少对于当前的可移植操作系统接口(portable operating system interface,简称 POSIX)[76]或 Linux 系统调用表接口而言是这样.因此,任何 SGX 框架都必须提供隔离支持,以验证或拒绝来自不受信任的操作系统的输入.接口复杂性直接影响隔离的复杂性,如操作系统库或填充(shim)库可以减小 enclave 接口的数量和复杂性,从而降低 Iago 攻击成功的风险.

总之,系统调用的返回值必须进行检查以防止 Iago 攻击.创建有原则的 enclave 接口可以更容易地抵御特定攻击.决定是否采用特定方案,使用 enclave 保护应用程序安全的重要判定原则是开发工作量及其是否适用于任何应用程序.目前主要有 3 种 enclave 接口方案.如图 6 所示.

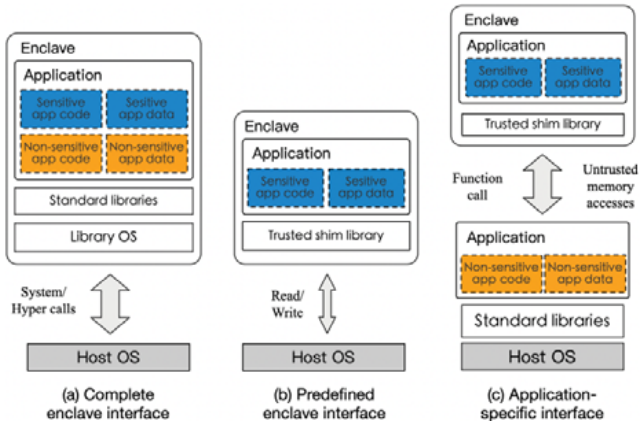

Fig.6 Design alternatives for the use of enclaves

图 6 Enclave 设计选择方案

- 完整的 enclave 接口:在 enclave 为应用程序提供完整的系统库支持,减少 enclave 与不可信的系统之间的交互,如在 enclave 提供库支持.例如,Graphene[51]是在 enclave 中使用一个操作系统库来支持快速部署未修改的 Linux 应用程序;

- 预定义的 enclave 接口:如图 6(b)所示,遵守预定义的受限制的 enclave 接口.例如,VC3[33]强制 map/reduce 任务仅通过特定的接口与不可信的环境进行交互;
- 特定应用程序的 enclave 接口:如图 6(c)所示,针对特定的应用程序划分可信与不可信代码,并设计相关的 enclave 接口.该方案开发要求高、工作量大,且无法在应用程序之间通用.

3.2.2 减少对外接口的安全解决方案

减少 enclave 接口方案一般会在 enclave 内设置操作系统库或者标准函数库来支持 enclave 内应用程序的执行,从而减少 enclave 内代码与操作系统之间的交互,例如 Haven[30]、SGXKernel[50]、Graphene[51]和 SCONE[35]等等.

Haven[30]使用 Drawbridge[40]操作系统库直接运行未修改的 Windows 应用程序.Haven 基于 drawbridge 沙箱机制,为用户程序的运行提供了一个微处理(picoprocess)容器,从而保证运行在里面的用户程序无法对外界系统造成破坏;再在容器中创建一个 enclave,将用户程序、系统库和防护模块(shield module)放进 enclave 中,以防止这些数据和代码被外界的特权软件或恶意程序所访问和篡改.系统库通过向下调用(downcalls)和向上调用(upcalls)的方式与 drawbridge 主机进行交互,用来完成用户程序需要的系统功能,防护模块配合检查函数的参数和返回结果,进而保护用户程序的安全执行.

Graphene[51]则是在 enclave 中部署一个操作系统库来支持在 SGX 上快速部署未修改的 Linux 应用程序.Graphene 为 SGX 提供了一个端口以及一些 SGX 改进措施,例如动态加载库(dynamically-loaded libraries)的完整性支持和安全多进程(secure multi-process)支持.Graphene 开销与使用填充层(shim library)的应用程序相近,大多数单进程应用的性能开销小于 2 倍,Graphene 目前已经开源.SGXKernel[50]也是在 enclave 内实现了一个操作系统库,通过结合异步交叉 enclave 通信和可抢占的 enclave 多线程避免 enclave 切换,其性能明显优于 Haven 和 Graphene.SCONE[35]则是在 enclave 中配置了标准 C 函数库的修改版本来支持重新编译的 Linux 应用程序.

文献[64]在 enclave 接口的安全设计方面给出了 3 个建议.

(1) SDK 允许将 ecall 定义为公有或私有[16]:公有 ecall 可以随时被调用,而私有 ecall 只能在 ocall 中被调用.将 ecall 定义为私有可以限制调用 ecall 的途径来增强 enclave 的安全性;

(2) 开发人员必须精确指定每个 ocall 中允许哪些 ecall 调用.如果没有指定特定的 ecall,则在执行期间会触发错误.如果开发人员未考虑特定的 ecall/ocall 组合,则攻击者可能会利用它来更改程序执行的控制流并获得对特定机密的访问权限;

(3) EDL 文件定义了作为 ecall 和 ocall 参数传递的指针的行为,其中,user_check 标明是否将指针留给开发人员.然而,user_check 可能带来安全漏洞,例如缓冲区溢出、time-of-check-to-time-of-use 攻击[77]或传递 enclave 内地址[78].因此,在进行 enclave 接口设计时,需要检查并限制指针如何在 enclave 接口中传递和使用.

3.3 敏感代码的自动化生成与安全检测

SGX 应用程序开发的关键是将应用程序划分为敏感代码与非敏感代码两部分,到底将哪些代码作为敏感代码放入 enclave 中目前还缺乏标准化的方法,并且划分的敏感代码是否安全和合理也缺乏相应的验证方法.总的来说,SGX 应用程序开发所面临的困难主要有以下 3 个方面.

- 开发工作量大:针对每一个应用程序进行开发,不仅需要开发人员具备丰富的开发经验和安全背景,同时也需要进行大量的开发工作;
- 通用性:目前需要开发人员针对每一个应用程序进行手动划分,缺乏通用性的解决方案.当前,SGX SDK 没有提供相关支持;
- 敏感代码划分的合理性和安全性:划分的敏感代码需要进行安全性和合理性验证,以确认划分出的敏感代码是否存在安全漏洞.

因此,如何对 enclave 的敏感代码进行自动化划分且保证划分结果的可靠性,是 SGX 应用程序开发需要解

决的问题.目前,相关工作分别实现了对程序敏感代码的自动化划分和对 enclave 代码的安全检查等.

3.3.1 敏感代码自动划分技术

文献[52]提出了一个用于开发 C 语言 SGX 应用程序的源码划分框架 Glamdring.通过开发人员对安全敏感应用程序数据的注释,Glamdring 可以自动地将应用程序划分为可信和不可信代码两部分,如图 7 所示,Glamdring 的操作分为 4 步.

(1) 代码注释:开发人员通过注释需要机密性和完整性保护的变量来提供有关安全敏感数据的输入和输出信息;

(2) 代码分析:Glamdring 基于带注释的源代码,使用数据流分析来识别可能处理敏感数据的函数,并使用向后切片来识别可能会间接影响敏感数据的函数.将直接或者间接访问或者影响敏感数据的代码标记为敏感代码.因此,Glamdring 获得了处理敏感数据或影响其完整性的最小代码集;

(3) 代码划分:接下来,Glamdring 创建一个划分规范(partition specification,简称 PS),该规范定义了必须由 enclave 保护的代码.PS 会根据程序分析枚举敏感的函数、内存分配和全局变量;

(4) 代码生成:Glamdring 使用源到源编译器(source-to-source compiler)基于 PS 将代码分为敏感和非敏感的代码,然后将敏感代码放置在 enclave 内,并在 enclave 边界自动添加运行时检查和加密操作,以保护其免受攻击.

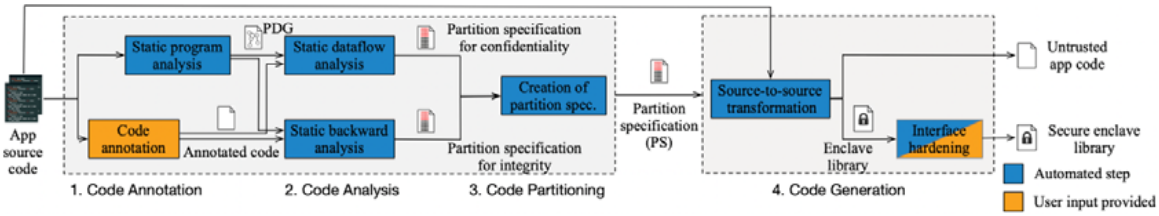

Fig.7 Overview of the Glamdring framework

图 7 Glamdring 架构

Glamdring 保护了敏感输入数据的机密性和敏感输出数据的完整性,基于最小特权原则,最小化访问敏感数据的代码,自动更改应用程序代码,极大地减少了程序开发人员的开发工作量,增强了 SGX 的易用性.Liu 等人则在其源码自动划分方案 PtrSplit[79]中,提出了一系列支持全局指针的方法.

3.3.2 Enclave 代码安全检测技术

SGX 可以为 enclave 内部代码和数据提供机密性和完整性保护,以抵御来自 enclave 外部的攻击.但是 enclave 代码本身的漏洞也可能会泄露机密,例如 enclave 可能受到 Heartbleed[80]之类的攻击,这些攻击已被证明会从内存中泄露秘密密钥.开发人员也必须采取必要的措施来抵御协议和应用程序攻击,包括正确使用 SGX 指令、使用安全的加密协议、避免由于违反内存安全性而造成错误等.

文献[53]开发了一个静态 enclave 代码安全性验证工具 Moat,用以验证 enclave 代码是否满足机密性要求,即是否存在将秘密泄露给敌手可见的非 enclave 内存的代码.Moat 分析了 enclave 二进制代码的指令级行为,采用信息流敏感类型检查算法自动验证 enclave 程序是否提供机密性保证.对于类型错误的程序,Moat 会利用漏洞程序证明机密可能会泄露给非 enclave 内存.Moat 为程序员编写安全的 enclave 提供了验证方法和工具支持.

3.4 潜在安全威胁的分析与防护

SGX 原则上能够提供安全的隔离计算环境,但实际上同样面临各种各样的安全威胁,比如侧信道攻击、内存攻击和回放攻击等.当前,学术界的相关工作也已经进行了证实[9−11,77].针对每类攻击,相关研究工作提出了一系列防御措施,但目前仍缺少通用的解决方案.因此,需要 SGX 应用开发人员在开发 SGX 应用程序时充分考虑相关的安全威胁,并采取一定的防御措施.

3.4.1 侧信道攻击和防护方案

SGX 面临的最大威胁是侧信道攻击.侧信道攻击的主要手段是通过攻击面获取数据,推导获得控制流和数据流信息,最终获取 enclave 的代码和数据的信息,比如加密密钥、隐私数据等.Wang 等人对相关的 SGX 侧信道攻击进行了详细的分析和总结[19].本文只介绍典型的攻击面,包括页表、缓存(cache)以及 CPU 内部结构,这几种侧信道攻击的方式如下.

