系统安全隔离技术研究综述

郑显义 史 岗 孟 丹
(中国科学院信息工程研究所 北京 100093)
(中国科学院大学 北京 100049)

摘 要 随着网络技术的迅速发展和系统功能的日益复杂,系统越来越需要一个可以信赖的计算环境来保证敏感信息的安全性、完整性和可靠性.系统不仅需要保证敏感应用程序自身代码的安全,而且要保证其执行过程的隔离性以确保程序执行的操作和结果不会被攻击和窃取.尽管近几年在系统安全研究方面有着显著的进步,然而,损坏系统内核的攻击仍引起很大的威胁.这类攻击能访问系统的敏感数据,隐藏恶意行为,提高恶意进程的权限,改变系统行为,甚至控制整个系统.传统地,系统保护是通过使用与内核一样运行在同样地址空间和权限级别的安全机制实现的.然而,这种途径不足够安全,因为攻击者一旦成功损坏内核随后也将能损坏这些安全机制.为了实现真正的内核和关键数据保护,安全机制应被进行隔离保护,为此在系统中构建一个可信的隔离运行环境对系统安全是至关重要的.该文首先对各种安全隔离技术进行了整体概述,重点对各自的实现机制和系统架构做了深入分析,紧接着探讨了安全隔离技术在解决系统安全问题方面的研究现状,并在此基础上分析了其各自的优势与存在的不足,并将它们做了对比分析,最后结合当前信息安全领域存在的突出问题展望了安全隔离技术未来的发展方向和应用需求.

关键词 系统架构;可信执行环境;系统安全;安全隔离;虚拟化技术
中图法分类号 TP302 **DOI 号** 10.11897/SP.J.1016.2017.01057

A Survey on System Security Isolation Technology

ZHENG Xian-Yi SHI Gang MENG Dan
(*Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093)
(*University of Chinese Academy of Sciences, Beijing* 100049)

Abstract Due to the rapid development of network technology and the increasing complexity of system function, the system security is becoming more and more important. Therefore, a trusted computing environment is intensively needed to ensure the security, integrity and reliability of sensitive information. The system not only needs to ensure the security of the sensitive applications itself, but also it is necessary to ensure the isolation of the execution process to protect the operations and results from being attacked. The attacks of compromising kernel still cause a great threat, although there is a remarkable progress in the study of system security in recent years. This kind of attack can access the sensitive data of the system, hide malicious behavior, improve the authority of the malicious process, change the system behavior, and even control the whole system. Traditionally, the system protection is implemented by using security mechanisms operating at the same address space and level of authority as well as the kernel. However, this approach is not enough safe, because these security mechanisms can be damaged lately if the attacker has damaged the kernel successfully. In order to realize the protection of authentic kernel and key data, the security

收稿日期:2016-02-29;在线出版日期:2016-10-29.本课题得到国家"八六三"高技术研究发展计划项目基金(2012AA01A401)和国家"核高基"科技重大专项基金(2013ZX01029003-001)资助.**郑显义**,男,1986 年生,博士研究生,主要研究方向为计算机体系结构、网络与系统安全.E-mail:zhengxianyi@iie.ac.cn.**史 岗**,男,1974 年生,博士,正高级工程师,主要研究方向为计算机体系结构和嵌入式系统.**孟 丹**,男,1965 年生,博士,研究员,中国计算机学会(CCF)高级会员,主要研究领域为计算机体系结构、云计算及网络与系统安全.

mechanism should be isolated and protected, so an isolated operating environment is essential. First of all, we summary kinds of secure isolation technologies as a whole and make the in-depth analysis of their implementing mechanisms and system architecture. Then we discuss the research status of solving the existing system security problems based on these security isolation technologies in further. And on the basis of this we analyze their respective advantages and disadvantages of these isolation technologies and make a comparative analysis of them on this basis of the analysis. Finally, we look forward to future development direction and application requirements of the secure isolation technology combing with the outstanding problems that exist in the field of information security.

Keywords system architecture; trusted execution environment; system security; security isolation; virtualization technology

1 引 言

随着互联网的快速发展,使得用户获取应用软件的方式越来越依赖于网络资源,致使大量无法验证其来源的软件存在于当前系统中,这无疑给系统引入了许多潜在的不安全因素,系统安全问题已在设计中备受关注[1].而用户在设备上访问高级安全服务和敏感应用的交易已经越来越普遍,这样就需要设备能够安全访问、存储、操纵和传送有价值的金融信息和个人敏感信息.因此,如何保证终端设备及其上应用和服务的安全变得尤为重要.