- 基于页表的攻击:利用页表对 enclave 页面的访问控制权,设置 enclave 页面为不可访问.之后,任何访问都会触发缺页异常,从而能够区分 enclave 访问了哪些页面.按照时间顺序把这些信息组合,就能够反推出 enclave 的某些状态和保护的数据.该类攻击包括 controlled-channel 攻击[10]和 pigeonhole 攻击[11]等;
- 基于 Cache 的攻击.在 SGX 环境中,大部分基于 Cache 的侧信道技术仍然适用,例如文献[81]证明,SGX 易受 Cache-timing 攻击;
- 基于 CPU 内部结构的攻击.CPU 内部有大量的结构是在 enclave 和非 enclave 之间共用的,这就给侧信道攻击提供了大量的攻击面,例如文献[82]实现的侧信道攻击.

这些侧信道攻击有的需要从 SGX 设计的角度来解决,有的则可以在应用程序内部进行解决.更加合理地设计应用程序,可以减少攻击者能够收集的有效信息,极大地降低攻击者成功攻击的几率.SGX 现有的部分侧信道防御方法可以按照不同层次进行分类,主要包括:

- 软件层:应用程序可以进行自我防御,通过合理地设计应用程序,避免泄露过多的差异信息,使得攻击者难以成功.例如:像 Opaque[34]等分析平台通过内部混淆让攻击者无法获取相关的差异信息,有效地保护了 enclave 内部数据的安全性;Chandra 等人提出的方案[42]在训练机器学习模型时充分考虑到了侧信道攻击,将决策树的数据结构进行重新设计,保证敌手无法观察到差异信息,以避免应用程序泄露相关模型信息;ORAM[83]技术同样通过无差别的访问等方法避免泄露差异信息,从而有效地抵御 SGX 面临的侧信道攻击,但是目前在性能方面仍然需要加以提高;
- 硬件层:通过优化 SGX 硬件设计,可以避免某些侧信道攻击.例如,在 enclave 的切换过程中,CPU 清除共用的内部结构体信息,避免非 enclave 得到任何残留记录.同时还要注意一些细节,比如清除的时间也必须是稳定不变的[82].硬件层为每个 enclave 隔离出一个 cache,也是侧信道攻击的解决方案之一,虽然文献[84]已经进行了一些尝试,但仍然还未能彻底解决该问题.

3.4.2 内存攻击和防护方案

基于 SGX 的隔离执行,为应用程序在不可信平台上的运行提供了强大的安全保障.但是隔离执行不能保护程序免受内存攻击[72],这些攻击很普遍,特别是用非安全语言(如 C/C++)编写的应用程序.例如,Heartbleed[80]之类的内存安全攻击可能会使隔离执行的机密性和完整性保障完全失效.远程攻击者可以通过利用现有的程序漏洞来调用越界内存访问,以破坏内存安全性.然后,攻击者可以劫持程序控制流程或泄露机密数据[85].

所有内存攻击的基础条件是能够访问禁止访问的内存区域,因此,可以通过边界检查技术[86]限制 enclave 程序可访问的内存范围.具体地,为每个分配对象的边界维护数据结构用于边界检查,确认指针是否仍在边界范围内.但是该数据结构可能会变得很大,并且查找成本很高.文献[55]基于 SGX 设计了一种用于 enclave 的高效边界检查技术,可有效抵御来自于 enclave 内部漏洞的威胁,并且结合标记指针和紧凑内存布局来提高性能.

3.5 小 结

本节总结的以上 4 种 SGX 应用安全防护技术概括了目前常见的 SGX 应用安全增强方案,为了更加直观地表述每一种技术的优缺点和性能,本小节给出如下总结(见表 2).通过分析这 4 种常见的 SGX 应用安全增强技术的方案与优劣性,可以将这 4 种技术的研究总结为以下 3 个方面.

- 最小化 TCB 和最少化对外接口,都可以降低 SGX 应用面临的安全风险.但是两者之间存在一定的冲突,例如,要最小化 TCB 就难以提供 enclave 内部丰富的系统调用支持,难以做到最少化对外接口.目前缺少能够自动平衡 TCB 大小和对外接口的开发工具,主要还是依赖程序员的开发经验来设计 TCB 和 enclave 接口.同样,也缺乏能够兼顾两个方面的安全增强方案;

- 敏感代码自动划分工具能够减少程序员的工作量,但其划分结果的准确性和可靠性难以验证,而且目前也缺乏普遍认可的划分标准.如何更准确、更可靠和更细粒度地自动划分应用程序,是需要实现的目标;
- 开发人员设计 SGX 应用程序时应该充分考虑到侧信道攻击等未知安全威胁,根据不同的场景进行相应调整,以抵御可能面临的安全威胁.这需要开发人员具有足够的安全知识,当前还缺乏自动化支持工具以辅助开发人员实现对潜在安全威胁的抵御.

Table 2 Security enhancement solutions of SGX application

表 2 SGX 应用安全增强方案

方案类型	主要方案	优势	劣势	相关工作简称
最小化 TCB	(1) 尽可能少的代码放入 enclave; (2) 内部微容器	最小化内部潜在漏洞造成的威胁	(1) 支持功能相对有限; (2) 频繁的模式切换	VC3[33], SecureKeeper[21], PANOPLY[49]等
对外接口最少化	在 enclave 设置操作系统库或函数库,支持执行完整应用程序	(1) 减小攻击面; (2) 支持未修改应用程序; (3) 减少模式切换	(1) TCB 增大; (2) 面临安全风险增加	Haven[30],SCONE[35], SGXKernel[50], Graphene[51]等
敏感代码的自动生成与安全检测	(1) 应用程序自动划分; (2) enclave 安全分析	(1) 开发人员工作量少; (2) 可复用	(1) 缺少统一划分标准; (2) 划分结果难以验证	Glamdring[52], Moat[53]等
潜在安全威胁的分析与防护	(1) 抵御侧信道攻击; (2) 抵御内存攻击	从应用程序设计层面增强 SGX 应用程序的安全性	难以抵御所有攻击,只能尽可能地抵御潜在威胁	ORAM[82],Opaque[34], SGXBOUNDS[55]等

4 SGX 应用性能优化技术

SGX 性能一直是学术界和工业界关注和研究的重要方向之一,也是制约 SGX 应用的重要因素.对 SGX 性能进行优化是必不可少的.相关研究[21,35,56−58]分析了 SGX 潜在的性能问题,并提出了多种技术来优化 SGX 性能.本文总结了优化 SGX 性能相关的研究工作,将 SGX 性能开销的问题和相关原因概括为以下几个方面.

- 模式切换开销.模式切换即 enclave 切换,原因主要包括 ecall、ocall 和 AEX.模式切换出于安全原因,必须执行一系列检查和更新,包括 TLB(translation lookaside buffer)刷新,基于内存的 enclave 参数也必须在可信和不可信的内存之间进行复制,从而导致模式切换需要付出高昂的代价;
- Enclave 页替换开销.内存要求超过 EPC 大小的应用程序必须在 EPC 和未受保护的内存之间替换页面.EPC 页换出的代价高昂,因为它们必须在被复制到外部内存之前被加密并得到完整性保护.为了防止地址转换攻击,替换协议中断所有 enclave 线程并刷新 TLB.并且,页替换同样会导致 enclave 模式的切换.在 enclave 执行期间的页替换,由于额外的转换和加密操作而开销巨大;
- 内存加密开销.由于 MEE 必须加密和解密高速缓存线,会导致加密内存访问存在高昂的开销,enclave 代码也会由于写入内存和高速缓存未命中而产生开销.

针对上述开销原因,相应地,本文将目前SGX性能优化方案也总结为3种技术——模式切换性能优化技术、内存页替换优化技术和内存加密优化技术.

- 模式切换开销优化技术.模式切换需要付出高昂的代价,因此应尽量避免模式切换,可以通过异步调用[56−58]来加以避免;
- 页替换开销优化技术.页替换的开销很大,应尽量避免页替换,可以通过多种技术来减少页替换.如尽量将 enclave 保持得较小,通过 EPC 页面预加载以避免 enclave 内部发生页错误,在 enclave 内部使用替换的内存管理机制,如 STANlite[20]和 Eleos[58];
- 内存加密开销优化技术.内存加密开销是因 SGX 的内存加密保护机制所致,可以通过减少对 enclave 内存的不必要使用、使用节省空间的数据结构或将较小的数据块加载到 enclave 来进行优化.

4.1 模式切换性能优化技术

模式切换是 SGX 能够在 enclave 中执行代码的基本保障机制,但是模式切换也是 SGX 应用程序性能开销的主要原因,尤其是对于系统调用密集型 SGX 应用程序而言,导致的开销会达到几倍、几十倍甚至上百倍.只要

CPU 进入或退出 enclave,就会发生模式切换.模式切换开销的主要原因是复杂的状态保存和安全检查、保存和恢复 CPU 状态以及由刷新 TLB 引起的 TLB 缓存未命中.

SGX SDK[16]提供的用于进行跨 enclave 函数调用的标准方法 ecall 和 ocall 都会触发 enclave 上下文切换.文献[35,44,50]分别对 SGX 的 ecall 和 ocall 的性能开销进行了评测.任何 ecall 和 ocall 都会导致性能开销,因为硬件必须执行某些操作来维护 SGX 的安全保障.Enclave 代码还必须验证所访问数据的完整性,例如 ecall 的参数、ocall 的返回值和从不可信内存读取的数据.本文对评测结果进行了总结,具体见表 3.从表中总结结果来看,直接使用 SGX SDK 的边缘函数性能开销高达 8 000 到 17 000 个时钟周期(CPU cycle).Ecall 的开销大约为 8 000 个时钟周期,ocall 的开销大约为 10 000 个时钟周期,这相比于系统调用 150 个时钟周期来说是非常高的.因此,减少模式切换导致的开销变得非常重要.目前,避免或降低模式切换的技术主要有通过共享内存和专用线程避免切换、增加库操作系统来减少切换操作以及合理地设计 enclave 接口.

Table 3 SGX ecall and ocall performance evaluation

表 3 SGX ecall 和 ocall 性能测试

文章简称	硬件测试环境	Ecall median latency (cycles)			Ocall median latency (cycles)		
		Warm cache	Cold cache	Buffer transfer to/from/to&from	Warm cache	Cold cache	Buffer transfer to/from/to&from
HotCalls[56]	Core I7-6700k 4GHz	8 640	14 160	9861/11172/10827	8 314	14 160	9252/11418/9801
Eleos[58]	Skylake i7-6700 4-core	9 000	⊗	⊗	8 000	⊗	⊗
SGXKernel[50]	Core i7-6670HQ 2.6GHz	8 815	⊗	⊗	7 840	⊗	⊗
Switchless[57]	i7 4 cores 2.6GHz	9 154	⊗	⊗	8 344	⊗	⊗
SGX-perf[64]	Xeon E3-1230 3.40 GHz	5 850	⊗	⊗	⊗	⊗	⊗

文献[64]则在 3 种设置中直接测量了 EENTER 和 EEXIT 指令的时间开销:(1) 在未修改的支持 SGX 的处理器上,使用一个热高速缓存(warm cache)进行了一次往返,测得的切换时间约为 5 850 个时钟周期(约为 2 130ns);(2) SDK 更新修复 Spectre[87]漏洞补丁(patchs)后(该漏洞影响了 SGX[88]),测得的切换时间约为 10 170 个时钟周期(约为 3 850ns),是没有补丁情况下切换时间的 1.74 倍;(3) 更新修复 Foreshadow[89]漏洞的补丁后,在同时抵御 Spectre 和 Foreshadow 漏洞的情况下,enclave 切换变得更加缓慢,约为 13 100 个时钟周期(约 4 890ns),约 2.24 倍,这些测试结果进一步说明了避免模式切换的必要性.

4.1.1 无切换调用

为了最大限度地降低模式切换带来的性能损失,文献[35,56−58]分别提出了解决方案,这些方案具有相同的核心思想:调用线程将 ecall/ocall 的请求发送到共享的不可信缓冲区,工作线程则从该缓冲区异步接收和处理请求.由于采用这种方式调用 eall/ocall 不会触发模式切换,故称为无切换调用(switchless calls)[57].如文献[56]提出一个由请求者和响应者组成的体系结构 HotCalls,通过未加密共享内存进行通信,采用同步自旋锁机制,相比于 SGX 默认的接口加速 13~27 倍.

Eleos[58]则采用远程过程调用(remote procedure call,简称 RPC)的方式处理系统调用,RPC 机制允许在不退出 enclave 的情况下,将调用阻塞到不可信代码中执行,实际调用被委托给工作线程,该工作线程在 enclave 所有者进程的不可信内存区域中执行.由于 enclave 可执行多个线程,因此,Eleos 维护一个具有多个工作线程的线程池,通过位于不可信内存中的共享作业队列与 enclave 交互.线程池中的线程轮询队列,调用请求的函数,并通过不可信的共享缓冲区传回结果.因为 enclave 的线程不能使用互斥等标准的 OS 同步原语,可信和不可信上下文之间的同步是通过轮询执行的.为了降低轮询成本,Eleos 通过简单的 ocall 机制调用长时间运行的系统调用.

上述通过使用共享内存作为通信信道和专门设置线程轮询消息的方案被证明是有效的,但是这种无切换调用技术在效率方面存在疑问,用额外的 CPU 内核来减少模式切换未必总是合理的.无切换调用必须由工作线程执行,因此需要额外的 CPU 内核.实现的加速比随着工作量的减少而减小.在几乎空闲的工作负载的极端情况下,使用的额外 CPU 内核显然是浪费资源的.文献[56]建议:如果工作线程长时间没有收到任何请求,则将其置于休眠状态.但是,可能存在请求稀疏却不足以触发超时睡眠的情况;并且,是否唤醒休眠工作线程处理一个或两个请求然后再次睡眠也存在疑问,这种情况下,浪费 CPU 问题变得更加严重.工作线程使用额外 CPU 核心可能

已用于为同台机器上其他应用程序的线程.由于上述原因,enclave 应用程序会遇到各种动态工作负载,是否采用无切换调用则需要进一步加以分析.

Intel 的研究团队将无切换调用作为一种实用技术,确保它能够有效地提高性能[57].通过数学和模拟分析建立性能模型,研究无切换调用可以有效提高性能的条件,并设计了一种基于效率的调度算法,该算法可以自动调整工作线程数量以响应不断变化的工作量.Intel 公司目前已将无切换调用及其优化算法集成到 SDK,从 SDK 2.2 开始,正式提供这一功能.

4.1.2 增加操作系统库/函数库减少模式切换

SGX 提供了强大的安全保证,但 SGX 的一个限制是 enclave 内缺乏系统调用支持,这导致 SGX 在性能方面并不理想.为解决这一问题,Haven[30]、SGXKernel[50]和 Graphene[51]通过提供用于 enclave 执行的兼容层,为 SGX 内部的应用程序提供操作系统库支持,无需修改应用程序代码.

SGXKernel[50]在 enclave 内部提供了一个操作系统库,从而可以减少 enclave 切换,使应用程序不再需要进行频繁的模式切换.这种减少模式切换的设计是通过结合两个设计实现的:即异步进行 enclave 通信和可抢占的 enclave 内部多线程.Graphene[51]除了提供一个操作系统库支持之外,还提供了一些使 SGX 更加可用的改进措施,例如动态加载库的完整性支持以及安全的多进程支持,有效地减少了模式切换.

4.1.3 轻量级细粒度的并行减少 enclave 之间模式转换

文献[60]提出了一个针对 SGX 的框架 EActors,提出新的概念 actor,实现轻量级细粒度的并行,从而避免了 SGX SDK 提供的同步功能的高开销问题,解决了 enclave 内部和多个 enclave 之间执行模式转换的高开销问题.最后,EActors 提供了基于安全和性能要求的高度选择自由来执行可信或不可信的 actor.性能评估结果表明:基于 EActors 实现的安全消息传递服务与基于 SGX SDK 实现的版本性能提高了 40 倍,EActor 可以更加无缝、灵活和高效地使用 SGX,特别是对于需要多个 enclave 的应用程序.

4.2 内存页替换开销优化技术

Enclave 性能开销高的另一个重要原因是 enclave 内存空间比较小,尤其是应用程序本身可用工作空间比较小.Enclave 所在的 EPC 大小有限,当前 SGX v1 最大可设置为 128M,实际约 93M 可供应用程序使用,其他空间用于存储完整性保护的元数据[7].SGX 支持从 EPC 到主内存的页替换,以满足不适合 EPC 的 enclave.当程序代码数量和规模增大时,敏感页面会在内存中的 EPC 和非 EPC 区域之间频繁交换.Enclave 执行期间的页替换开销巨大(约 4 万时钟周期),因为它需要进行系统调用、页面复制、完整性树和元数据的更新等.文献[59]分析表明:页替换开销使系统平均减慢 5 倍,内存密集型应用的速度更慢,对 enclave 的性能有很大影响[21,35].

页替换会导致过高的性能开销,因此,enclave 设计应尽可能地避免遇到页替换.本文分析了页替换开销的原因以及相关的解决方案,具体如下.

- Enclave 太大可能是由于内部的数据集太大或数据处理不当所致.保持 enclave 较小以始终适合 EPC,可以避免发生页替换,如通过使用节省空间的数据结构或将较小的数据块加载到 enclave 来实现.但是 EPC 在所有运行的 enclave 之间共享,很难确定 enclave 大小是否合适,因为 EPC 可能已被其他 enclave 阻塞,从而无法避免页替换,尤其是在多租户的云平台中;
- EPC 空间太小的问题也可以通过增加 EPC 大小和增加并发线程来加以解决.鉴于大量的页替换开销,可以让 EPC 更大以提高其命中率.如直接提供更大的 EPC 空间或支持创建 enclave 后再进行扩展[90],VAULT[59]提供了支持 EPC 扩展的数据结构支持,但 enclave 超出 EPC 大小仍然会导致页替换;
- 在 enclave 内使用替换的内存管理机制,取代 SGX 采用的系统内存管理机制.例如,STANlite[20]和 Eleos[58],这些系统在 enclave 内部提供了新的内存管理机制,从而消除了 EPC 硬件页面故障和 enclave 退出导致的性能开销;
- 通过预加载页到 EPC 中来防止 enclave 执行期间发生页面错误.例如,可以通过在发出 ecall 之前加载所需的页面来实现.这样可以防止执行过程中 enclave 内部发生页面错误和 AEX.

4.2.1 高效的完整性验证结构减少分页开销技术

通过增加 EPC 的大小来匹配物理内存的大小,允许 EPC 容纳非敏感页面,可以有效地减少分页开销.Taassori 等人提出了解决方案 VAULT[54],他们首先分析了目前 EPC 最大只支持 128M 的原因,具体如下.

- 用于存储 EPC 元信息的完整性树深度和大小限制:完整性树的深度和大小随着受保护的内存的增大而增大.过大的树大小和深度导致可缓存性差以及带宽损失更高;
- 内存容量开销:每个 EPC 块所需要的完整性树和 MAC 可占据受保护内存的四分之一(约占 128MB 中的 32MB);
- 性能要求:由于 EPC 页面访问开销巨大,因此不适合非敏感页面.在设计时将大部分内存指定为 EPC,会导致敏感页面需求较少的应用程序不能充分利用 EPC 和更高的性能开销.

因此,为了支持启用更大的 EPC,必须解决至少两个重要问题:(1) 改进完整性树的深度及其可缓存性以保持内存容量和带宽开销较低;(2) 减少完整性验证的空间开销(包括完整性树和 MAC).

VAULT[59]首先引入了用于完整性验证的可变元统一加密叶子树(variable arity unified encrypted-leaf tree,简称 VAULT)计数器,如图 8 所示.VAULT 可以有效平衡管理树深度和计数器溢出问题.SGX 完整性树的参数(arity)为 8,VAULT 则将完整性树的 arity 设计为 16 到 64 可变,从而实现一个更紧凑且深度更低的 VAULT,每次读取和写入的可缓存性和带宽开销都得到了提高.并且,VAULT 提出了压缩共享 MAC(shared MAC with compression,简称 SMC)技术,用以压缩数据块并将其 MAC 打包到单个高速缓存行中,从而减少带宽开销,通过在多个数据块之间共享 MAC 来减少存储开销,并证明了基于压缩的技术可以在大多数情况下消除或减少带宽损失.因此,SMC 可以同时减少带宽和内存容量开销.最后,VAULT 仅为敏感页面按需分配 MAC,以进一步减少 MAC 容量开销.通过结合 VAULT 和 MAC 共享和压缩,VAULT 实现了扩展 EPC 到整个物理内存,并且可以解决 SGX 内存访问中的大多数低效问题,相对于 SGX,整体性能提高 3.7 倍,而内存空间开销仅为 4.7%.

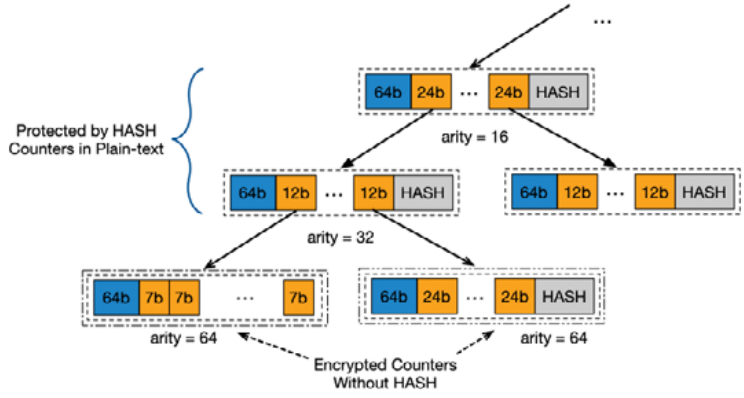

Fig.8 VAULT schematic diagram

图 8 VAULT 示意图

4.2.2 安全用户管理虚拟内存技术

Orenbach 等人提出了用于 enclave 的无退出(exit-less)运行时系统服务 Eleos[58],通过透明且安全地将系统调用委派给运行在一个应用程序线程中的 RPC 服务,实现了无退出系统调用,即无需退出 enclave.Eleos 包含 3 个部分(如图 9 所示):可信的运行时负责在 enclave 内提供 RPC 和 SUVM;在单独的应用程序线程中运行的不可信运行时负责处理 RPC 请求并与 SGX 驱动程序进行交互;SGX 驱动程序模块公开了用于在多个协调 SUVM 内存分配的接口.