通过核查 PC 端和移动终端的病毒感染机理显示当前设备已经非常复杂而不能保证足够的安全,可通过各种方式向系统注入攻击,如网络协议、内核系统调用接口或外置存储卡的使用等,这些都会使系统遭到恶意攻击[2].为了解决这个问题,系统设计者通常在系统中添加安全功能来保证更多安全软件系统的部署,如特定的加密算法和安全协议.另外,针对当前处理器和芯片集硬件,设计厂商也能做更多工作,而不是完全信任软件本身,如利用 TPM(Trusted Platform Module)度量软件保证原始软件未被恶意篡改;将设备密钥和可信根存储在芯片处理器内部的内存,这样使它们免于受到离线攻击,如冷启动(Cool Boot)攻击;利用 MMU(Memory Management Unit)并增加一些新寄存器能很好地实现安全域隔离和新的访问控制原语.然而,这些纯软件或纯硬件的安全措施在安全设计方面都暴露出各自的不足.设计者应从处理器设计整个过程都要考虑安全问题,使安全理念贯穿整个系统设计过程中.同时,设计者也需充分衡量诸如成本、性能和功耗等指标,才能实现真正意义上的系统安全[3].

且日益严峻的网络与系统安全问题对传统信息安全技术提出了严峻的挑战,存在于系统内核中的传统保护机制已经难以满足当前系统的安全需求且自身安全性也会因为内核的损害而难以得到足够保证.目前提高计算设备安全性的一个比较行之有效的方法是创建软件隔离层,系统某一层的安全问题不至于影响其他层的安全,如利用沙箱来隔离应用程序.这样,即使存在恶意行为的应用也不会影响到系统中其它应用的安全执行.虚拟机(Hypervisor/VMM)就是通过引入另外一个高权限的隔离软件层来进一步扩展了这个思想,它能隔离多个操作系统.从以上分析可以看出,安全隔离技术是确保系统安全与可靠的一种非常重要的手段,它可防止不同的系统组件之间相互干扰而导致的威胁.为此,在系统中构建一个安全隔离的运行环境,使其既能够确保敏感应用的正常执行,保护隐私数据和敏感信息,还能够检测、监控和防护系统的恶意行为,从而提高系统抵御安全威胁的能力.总之,在保障系统安全方面安全隔离技术已经成为一种趋势,能够从一定程度上打破当前系统安全存在的瓶颈,它已然成为一种重要的系统安全技术.

本文首先对各种安全隔离技术进行了整体概述,并对其实现机制进行了深入分析,然后分析总结了以上各种隔离机制的研究现状,接着基于此对各种安全隔离机制的优势与不足进行了深入分析阐述,并剖析了基于这些机制在解决系统安全问题方面的优势与不足之处,也将各类安全机制通过抽象 3 个关键指标作了整体对比分析,最后基于当前系统存在的安全问题展望了安全隔离技术未来的发展方向及应用研究情况.

2　安全隔离技术概述

随着网络的广泛运用，系统中的恶意攻击已经越来越普遍，为此系统针对关键信息保护和系统安全检测防御引入了各种安全机制，包括访问控制、病毒检测及沙箱等. 不幸的是，一方面现有各种不可信软件中隐藏的恶意代码已对这些安全机制造成很大的威胁；另一方面，如今庞大的商业操作系统，如 Windows，Mac OS X，Linux 及 FreeBSD 等，都因为功能丰富存在大量的漏洞，这些漏洞极易被攻击者利用，而系统中又缺少足够的保护措施来保护内核中设计的安全机制的安全[4]. 因此，为了限制不可信软件和不可避免的系统漏洞对系统可能造成的损害，研究者和企业都纷纷引入安全隔离机制，利用其提供的安全隔离运行环境来维护代码的完整性，保护各种安全机制，并检测和防护系统的安全. 根据系统中隔离机制所实现的方式将其归纳为以下三类：硬件隔离技术、软件隔离技术及系统级隔离技术.