Eleos 还提出一种安全用户管理虚拟内存(secure user-managed virtual memory,简称 SUVM)抽象,这种抽象通过引入 enclave 子页面访问粒度,消除了 EPC 硬件页面故障和 enclave 退出导致的性能开销.用户通过特殊的分配器,在 SUVM 中分配缓冲区,并获得安全指针;然后可以将其用作应用程序中的常规指针,指针访问被逐出

的 SUVM 页面会触发软件页面错误,并且完全在 enclave 内部进行处理.这些机制提高了 enclave 性能,且不会削弱 SGX 安全保证.Eleos 只在 TCB 中添加了几百行代码,与 SGX SDK 或 Graphene[51]等操作系统库(约 1 000 行代码)相比,Eleos 在 TCB 中添加的代码相对较少.

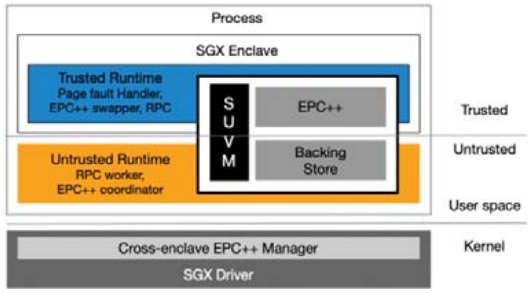

Fig.9 Eleos high-level design

图 9 Eleos 架构示意图

Sartakov 等人提出了在机架规模环境中进行安全数据处理的内存数据库引擎 STANlite[20].STANlite 通过在 enclave 内使用可扩展的虚拟内存引擎(virtual memory engine,简称 VME),替代了操作系统提供的内存页替换机制,并使其对内存具有完全控制权,从而避免巨大的模式转换开销.STANlite 在不需要考虑机密性的情况下,提供了一种纯粹的完整性保护数据管理模式,以提高性能.STANlite 还具有基于远程直接内存访问(remote direct memory access,简称 RDMA)、无切换和零拷贝的通信层,可在机架级环境中进行快速的远程数据库访问.

4.3 安全内存访问开销优化

文献[35]使用 SGX SDK 微基准测试评估了 enclave 内存访问开销.该基准测试了顺序读/写和随机读/写操作的时间开销,并与未部署 enclave 安全内存的访问时间进行了对比.所有操作总共处理 256MB 数据,但访问的内存区域大小不同.

如图 10 所示:只要访问的内存大小不超过 3 级(L3)缓存(8MB),enclave 内存访问开销就可以忽略不计.如果 L3 缓存未命中,随机存储器访问的性能开销可高达 12 倍左右.当访问的内存超出可用的 EPC 大小时,触发的页错误会导致 3 个数量级(1 000 倍)左右的开销.连续内存访问由于 CPU 预读取实现了更好的性能,这隐藏了一些解密开销,在 EPC 大小范围内几乎没有内存访问开销,而超出 EPC 大小的内存访问大约有 2 倍左右的性能开销.

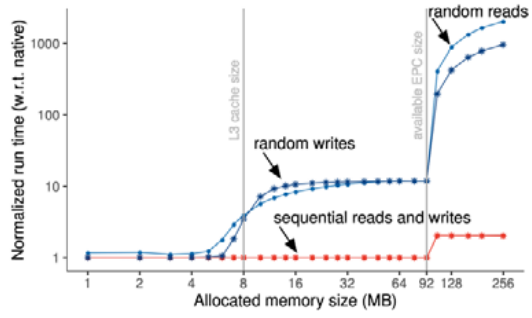

Fig.10 Normalized overhead of memory accesses with SGX enclaves

图 10 SGX encalve 内存的标准化开销

文献[56,58]也对 enclave 内存访问开销进行了测试,见表 4.文献[56]采用 SPEC[91]测试集进行测试,表明内存写操作的开销(30%~102%)要比内存读操作(6.5%~19.5%)高 5 倍左右,但是没有区分连续访问和随机访问操作.文献[58]则采用生成 10 万个随机数进行 enclave 内存访问开销测试,内存随机访问和连续访问以及读和写操作的开销相近,大约在 5.6 倍~9.5 倍开销之间.

Table 4 Performance evaluation of SGX memory read/write operation

表 4 SGX 内存读写操作性能测试

文章(系统)简称	测试用例	测试结果		
		内存读操作	内存写操作	内存读和写操作
SCONE[35]	SGX SDK 自带的 micro-benchmark	12~10^3 倍(随机访问)/ 1~2 倍(连续访问)	12~10^3 倍(随机访问)/ 1~2 倍(连续访问)	⊗
HotCalls[56]	SPEC 2006	6.5%~19.5%	30%~102%	⊗
Eleos[58]	生成 100 000 个随机数	5.6×(随机访问)/ 5.6×(连续访问)	8.9×(随机访问)/ 6.8×(连续访问)	9.5×(随机访问)/ 7.4×(连续访问)

以上结果表明,enclave 内存加密开销由 SGX 设计本身所致,难以作进一步的优化.出于性能原因,安全程序设计应尽量减少对 enclave 内存的使用和访问.例如,SCONE[35]通过将加密的应用程序数据(如缓存文件和网络缓冲区)存储在 enclave 之外,以减少 enclave 昂贵的内存访问.如果必须使用 enclave 内存,应尽量采用连续访问的数据结构,避免采用随机访问的数据结构.

4.4 小 结

本节总结的以上 3 种 SGX 性能优化技术概括了目前常见的 SGX 应用性能优化方案,为了更加直观地对比每一种技术的性能提升程度和对安全性的影响,本节作了如下总结(见表 5).

Table 5 SGX performance optimization solutions

表 5 SGX 性能优化方案

分类	方案类型	文章(系统)	测试应用	性能提升	安全性分析
模式切换性能优化技术	无切换调用	SCONE[35]	Apache,NGINX, Redis 和 memcached	增加 0.6~1.2 倍的吞吐量	TCB 增加 0.6~2 倍
		HotCalls[56]	memcached,open-VPN 和 lighttpd	(1) 吞吐量提高 2.6~3.7 倍; (2) 延迟降低 62%~74%	与 SGX 安全保证相同
		Switchless Calls[57]	fwrite, sgx_fwrite	(1) ecall 性能提升 5.9 倍; (2) ocall 性能提升 8.2 倍	与 SGX 安全保证相同
	增加操作系统库/函数库	Haven[30]	SQL Server 和 Apache	(1) 性能提升 31%~54%	TCB 增加 10^6 行代码
		SGXKernel[50]	Redis	(1) 性能开销降低 77% (2) 比 Graphene 快 7 倍	(1) 减少模式切换 (2) TCB 增加 70 000 行代码
		Graphene[51]	Apache,GCC 和 R 解释器	(1) 性能提升 1~2 倍	TCB 增加 5×10^5 行代码
	轻量级细粒度并行框架	EActor[60]	JabberD2 和 ejabberd	性能提升 1~40 倍	(1) TCB 仅增加 3 300 行代码 (2) 减少攻击面,增强了安全
内存页替换开销优化技术	高效的完整性验证结构	VAULT[59]	SPEC2k6,NPB	(1) 整体性能提高 3.7 倍; (2) 内存空间开销 4.7%	与 SGX 安全保证相同
	安全用户管理虚拟内存	Eleos[58]	Memcached 和面部验证	(1) 2.2~2.3 倍的吞吐量; (2) 处理的数据比 enclave 物理内存大 5 倍	与 SGX 安全保证相同
		STANlite[20]	microbenchmark, Speedtest1 和 TPC-C	响应时间提升 1.79 倍(micro)和 2.44 倍(TPC-C)	TCB 保持较小,仅包含必须的敏感代码

通过分析这 3 种常见的 SGX 应用性能优化技术的性能提升与安全性影响,可以将这 3 种技术的研究总结为以下 4 个方面.

- Enclave 的开销主要归结为模式切换、页替换和内存加解密开销.目前,主要的研究工作集中在对模式切换和页替换开销优化方面;
- 模式切换开销目前的主要解决方案是将同步执行改为异步执行,最大化 enclave 内部和外部的执行时间,从而减少频繁的模式切换导致的性能开销;
- 页替换开销目前的解决方案是,通过高效的数据存储结构来增大 EPC 和采用更高效的内存管理机制;
- Enclave 内存加解密开销是由于 SGX 本身的设计,目前主要通过合理使用加密内存来有效避免这些开销.要从根本上降低加密内存访问开销,需要从 SGX 设计层面来优化,如重新设计加解密算法等.

5 SGX 应用辅助开发技术

SGX 技术目前应用的场景越来越广泛,但是安全且简便地使用 SGX 并不容易,错误或者不合适的使用方式甚至会导致安全风险.因此,辅助用户更好地使用 SGX 技术,也是一个非常重要的需求.目前,SGX 应用辅助开发技术主要包括安全开发语言、辅助开发工具和通用的操作系统库和函数库等.

5.1 安全开发语言

通过使用 SGX SDK[16],开发人员可以创建加载到 enclave 的代码.虽然 SDK 简化了 SGX 的使用,但 SGX 所提供的编程支持有限,如不提供 enclave 内部交互的编程支持、要求将可信代码硬编码到 enclave 中,这限制了灵活性.开发 SGX 应用需要对程序进行改造,当程序规模比较大时,带来的编程成本非常高.为了方便用户进行 SGX 应用的开发并增强 SGX 的安全性,目前学术界和工业界提出了支持 SGX 应用开发的安全开发语言.

Rust SGX SDK[61]是百度安全实验室开发的一个 SGX Rust 语言开发工具包.通过将 Rust 语言和 SGX 技术相结合,开发人员可通过用 Rust 语言编写 SGX 应用程序.开发人员基于 Rust 语言可以快速开发出没有内存安全漏洞的 SGX 应用程序,即使在操作系统被恶意控制时也能提供强大的安全防护能力,避免敏感数据被窃取.

MesaPy[62]也是百度安全实验室开发的一个安全开源项目,是一个内存安全的 Python 实现.Python 包含超过 30 万行 C 代码,含有很多安全漏洞和隐患.MesaPy 基于 Python 编译器 PyPy,通过使用内存安全语言重写外部库和形式化验证保障代码的内存安全等方法,全面提升 Python 解释器的安全性,避免内存问题引发的高危安全漏洞.基于这些安全特性,MesaPy 也支持运行在 SGX 中,开发者可以使用 Python 轻松地开发 SGX 应用运行于可信运行环境中.

Ghosn 等人为增强 SGX 的可用性,提出了一种将可信执行环境(trusted execution environment,简称 TEE)集成到语言中的方案 Secured Routines[63].该工作扩展了 Go 语言,允许开发人员依靠编译器自动提取安全代码和数据.其原型 GOTEE 是基于 Go 编译器进行的扩展.GOTEE 为 TEE 提供了一种简单的编程模型,开发人员可以直接通过关键字定义敏感代码.

5.2 辅助开发工具

5.2.1 安全划分与检测工具

自动化安全开发和检测工具也是 SGX 应用辅助开发工具的重要类型,该类工具的主要目标是实现对应用敏感代码的自动化分析和划分,以及对 enclave 代码的安全性检测.本文第 3.3 节介绍了用于 SGX 应用程序的源码划分框架 Glamdring,可以基于开发人员对敏感数据的注释,自动地将应用程序划分为不可信和可信两部分.

SGX 为 enclave 内的代码和数据提供硬件保护,以抵御来自底层操作系统的攻击,但是应用程序本身中的漏洞(例如 SGX 指令使用不正确或内存安全错误)也存在泄露机密的风险.文献[53]基于自动定理证明和信息流分析,提出了一套 SGX 的使用规范,设计了 Moat 这一检测工具,通过在汇编语言层面对程序进行分析,从而检测应用程序是否存在泄露 SGX 区域中秘密信息的可能.

5.2.2 性能测试与优化工具

SGX 目前所提供的开发支持工具非常有限,这会导致耗时的反复实验开发和调试,并存在性能下降的风险.SDK 自带了一个调试器插件,可用于检查 enclave、设置断点等,但是该插件仅适用于使用 SDK 开发的应用程序.Intel 提供的性能分析器 VTune Amplifier[92]也可用于测试 SGX 应用程序(包括 enclave)性能.VTune Amplifier 支持一种称为热点(hotspots)的新分析类型,可用于分析 enclave 应用程序.热点是一段频繁执行的代码,例如循环主体,由类似于每条指令的总周期数或高速缓存未命中等指标来定义.开发人员可以使用热点的默认设置来深入了解 SGX 应用程序与热点有关的 enclave 函数,确定要进一步优化的代码.VTune 专注于代码片段的底层分析,未充分考虑 SGX 特性,不足以帮助开发人员编写高效的 enclave 域代码.

文献[64]介绍了一组用于 SGX 应用程序的动态性能分析的工具 sgx-perf,它可以对 enclave 中的性能关键事件进行细粒度分析,允许开发人员跟踪 enclave 执行并记录影响性能的关键事件,例如 enclave 切换和页替换.它通过隐藏 SGX SDK 的特定功能并重定向控制流来实现,分析记录的数据可以发现潜在的性能瓶颈.此外,

sgx-perf 提供了改进 enclave 代码和接口提高性能的建议.测试表明:使用 sgx-perf 提供的建议最多可以将应用程序的性能提高 2.66 倍,这有助于开发出性能良好的 SGX 应用程序.

5.3 系统库支持

相关工作中提供操作系统库和函数库来支持 enclave 内通用应用程序的执行,如 Haven[30]、SGXKernel[50]、Graphene[51]和 SCONE[35]等.本文已在上文中有所介绍,这里再进行一些简单梳理.Haven 将 drawbridge 系统库部署到 enclave 内部,支持在 enclave 内直接运行未修改的 Windows 应用程序.Graphene 则是在 enclave 中部署了一个支持 Linux 应用程序运行的操作系统库,可以实现在 SGX 上快速部署未修改的 Linux 应用程序.SGXKernel 同样也是在 enclave 内部署了一个支持 Linux 应用程序操作系统库,且其性能明显优于 Haven 和 Graphene. SCONE 则是在 enclave 中配置了标准 C 函数库[79]的修改版本,以支持重新编译的 Linux 应用程序.

5.4 小 结

本文将 SGX 应用辅助开发技术总结为安全开发语言、辅助开发工具以及系统库和函数库这 3 类(见表 6).通过分析总结上述研究工作,并结合目前 SGX 应用需求,本文分析了 SGX 应用辅助开发技术的发展方向.

- 开发语言目前已经支持 C/C++、Python、GO 语言,并且 GOTEE 已将可信的开发环境集成到了 GO 语言中,可以通过关键字直接定义敏感函数.相关工作都考虑到将内存安全功能集成到开发语言中,简便使用和更多的功能集成,是安全开发语言的发展方向;
- 辅助开发工具目前主要有 SGX 应用程序自动划分工具、enclave 机密性验证工具和程序性能测试工具,但是这些工具仍存在一些问题需要解决,如自动划分工具代码划分粒度不够细、缺乏划分标准等;
- 操作系统库和函数库目前主要有通用 Linux 系统库、Windows 系统库和标准 C 函数库,并且基本能够支持 90%以上的常用系统库和函数库 API.

Table 6 SGX application-assisted development techniques

表 6 SGX 应用辅助开发技术

分类	文章(系统)简称	主要工作/解决(改进)问题	可用性提升	支持的语言/库/特性等
开发语言SDK	Rust SGX SDK[61]	将 RUST 语言集成到 SGX	集成内存安全保障到 SGX	RUST 语言
	MesaPy[62]	内存安全的 Python 实现	避免漏洞引发的安全问题	Python 语言
	GOTEE[63]	将 SGX 集成到 GO 语言	直接用关键字指定安全代码	GO 语言
辅助开发工具	Glamdring[52]	自动应用程序划分	自动划分代码,减少工作量	保证尽可能小的 TCB
	Moat[53]	验证 enclave 程序机密性	降低了泄露机密的安全风险	提出一套 SGX 使用规范
	SGX-perf[64]	Enclave 性能测试工具	辅助开发性能良好的应用	测试 SGX 应用开销
通用操作系统库/函数库	Haven[30]	通用 Windows 操作系统库	支持未修改的 Windows 应用	抵御内部恶意代码攻击
	SGXKernel[50], Graphene[51]	通用 Linux 操作系统库	支持未修改的 Linux 应用	支持 94%以上的 Linux API
	SCONE[35]	标准 C 函数库的精简版	同上	支持容器(container)

6 SGX 应用功能扩展技术

SGX 的主要功能是创建可信执行空间,抵御特权软件的威胁,应用的主要场景是云平台.然而,目前 SGX 自身缺乏对虚拟机迁移和容器等云平台常用技术的支持.目前很多研究工作对 SGX 进行了功能扩展,使其支持常用的云平台技术.

6.1 支持虚拟机迁移

SGX 可以解决云计算中的信息泄露问题,但是现有的虚拟机管理器(virtual machine manager,简称 VMM)不提供启用 SGX 虚拟机(vitual machine,简称 VM)的高效管理操作,如实时迁移.随着 SGX 越来越多地部署到云平台中,内部运行 enclave 的 VM 失去了实时迁移的能力,而 VM 实时迁移在云计算中被广泛使用,例如用于负载平衡、容错.这是由于 SGX 硬件防止不可信的虚拟机管理程序或操作系统访问 enclave 的运行状态所致,而这是传统 VM 实时迁移所必需的.启用 SGX 的 VM 能够实时迁移不是一个简单的问题,也面临着如下困难.

- 加密内存页在目的主机解密问题:Enclave 绑定到单个物理 CPU,使用特定于 CPU 的唯一密钥加密,如果 VMM 以现有方式迁移 enclave,则由于密钥不匹配,另一主机无法从源主机解密 enclave 内存页;
- 抵御回放攻击(roll-back attack):Enclave 存储空间有限,有时需要将 enclave 内部的数据加密并换出 enclave,然后在需要处理时再换入 enclave 并解密.但这样会面临回放攻击,恶意系统使用旧版本的数据欺骗 enclave,enclave 需要使用硬件计数器来抵御;
- 持久状态的迁移:包括密封数据和单调计数器,前者存在数据丢失风险,后者破坏了 SGX 的安全保障.

Park 等人提出了一种支持 SGX VM 迁移的方法及其实现模型[65],基于 KVM(kernel-based virtual machine)实现了相关设计,为应用程序和支持 SGX 的 KVM 客户操作系统提供了 SGX 库,并为开发人员提供了 SDK,以便他们编写 enclave 代码时无需了解相关的迁移机制,例如控制线程.其系统性能开销可以忽略不计,迁移内部运行 64 个 enclave 的虚拟机,迁移的总时间增长了 4.7%,而停机时间仅增加了 3ms.

Chen 等人设计了一个基于软件的 VM 安全迁移机制,允许 enclave 迁移其运行状态[66].迁移过程需要 enclave 和不可信特权软件之间的合作.该机制在每个 enclave 中引入一个控制线程来帮助 VM 迁移,从内部安全地转存所有 enclave 的状态.对于 enclave 中的数据和 CPU 上下文等状态,控制线程将对它们进行加密,并在转存之前进行完整性保护;对于不能由软件直接访问的状态,例如由 CPU 维护的当前状态保存区域(CSSA),设计记账机制来推断 enclave 内的值,并通过两阶段检查点方案强化设计,利用远程证明和自我销毁来保证每个 enclave 实例在迁移后不会回放或生成多个实例,以此来防御回放攻击.该工作采用纯软件方法实现密封数据和单调计数器迁移,并且维持 SGX 的安全保证,最大限度地减少了开发人员的工作量,性能开销可以忽略不计.

Alder 等人则将 SGX 计数器作为持久状态,实现了相应的迁移方案[67],并设计和实现一个框架,实现具有持久状态 enclave 的迁移,同时保持 SGX 相同的安全性保证.该迁移方案解决了一些实际挑战,例如从 VM 访问 enclave,并将 enclave 迁移到目标计算机的性能开销限制在 12.3%以下,大大减少了 enclave 开发人员的工作量.

6.2 支持容器

基于容器的虚拟化技术变得越来越流行[93].许多多租户环境使用容器来实现应用程序的性能隔离,比如使用 Docker[37]来封装容器、使用 Docker Swarm[94]或 Kubernetes[95]进行部署.尽管硬件虚拟化得到了改进,但容器在虚拟机管理程序上仍然比虚拟机(VM)具有性能优势:启动时间更短、I/O 吞吐量更高且延迟也更低[96].但提供的安全属性比 VM 要弱,因为主机 OS 通常只使用软件机制进行隔离[97].现有的容器隔离机制主要保护环境免受不可信容器的访问,但是租户希望保护其应用程序数据的机密性和完整性,防止未授权方的访问,不仅来自其他容器,还来自更高权限的系统软件.攻击者会攻击虚拟化系统软件中的漏洞,或损害特权系统管理员的安全凭证[98].

SGX 可以保护应用程序代码和数据免受其他软件的访问,包括更高权限的软件.使用 enclave 可以保护容器不受更高权限软件的威胁.容器的应用程序可以在 enclave 内执行,以确保数据的机密性和完整性.但是,使用 SGX 设计安全容器机制面临着两个挑战:(1) 最小化 enclave 内 TCB 的大小,同时支持安全容器中的现有应用程序;(2) 保证安全容器的低性能开销.

文献[35]设计了一个用于 Docker 的安全容器环境 SCONE,基于 SGX 实现在安全容器中运行 Linux 应用程序,SCONE 内部架构如图 11 所示.

SCONE 的内部组件如下.

- 对主机 OS 系统调用的外部接口.SCONE 在将参数传递给应用程序之前执行完整性检查,并将所有基于内存的返回值复制到安全区内部来抵御攻击.为保护通过文件描述符处理的数据的完整性和机密性,SCONE 通过屏蔽(shield)来支持数据的透明加密和身份验证;
- *M*:*N* 线程模型.*M* 个 enclave 绑定的应用程序线程在 *N* 个 OS 线程之间进行多路复用.当应用程序线程发出系统调用时,SCONE 检查是否存在另一个可以唤醒并执行的应用程序线程,直到系统调用的结果是可用为止,从而避免了不必要的模式切换开销;
- 异步系统调用接口.异步系统调用通过使用共享内存来传递系统调用参数和返回值,并发出信号请求

执行系统调用实现.系统调用由 SCONE 内核模块中单独运行的线程执行.因此,执行系统调用时,enclave 内的线程不必退出;

- 与现有的 Docker 容器环境集成,并确保与标准 Linux 容器兼容.但是 Linux SGX 驱动程序和 SCONE 内核模块必须运行在主机 OS 上,因此,除了 Linux SGX 驱动程序以外,SCONE 不使用 SDK[16]中的任何功能.