早在 1973 年 Lampson 就认识到了隔离这一概念，并且早期的计算机结构，如 Multics 和 Cambridge CAP 已经使用硬件特性实现了地址隔离. Salzer 和 Schroeder 也在 1975 年将这类硬件概述为计算机保护的“technical underpinnings”，并基于一位特权位实现了内存访问限制，将它作为隔离虚拟机的基础构建块. 随后，组件隔离成为系统一项基本的安全策略，也是实现更高级别的系统安全策略的基础. 处理器有保护内存的硬件（如 MMU（Memory Management Unit）），操作系统或管理程序可以利用这些硬件组件和自身的软件技术，在软件组件之间实现一种隔离策略. 如操作系统内核必须与驻留的应用程序隔离，这样操作系统就可以控制和实施 I/O 资源的访问控制策略. 如果没有这种隔离，一个恶意的应用程序可以破坏内核，进而阻止内核运行任何其它的安全服务或者窃取其中的安全敏感信息[5-6].

如今，我们发现大量硬件隔离设计成为计算设备的一部分用来服务于安全敏感的操作和储存敏感信息. 使用最广泛的是智能卡，它在移动网络中作为用户身份的标识，也作为信用卡的安全组件，并作为物理或网络访问时各种各样的认证需求. 另一个部署隔离的计算设备是经典的 IBM-4758 加密协处理器，它被广泛地使用在银行应用中. 这些协处理器存在于通用计算机中，其非易失性存储被隔离在可以防止篡改的空间内.

安全隔离技术最主要的是将安全隔离环境与通用操作系统 ROS（Rich Operating System）实现隔离，这样才能保证隔离运行环境能够抵御来自 ROS 中特权代码的恶意攻击，实现安全隔离. 虽然安全隔离环境中的软件有访问 ROS 资源的权限，然而这些软件在未经授权的情况下也不能随意改变普通执行环境下 ROS 的运行状态. 其次，该隔离环境的部署应该尽量不修改 ROS 系统和应用程序代码，从而保证隔离环境具备较好的可移植性和适用性. 另外，需要保证隔离运行环境自身的性能负载，即在该环境中运行的软件和其自身的性能开销对 ROS 系统的性能影响不大，以保证隔离环境的可用性. 与此同时，要保证隔离运行环境行为可监控性，即在该环境中运行的软件行为可被该环境监控，它能够检测到恶意软件修改代码和数据，从而为安全隔离的防护机制提供防护依据. 为了能够利用安全隔离环境来检测和分析普通执行环境下 ROS 系统的恶意行为，该环境也必须具备获取 ROS 语义信息的能力[1].

隔离策略也具有许多优势，如有效地在一个系统中部署加密服务，像 Internet 协议安全性（Internet Protocol Security，IPSec）或加密文件系统这样的应用，它们都采用加密技术来实施安全策略，并需要依赖加密子系统（特别是私有密钥）与其他应用程序或子程序的隔离. 没有这种隔离，私有密钥一旦泄露，用于动态数据（传输的数据，其安全依赖于网络安全协议）和静态数据（其安全依靠存储介质加密）保护的保密性策略就会完全失效. 另外，隔离可以有效降低安全失败带来的损失，如一个文件系统崩溃，无法提供文件服务，但一个独立的网络堆栈可以继续提供通讯服务，因此可以认为降低损失是一种安全策略. 系统的敏感操作和关键数据也都可以在隔离环境中处理而确保其自身的安全性.

2.1　硬件隔离技术

为了保证系统中敏感信息的安全，设计者考虑通过设计专用的安全硬件模块来提供一个相对安全的硬件隔离环境，利用硬件来实施访问控制，运行软件一般难以绕过这种隔离机制. 这样，可以将系统中的关键数据、密钥或加解密服务存储在该模块中，且限制其它非法软件的访问. 这种类型的隔离一般由处理器或与主处理器连接的专用设备提供. 大多数处理器提供 MMU（Memory Management Unit）来分配不同的虚拟地址给不同的进程，这样来进行进

程隔离. 类似地，IOMMUs(IO Memory Management Units)转换设备 DMA 地址到物理地址，一个 IOMMU 能限制设备仅仅能访问其得到授权的那部分内存. 操作系统利用 IOMMUs 来隔离设备驱动，虚拟机利用 IOMMUs 来限制硬件对虚拟机的直接访问.