SCONE 在保证安全容器的 TCB 较小的情况下,通过线程异步执行等操作实现了很低的性能开销,且对 Docker 是透明的,与普通容器类似.由于容器镜像通常由专家生成,因此,缺乏经验的用户可以从 SCONE 中受益,只要信任安全容器镜像的创建者即可.

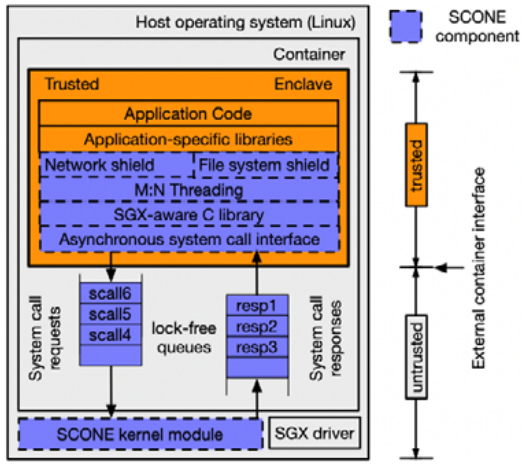

Fig.11 SCONE architecture

图 11 SCONE 架构图

在异构集群上部署和调度容器,混合使用具有和不具有 SGX 功能的计算机存在挑战.需要使用 SGX 的容器将会存在争夺 enclave 内存的可能性,因此,必须使用具有度量资源使用功能的调度管理器跟踪 SGX 内存请求并相应地调度容器.但是现有的容器协调器都没有提供有关的 SGX 资源使用运行时监控,只能依赖用户在部署时提供的静态信息.这些信息可能是错误的或不符合容器的实际使用,导致分配过多或分配不足.文献[68]提出了一种用于调度容器的 SGX 感知架构,并提供了该框架的开源实现,包括修改 SGX 的 Linux 驱动程序作为 Kubernetes 设备插件.

6.3 构建可信的外部路径

在本地应用环境中,用户可以使用 SGX 保护应用程序不被受到破坏的操作系统的影响.但是 SGX 缺乏对通用可信 I/O 路径的支持,以保护用户在 enclave 和 I/O 设备之间的输入和输出.文献[69]介绍了一种 SGX 通用可信路径架构 SGXIO,允许用户应用程序在不可信的操作系统上安全运行.SGXIO 将 SGX 编程模型的优势与传统的基于 hypervisor 的可信路径架构相结合,实现了支持通用 I/O 设备的可信路径.SGXIO 超越了云计算中的传统用例,使 SGX 技术可用于保护以用户为中心的本地应用程序,以防止内核级别的键盘记录程序等攻击手段的使用.

6.4 小 结

本节对 SGX 的相关功能扩展技术进行了总结和分析.当前,SGX 的功能扩展技术主要包括虚拟机迁移支持技术、容器支持技术和通用 I/O 可信路径支持技术.具体扩展的功能和优势见表 7.

通过总结和分析上述研究工作,并结合云平台等典型的计算场景,本文总结和分析了 SGX 应用功能扩展技术未来的研究方向.

- 目前,SGX VM 迁移主要基于软件模拟的方式实现,并不能完美地解决 SGX VM 迁移问题,VM 迁移等操作需要 SGX 设计和实现的进一步支持;
- 容器技术与 SGX 基本的结合问题得到了有效的解决,但在性能方面仍然有进一步优化的空间;
- SGX 技术在很多方面仍然缺乏一定的功能支持,尤其是在某些具体的应用场景,如多方计算场景下,需要多个 enclave 之间进行交互,而多个 enclave 之间的切换开销高昂,目前,SGX 缺乏有效的支持技术.

Table 7 Feature extension techiques for SGX application

表 7 SGX 应用功扩展技术

分类	文章简称	主要工作	功能扩展	优势
支持虚拟机迁移	Ref.[65]	提出支持迁移 SGX VM 的方法及实现模型	为应用程序和客户操作系统提供了 SGX 库	(1) 系统性能开销仅增长 4.7%; (2) 停机时间仅增加 3ms
	Ref.[66]	设计基于软件的 VM 迁移机制	允许 enclave 迁移运行状态,如密封数据和单调计数器	(1) 维持 SGX 的安全保证; (2) 减少了开发人员的工作量; (3) 可以忽略不计的性能开销
	Ref.[67]	实现具有持久状态 enclave 能够迁移的框架	实现 SGX 计数器的迁移方案	(1) 实现从 VM 访问专用存储区; (2) 保持 SGX 相同功能和安全性保证; (3) 性能开销不到 12.3%
支持容器	SCONE[35]	设计了用于 Docker 的安全容器环境	使用 SGX 在安全容器中运行 Linux 应用程序	(1) TCB 较小; (2) 性能开销较低; (3) 对 Docker 是透明的
	Ref.[68]	提出了用于调度容器的 SGX 感知架构	实现了在 SGX 异构集群上部署和调度容器	(1) 对整体性能几乎没有影响; (2) 提供开源实现
支持可信的外部路径	SGXIO[69]	介绍了 SGX 通用可信路径架构	实现支持通用 I/O 设备的可信路径	(1) 实现对本地应用程序的 I/O 保护; (2) 与未经修改的操作系统兼容

7 SGX 应用支持技术研究展望

SGX 技术的推出,为远程可信计算问题提供了解决方案.本文对 SGX 技术的研究现状及瓶颈问题进行了总结和归纳,如何完美地解决 SGX 技术应用的瓶颈问题,需要进一步的研究.本文从安全防护、性能优化和易用性这 3 个方面进行了分析.

(1) 安全防护:应用安全是 SGX 应用所面临的最基本的问题.如何有效地解决 SGX 应用所面临的侧信道攻击、内存攻击等安全威胁,增强 SGX 应用的安全性,依然是该技术应用研究的重要研究方向,具体的研究方向包括:SGX 代码运行时的安全性分析与检测、SGX 应用代码合理划分、抵御侧信道攻击等问题;

(2) 性能优化:性能开销高是制约 SGX 应用的重要瓶颈.如何减小 SGX 技术的内在开销,实现其性能优化的一般性解决方案,是后续的研究方向.具体的研究方向包括:通过硬件加速或可选加解密算法等方式降低 enclave 内存加解密开销、具备性能调优功能的自动化代码划分和更加高效的加密内存页管理方式等;

(3) 易用性:SGX 应用开发的困难性和功能的局限性也是制约其应用的重要原因.如何增强 SGX 技术的易用性和扩展 SGX 功能,是该技术研究的热点问题.具体的研究方向包括:对遗留应用中敏感代码自动化识别划分和安全性分析、避免重构代码支持原有程序直接运行的方案、更简洁易用的开发语言和动态可调整的库支持、对虚拟化等云平台常用技术的有效支持以及辅助用户开发安全和性能良好的 SGX 应用的开发工具等.

SGX 应用支持技术在学术界和工业界都引起广泛关注,我们认为,SGX 应用支持技术在未来的研究工作可能会偏重以下几个方面.

(1) 对硬件能力的扩展

SGX 目前的功能主要是基于 17 条指令来实现的,SGX 本身不支持虚拟机迁移,虽然基于软件模拟的方案实现了相关功能,但是更好的解决方案是对 SGX 的硬件能力进行扩展,提供支持相应操作的 SGX 指令及安全机

制.目前,SGX v1 可以支持的最大 EPC 为 128M,但仍然无法满足应用对 SGX 的需求,EPC 需要进一步地加以扩展.这其中,有一些挑战性问题需要解决,如高效的完整性验证结构以及无安全需求的代码对 EPC 的浪费和高开销问题.

(2) 对各种应用场景的有效支持

SGX 目前的主要应用场景是云计算,但是目前可信计算的需求越来越广泛,IoT、边缘计算等都需要可信支持.SGX 由于性能等限制难以直接应用到相关场景中,需要对 SGX 技术进行扩展和改进.目前,SGX 已经应用到的场景中,如多方计算,SGX 主要起到基本的安全隔离作用.但在多方计算场景下,应用本身交互频繁,SGX 的设计无法直接对这些场景提供有效的支持,性能开销很高.SGX 的密钥管理同样面临性能问题,在复杂场景和应用中,性能开销过高、实用性差,SGX 需要更加高效的密钥管理方案.

(3) 与区块链等技术结合扩展 SGX 能力

SGX 与区块链等新兴技术的结合,目前是 SGX 技术的研究热点之一.SGX 提供的可信环境可以为智能合约等提供区块链所不具备的机密性保证[99],可信的计算节点也可以支持区块链应用于更加复杂的应用场景[47].目前,利用 SGX 提高区块链安全性和性能的研究工作还有很多瓶颈问题需要解决,如不支持 enclave 内部智能合约之间的调用问题[47]、利用 SGX 的可信性替代区块链共识机制可信性的合理性问题等[47].SGX 可以为区块链提供机密性和可信性保证,同样,区块链技术也可以为 SGX 提供持久化存储和可信性验证方法,对计算结果进行记录和验证,从而实现可追溯等.这些都是对 SGX 非常重要的能力扩展,可以使其应用到更加广泛的场景.如何结合区块链等技术对 SGX 能力进行扩展,也是后续研究的重要研究方向.

(4) 如何解决 SGX 固有的信任问题

SGX 自公布以来在很多方面受到质疑,例如固件的不可审计、对 Intel 管理引擎(management engine)代码模块的依赖导致会受到安全漏洞的威胁以及必须信任 Intel 作为远程证明的前提.尤其是 SGX 必须使用 Intel 认证服务(Intel attestation service,简称 IAS)使其可信性受到广泛质疑,Intel 公司作为 SGX 技术远程证明服务提供方,用户也无法对其进行审计,因此很难完全信任 SGX.目前,Intel 公司在固件中实现的新特性 FLC(flexible launch control)可以实现开放第三方验证服务,一定程度上增强了 SGX 的可信性.但是,固件的不可审计和对 Intel 管理引擎代码模块的依赖,仍然没有得到有效解决.

8 结束语

本文对 SGX 应用支持技术的研究进展进行了深入分析和总结.首先介绍了 SGX 技术的相关机制和 SDK,分析了 SGX 技术应用现状及瓶颈问题的研究进展;接着,分别介绍了 SGX 应用安全防护技术、性能优化技术、辅助开发技术和功能扩展技术;最后,展望了 SGX 应用支持技术的未来研究方向.从我们的总结来看,SGX 应用支持技术仍然有一系列的挑战问题,未来有很多工作值得继续加以研究.期望通过我们的工作,能够给以后的研究者提供有益的借鉴与参考,为 SGX 技术的进一步应用和发展做出贡献.

致谢 在此,我们向对本文工作给予支持并给出宝贵建议的评审老师和同行表示衷心的感谢.

References:

[1] Amazon Web services. 2019. https://aws.amazon.com

[2] Microsoft azure. 2019. https://azure.microsoft.com

[3] Popa RA, Redfield C, Zeldovich N, *et al*. CryptDB: Protecting confidentiality with encrypted query processing. In: Proc. of the 23rd ACM Symp. on Operating Systems Principles. ACM, 2011. 85−100. [doi: 10.1145/2043556.2043566]

[4] Puttaswamy KPN, Kruegel C, Zhao BY. Silverline: Toward data confidentiality in storage-intensive cloud applications. In: Proc. of the 2nd ACM Symp. on Cloud Computing (SOCC 2011). ACM, 2011. 1−11. [doi: 10.1145/2038916.2038926]

[5] Tu S, Kaashoek MF, Madden S, *et al*. Processing analytical queries over encrypted data. VLDB Endowment. 2013,6(5):289−300. [doi: 10.14778/2535573.2488336]

[6] Gentry C. Fully homomorphic encryption using ideal lattices. In: Proc. of the 41st Annual ACM Symp. on Theory of Computing (SToC 2009). 2009. 169−178. [doi: 10.1145/1536414.1536440]

[7] Intel Corporation. Intel Software Guard Extensions (Intel SGX). Intel Labs, 2019. https://software.intel.com/sgx

[8] Intel Corporation. Intel Software Guard Extensions Programming Reference. Intel Labs, 2019. https://download.01.org/intel-sgx/sgx-linux/2.7.1/docs/ Intel_SGX_Developer_Reference_Linux_2.7.1_Open_Source.pdf

[9] Wang WH, Chen GX, Pan XR, *et al*. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2017). 2017. 2421−2434. [doi: 10.1145/3133956.3134038]

[10] Xu YZ, Cui WD, Peinado M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In: Proc. of the 2015 IEEE Symp. on Security and Privacy (SP 2015). 2015. 640−656. [doi: 10.1109/SP.2015.45]

[11] Shinde S, Chua ZL, Narayanan V, Saxena P. Preventing page faults from telling your secrets. In: Proc. of the 11th ACM on Asia Conf. on Computer and Communications Security (ASIA CCS 2016). 2016. 317−328. [doi: 10.1145/2897845:2897885]

[12] Costan V, Lebedev I, Devadas S. Secure processors part ii: Intel SGX security analysis and mit sanctum architecture. Foundations and Trends in Electronic Design Automation, 2017,11(3):249−361. [doi: 10.1561/1000000052]

[13] Neiger G, Santoni A, Leung F, *et al*. Intel virtualization technology: Hardware support for efficient processor virtualization. In: Proc. of the Computer. IEEE, 2005. 48−56. [doi: 10.1109/MC.2005.163]

[14] Intel SGX Developer Guide. 2019. https://download.01.org/intel-sgx/sgx-linux/2.7.1/docs/Intel_SGX_Developer_Guide.pdf

[15] McKeen F, Alexandrovich I, Berenzon A, *et al*. Innovative instructions and software model for isolated execution. In: Proc. of the 2nd Int'l Workshop on Hardware and Architecture Support for Security and Privacy (HASP 2013). ACM, 2013. [doi: 10.1145/2487726.2488368]

[16] Hoekstra M, Lal R, Pappachan P, *et al*. Using innovative instructions to create trustworthy software solutions. In: Proc. of the 2nd Int'l Workshop on Hardware and Architectural Support for Security and Privacy (HASP 2013). ACM, 2013. 11. [doi: 10.1145/2487726.2488370]

[17] Anati I, Gueron S, Johnson S, *et al*. Innovative technology for CPU based attestation and sealing. In: Proc. of the 2nd Int'l Workshop on Hardware and Architectural Support for Security and Privacy (HASP 2013). ACM, 2013. 13.

[18] Wang JW, Jiang Y, Li Q, Yang Y. Survey of research on SGX technology application. Journal of Network New Media, 2017,6(5): 3−9 (in Chinese with English abstract). [doi: CNKI:SUN:WJSY.0.2017-05-003]

[19] Wang J, Fan CY, Cheng YQ, Zhao B, Wei T, Yan F, Zhang HG, Ma J. Analysis and research on SGX technology. Ruan Jian Xue Bao/Journal of Software, 2018,29(9):2778−2798 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5594.htm [doi: 10.13328/j.cnki.jos.005594]

[20] Sartakov V, Weichbrodt N, Krieter S, *et al*. STANlite—A database engine for secure data processing at rack-scale level. In: Proc. of the 2018 IEEE Int'l Conf. on Cloud Engineering (IC2E 2018). IEEE, 2018. 23−33. [doi: 10.1109/IC2E.2018.00024]

[21] Brenner S, Wulf C, Goltzsche D, *et al*. Securekeeper: Confidential ZooKeeper using Intel SGX. In: Proc. of the 17th Int'l Middleware Conf. (Middleware 2016). ACM, 2016. 14. [doi: 10.1145/2988336.2988350]

[22] Goltzsche D, Rüsch S, Nieke M, *et al*. Endbox: Scalable middlebox functions using client-side trusted execution. In: Proc. of the 48th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN 2018). IEEE, 2018. 386−397. [doi: 10.1109/DSN.2018.00048]

[23] Shih MW, Kumar M, Kim T, *et al*. S-NFV: Securing NFV states by using SGX. In: Proc. of the 2016 ACM Int'l Workshop on Security in Software Defined Networks & Network Function Virtualization. ACM, 2016. 45−48. [doi: 10.1145/2876019. 2876032]

[24] Trach B, Krohmer A, Gregor F, *et al*. ShieldBox: Secure middleboxes using shielded execution. In: Proc. of the Symp. on SDN Research. ACM, 2018. 2. [doi: 10.1145/3185467.3185469]

[25] Pires R, Pasin M, Felber P, *et al*. Secure content-based routing using Intel software guard extensions. In: Proc. of the 17th Int'l Middleware Conf. (Middleware 2016). ACM, 2016. 1−10. [doi: 10.1145/2988336.2988346]

[26] Krawiecka K, Kurnikov A, Paverd A, *et al*. SafeKeeper: Protecting Web passwords using trusted execution environments. In: Proc.of the 2018 World Wide Web Conf. (WWW 2018). Int'l World Wide Web Conf. on Steering Committee, 2018. 349−358. [doi: 10.1145/3178876.3186101]

[27] Kim S, Han J, Ha J, *et al*. Enhancing security and privacy of tor's ecosystem by using trusted execution environments. In: Proc. of the 14th USENIX Symp. on Networked Systems Design and Implementation (NSDI 2017). 2017. 145−161.

[28] Karande V, Bauman E, Lin Z, *et al*. SGX-Log: Securing system logs with SGX. In: Proc. of the 2017 ACM on Asia Conf. on Computer and Communications Security. ACM, 2017. 19−30. [doi: 10.1145/3052973.3053034]

[29] Goltzsche D, Wulf C, Muthukumaran D, *et al*. Trustjs: Trusted client-side execution of javascript. In: Proc. of the 10th European Workshop on Systems Security (EuroSec 2017). ACM, 2017. 1−6. [doi: 10.1145/3065913.3065917]

[30] Baumann A, Peinado M, Hunt G. Shielding applications from an untrusted cloud with haven. ACM Trans. on Computer Systems (TOCS), 2015,33(3):8. [doi: 10.1145/2799647]

[31] Ahmad A, Kim K, Sarfaraz MI, *et al*. OBLIVIATE: A data oblivious filesystem for Intel SGX. In: Proc. of the 2018 Network and Distributed System Security Symp. (NDSS 2018). 2018.

[32] Hunt T, Zhu Z, Xu Y, *et al*. Ryoan: A distributed sandbox for untrusted computation on secret data. ACM Trans. on Computer Systems (TOCS), 2018,35(4):1. [doi: 10.1145/3231594]

[33] Schuster F, Costa M, Fournet C, *et al*. VC3: Trustworthy data analytics in the cloud using SGX. In: Proc. of the 2015 IEEE Symp. on Security and Privacy (SP 2015). IEEE, 2015. 38−54. [doi: 10.1109/SP.2015.10]

[34] Zheng W, Dave A, Beekman JG, *et al*. Opaque: An oblivious and encrypted distributed analytics platform. In: Proc. of the 14th USENIX Symp. on Networked Systems Design and Implementation (NSDI 2017). 2017. 283−298.

[35] Arnautov S, Trach B, Gregor F, *et al*. SCONE: Secure linux containers with Intel SGX. In: Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI 2016). 2016. 689−703.

[36] Apache Software Foundation. Hadoop. 2019. http://wiki.apache.org/hadoop/

[37] Merkel D. Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014,239:76−90. [doi: 10.1097/01.NND.0000320699.47006.a3]

[38] Microsoftware azure confidential computing with Intel SGX. https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions/microsoft-confidential-computing-sgx-video.html

[39] IBM data shield with Intel SGX. https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions/ibm-cloud-data-shield-sgx-video.html

[40] Porter DE, Boyd-Wickizer S, Howell J, *et al*. Rethinking the library OS from the top down. In: Proc. of the 16th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM, 2011,46(3):291−304. [doi: 10.1145/1950365.1950399]

[41] Ohrimenko O, Schuster F, Fournet C, *et al*. Oblivious multi-party machine learning on trusted processors. In: Proc. of the 25th USENIX Security Symp. (USENIX Security 2016). 2016. 619−636.

[42] Chandra S, Karande V, Lin Z, *et al*. Securing data analytics on SGX with randomization. LNCS, 2017. 352−369. [doi: 10.1007/978-3-319-66402-6_21]

[43] Shaon F, Kantarcioglu M, Lin Z, *et.al*. SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors. In: Proc. of the 2017 ACMSIGSAC Conf. on Computer and Communications Security. 2017. 1211−1228. [doi: 10.1145/3133956.3134095]

[44] Zhang F, Eyal I, Escriva R, *et al*. REM: Resource-efficient mining for blockchains. In: Proc. of the 26th USENIX Security Symp. (USENIX Security 2017). 2017. 1427−1444.

[45] Zhang F, Cecchetti E, Croman K, *et al*. Town crier: An authenticated data feed for smart contracts. In: Proc. of the 2016 ACMSIGSAC Conf. on Computer and Communications Security. ACM, 2016. 270−282. [doi: 10.1145/2976749.2978326]

[46] Das P, Eckey L, Frassetto T, *et al*. FastKitten: Practical smart contracts on Bitcoin. In: Proc. of the 28th USENIX Security Symp. (USENIX Security 2019). 2019. 801−818.

[47] Cheng R, Zhang F, Kos J, He W, Hynes N, Johnson N, *et al*. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: Proc. of the 2019 IEEE Symp. on Security and Privacy (SP 2019). IEEE, 2019. 185−200. [doi: 10.1109/EuroSP.2019.00023]

[48] Confidential computing cloud service. https://tech.antfin.com/docs/2/151322

[49] Shinde S, Le Tien D, Tople S, *et al*. Panoply: Low-TCB Linux applications with SGX enclaves. In: Proc. of the 2017 Network and Distributed System Security Symp. (NDSS 2017). 2017.

[50] Tian H, Zhang Y, Xing C, *et al*. SGXKernel: A library operating system optimized for Intel SGX. In: Proc. of the Computing Frontiers Conf. 2017. [doi: 35-44.10.1145/3075564.3075572]

[51] Tsai CC, Porter DE, Vij M. Graphene-SGX: A practical library OS for unmodified applications on SGX. In: Proc. of the 2017 USENIX Annual Technical Conf. (ATC 2017). 2017. 645−658.

[52] Lind J, Priebe C, Muthukumaran D, *et al*. Glamdring: Automatic application partitioning for Intel SGX. In: Proc. of the 2017 USENIX Annual Technical Conf. (ATC 2017). 2017. 285−298.

[53] Sinha R, Rajamani S, Seshia S, *et al*. Moat: Verifying confidentiality of enclave programs. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2015. 1169−1184. [doi: 10.1145/2810103.2813608]

[54] Shih MW, Lee S, Kim T, *et al*. T-SGX: Eradicating controlled-channel attacks against enclave programs. In: Proc. of the 2017 Network and Distributed System Security Symp. (NDSS 2017). 2017.

[55] Kuvaiskii D, Oleksenko O, Arnautov S, *et al*. SGXBOUNDS: Memory safety for shielded execution. In: Proc. of the 12th European Conf. on Computer Systems (EuroSys 2017). ACM, 2017. 205−221. [doi: 10.1145/3064176.3064192]

[56] Weisse O, Bertacco V, Austin T. Regaining lost cycles with HotCalls: A fast interface for SGX secure enclaves. ACM SIGARCH Computer Architecture News, 2017,45(2):81−93. [doi: 10.1145/3079856.3080208]

[57] Tian H, Zhang Q, Yan S, *et al*. Switchless calls made practical in Intel SGX. In: Proc. of the 3rd Workshop on System Software for Trusted Execution (SysTEX 2018). ACM, 2018. 22−27. [doi: 10.1145/3268935.3268942]

[58] Orenbach M, Lifshits P, Minkin M, *et al*. Eleos: ExitLess OS services for SGX enclaves. In: Proc. of the 12th European Conf. on Computer Systems. ACM, 2017. 238−253. [doi: 10.1145/3064176.3064219]

[59] Taassori M, Shafiee A, Balasubramonian R. VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. In: Proc. of the 23rd Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM, 2018. 665−678. [doi: 10.1145/3173162.3177155]

[60] Sartakov VA, Brenner S, Ben Mokhtar S, *et al*. EActors: Fast and flexible trusted computing using SGX. In: Proc. of the 19th Int'l Middleware Conf. (Middleware 2018). ACM, 2018. 187−200. [doi: 10.1145/3274808.3274823]

[61] Ding Y, Duan R, Li L, *et al*. POSTER: Rust SGX SDK: Towards memory safety in Intel SGX enclave. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2017). ACM, 2017. 2491−2493.

[62] MesaPy. 2019. https://github.com/mesalock-linux/mesapy

[63] Ghosn A, Larus JR, Bugnion E. Secured routines: Language-based construction of trusted execution environments. In: Proc. of the 2019 USENIX Annual Technical Conf. (ATC 2019). 2019. 571−586.

[64] Weichbrodt N, Aublin PL, Kapitza R. SGX-perf: A performance analysis tool for Intel SGX enclaves. In: Proc. of the 19th Int'l Middleware Conf. (Middleware 2018). ACM, 2018. 201−213. [doi: 10.1145/3274808.3274824]

[65] Park J, Park S, Oh J, *et al*. Toward live migration of SGX-enabled virtual machines. In: Proc. of the 2016 IEEE World Congress on Services (SERVICES 2016). IEEE, 2016. 111−112. [doi: 10.1109/SERVICES.2016.23]

[66] Gu J, Hua Z, Xia Y, *et al*. Secure live migration of SGX enclaves on untrusted cloud. In: Proc. of the 47th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN 2017). IEEE, 2017. 225−236. [doi: 10.1109/DSN.2017.37]

[67] Alder F, Kurnikov A, Paverd A, *et al*. Migrating SGX enclaves with persistent state. In: Proc. of the 48th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN 2018). IEEE, 2018. 195−206. [doi: 10.1109/DSN.2018.00031]

[68] Vaucher S, Pires R, Felber P, *et al*. SGX-aware container orchestration for heterogeneous clusters. In: Proc. of the 38th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS 2018). IEEE, 2018. 730−741. [doi: 10.1109/ICDCS.2018.00076]

[69] Weiser S, Werner M. SGXIO: Generic trusted I/O path for Intel SGX. In: Proc. of the 7th ACM on Conf. on Data and Application Security and Privacy. ACM, 2017. 261−268.

[70] Misra SC, Bhavsar VC. Relationships between selected software measures and latent bug-density: Guidelines for improving quality. In: Proc. of the Int'l Conf. on Computational Science and Its Applications. Springer-Verlag, 2003. 724−732.

[71] Shen VY, Yu TJ, Thebaut SM, *et al*. Identifying error-prone software-an empirical study. IEEE Trans. on Software Engineering, 1985,SE-11(4):317−324. [doi: 10.1109/TSE.1985.232222]

[72] Larsen P, Homescu A, Brunthaler S, *et al*. SoK: Automated software diversity. In: Proc. of the 2014 IEEE Symp. on Security and Privacy. IEEE, 2014. 276−291. [doi: 10.1109/SP.2014.25]

[73] Saltzer JH, Schroeder MD. The protection of information in computer systems. Proc. of the IEEE, 1975,63(9):1278−1308. [doi: 10.1109/PROC.1975.9939]

[74] Hunt P, Konar M, Junqueira FP, *et al*. ZooKeeper: Wait-free coordination for Internet-scale systems. In: Proc. of the USENIX Annual Technical Conf. 2010. [doi: 10.1111/j.1468-0262.2007.00765.x]

[75] Checkoway S, Shacham H. Iago attacks: Why the system call API is a bad untrusted RPC interface. ACM SIGPLAN Notices, 2013, 13:253−264. [doi: 10.1145/2499368.2451145]

[76] Howell J, Parno B, Douceur JR. How to run POSIX apps in a minimal picoprocess. In: Proc. of the USENIX Conf. on Technical Conf. 2013. 321−332.

[77] Weichbrodt N, Kurmus A, Pietzuch P, Kapitza R. AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves. In: Proc. of the 21st European Symp. on Research in Computer Security (ESORICS 2016). 2016. [doi: 10.1007/978-3- 319-45744-4_22]

[78] Jiao XJ. Potential security issue: Ecall SSL write using user check. 2018. https://github.com/lsds/TaLoS/issues/13

[79] Liu S, Tan G, Jaeger T. PtrSplit: Supporting general pointers in automatic program partitioning. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2017). ACM, 2017. 2359−2371. [doi: 10.1145/3133956. 3134066]

[80] Durumeric Z, Li F, Kasten J, *et al*. The matter of heartbleed. In: Proc. of the 2014 Conf. on Internet Measurement Conf. ACM, 2014. 475−488. [doi: 10.1145/2663716.2663755]

[81] Götzfried J, Eckert M, Schinzel S, Müller T. Cache attacks on Intel SGX. In: Proc. of the 10th European Workshop on Systems Security (EuroSec 2017). 2017. 1−6. [doi: 10.1145/3065913.3065915]

[82] Lee S, Shih MW, Gera P, Kim T, Kim H, Peinado M. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In: Proc. of the 26th USENIX Security Symp. (USENIX Security 2017). 2017. 557−574.

[83] Goldreich O, Ostrovsky R. Software protection and simulation on oblivious RAMs. Journal of the ACM (JACM), 1996,43(3): 431−473. [doi: 10.1145/233551.233553]

[84] Costan V, Lebedev I, Devadas S. Sanctum: Minimal hardware extensions for strong software isolation. In: Proc. of the 25th USENIX Security Symp. (USENIX Security 2016). 2016. 857−874.

[85] Lee J, Jang J, Jang Y, *et al*. Hacking in darkness: Return-oriented programming against secure enclaves. In: Proc. of the 26th USENIX Security Symp. (USENIX Security 2017). 2017. 523−539.

[86] Akritidis P, Costa M, Castro M, *et al*. Baggy bounds checking: An efficient and backwards-compatible defense against out-of-bounds errors. In: Proc. of the 18th USENIX Security Symp. (USENIX Security 2009). 2009. 51−66.

[87] Kocher P, Horn J, Fogh A, *et al*. Spectre attacks: Exploiting speculative execution. In: Proc. of the 2019 IEEE Symp. on Security and Privacy (SP 2019). IEEE, 2019. 1−19. [doi: 10.1109/SP.2019.00002]

[88] Chen G, Chen S, Xiao Y, *et al*. SGXpectre: Stealing Intel secrets from SGX enclaves via speculative execution. In: Proc. of the 2019 IEEE European Symp. on Security and Privacy (EuroS&P). 2019. 142−157. [doi: 10.1109/EuroSP.2019.00020]

[89] Van Bulck J, Minkin M, Weisse O, *et al*. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In: Proc. of the 27th USENIX Security Symp. (USENIX Security 2018). 2018. 991−1008.

[90] Xing BC, Shanahan M, Leslie-Hurd R. Intel software guard extensions (Intel SGX) software support for dynamic memory allocation inside an enclave. In: Proc. of the Hardware and Architectural Support for Security and Privacy 2016 (HASP 2016). ACM, 2016. 1−9. [doi: 10.1145/2948618.2954330]

[91] Henning JL. SPEC CPU2006 benchmark descriptions. ACM SIGARCH Computer Architecture News, 2006,34(4):1−17. [doi: 10.1145/1186736.1186737]

[92] Intel VTune Amplifier. 2019. https://software.intel.com/en-us/vtune

[93] Soltesz S, Pötzl H, Fiuczynski ME, *et al*. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. ACM SIGOPS Operating Systems Review (SIGOPS OSR), 2007,41(3):275−287. [doi: 10.1145/1272996.1273025]

[94] SWARM. 2016. https://docs.docker.com/engine/swarm/

[95] KUBERNETES. 2016. http://kubernetes.io/

[96] Felter W, Ferreira A, Rajamony R, *et al*. An updated performance comparison of virtual machines and linux containers. In: Proc. of the 2015 IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS 2015). IEEE, 2015. 171–172. [doi: 10.1109/ISPASS.2015.7095802]

[97] Gao X, Gu Z, Kayaalp M, *et al*. ContainerLeaks: Emerging security threats of information leakages in container clouds. In: Proc. of the 47th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN). 2017. 237–248. [doi: 10.1109/DSN.2017.49]

[98] Zetter K. NSA hacker chief explains how to keep him out of your system. Wired, 2016. https://www.wired.com/2016/01/nsa-hacker-chief-explains-how-to-keep-him-out-of-your-system/

[99] UCloud. https://www.ucloud.cn/

附中文参考文献:

[18] 王进文,江勇,李琦,等.SGX 技术应用研究综述.网络新媒体技术,2017,6(5):3–9.

[19] 王鹃,樊成阳,程越强,赵波,韦韬,严飞,张焕国,马婧.SGX 技术的分析和研究.软件学报,2018,29(9):2778–2798. http://www.jos.org.cn/1000-9825/5594.htm [doi: 10.13328/j.cnki.jos.005594]

董春涛(1991－),男,博士生,主要研究领域为分布式系统安全,云计算,大数据安全,可信计算.

沈晴霓(1970－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为操作系统,虚拟化安全,云计算,大数据安全与隐私,可信计算.

罗武(1994－),男,博士生,主要研究领域为操作系统,虚拟化安全,云计算,可信计算,浏览器安全.

吴鹏飞(1994－),男,博士生,主要研究领域为分布式系统安全,隐私保护,大数据安全.

吴中海(1968－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为大数据系统与分析,大数据与云安全,嵌入式系统.