目前实现硬件隔离比较主流的方案有两种：一种方案是在芯片设计时在 SoC 外设计一个专门的硬件安全模块；另一种方案是在芯片设计时在 SoC 内部集成一个硬件安全模块. 其中，第一种应用比较广泛的是手机中的 SIM 卡和智能卡. 这里以智能卡为例进行说明，它和加密协处理器一般划分为同一个安全级别并可防止物理篡改. 仅仅智能卡或协处理器标记的程序允许安装在其中，智能卡或协处理器与外置服务器能建立连接来远程提供外置密钥到安装在隔离组件的代码中. 智能卡和加密协处理器的主要区别是安全相关是否在物理上与计算平台连接. 而当组件被使用作为终端用户身份认证时，移除智能卡是被允许的. 第二种主要包括两大类：管理加密操作和密钥存储的硬件安全模块和专门为安全子系统设计的通用处理器. 这里以通用处理器为例，它是内置在主处理器中的通用处理引擎，也是专为安全子系统提供专用的通用处理器的系统安全设计，主要是使用定制的硬件逻辑来阻止未授权软件对系统敏感资源的访问. 从以上陈述可以看出，通用处理器设计与 ARM TrustZone 技术的硬件安全扩展有许多类似之处，但也不尽相同，后续章节会详细阐述它与 TrustZone 设计的不同之处，也会对各种硬件隔离模块的实现方案的优势与不足进行对比分析.

2.2 软件隔离技术

软件隔离技术是在软件层构建一个可信的隔离运行环境，从而限制恶意代码的扩散或将可信软件、可信代码或敏感数据保护在该隔离环境中. 其中典型的软件隔离技术包括虚拟化技术和基于瘦特权软件层的隔离技术等.

虚拟化技术的功能是抽象一个虚拟的软件或硬件接口来保证在其上的软件模块能运行在一个虚拟出来的环境上. 虚拟化实质上是再现了整个物理服务器作为一个虚拟机来运行一个应用，并由虚拟化监控程序来抽象服务器资源和分配资源给虚拟机(VM)，然而监控程序执行抽象的开销将造成一定程度系统性能的损耗. 在系统各个层次都能实现虚拟化技术，其主要包括硬件虚拟化、OS 虚拟化及应用程序虚拟化等. 这里以典型的硬件虚拟化为例来进行介绍，其系统虚拟化架构如图 1 所示，通过使用虚拟化技术能抽象出多个虚拟硬件抽象层，从而隔离多个客户端操作系统. 比较典型的虚拟机是 KVM(Kernel-based Virtual Machine)和 Xen. 前者基于内核的虚拟机，它是 Linux 内核的一个非常小的模块；后者主要运行在裸机上.

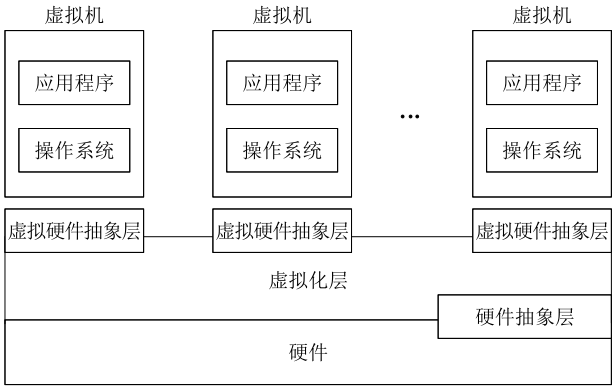

图 1 系统虚拟化架构

具体是硬件虚拟化抽象了整个系统的硬件层，处理器指令集为它与客户操作系统的接口. 虚拟机作为一个更高权限的软件隔离层运行在硬件与操作系统之间，它抽象化了整个硬件层，实现了虚拟资源到物理资源的映射机制，因此能协助客户操作系统映射到具体的硬件设备. 当客户操作系统需要使用敏感指令来访问系统资源时，虚拟机也能拦截这个操作并通过模拟相应的指令来进行处理. 这种使敏感指令沦陷，然后由虚拟机来模拟处理这些指令操作并返回结果给客户操作系统的机制有效地阻止了非法操作的执行. 同时，虚拟机能够保存和切换多个客户操作系统，这样有效地保证了各个虚拟系统之间安全隔离，互相独立，且互不影响[1].

另外，更多研究聚焦在减少内核和虚拟机的大小，大内核或大虚拟机意味着大的可信计算基(Trusted Computing Base，TCB)，这引起一系列安全问题. 为此，诞生出基于瘦特权软件层的隔离，如微内核或微虚拟机，它们因容易验证且组件之间不会受到影响等特点成为解决安全问题一种新途径，也逐渐成为研究的焦点. 以微内核为例，一个微内核操作系统在特权模式下只运行一组最小化的关键系统服务，如线程管理、异常处理和进程间通讯. 同时，它提供了一个让复杂系统软件运行在用户模式下的架构，在用户模式下，这些系统软件只被允许访问由系统设计者赋予的资源. 在该架构中一个组件的损坏不会破坏硬盘驱动，因为被感染的组件根本没有硬盘驱动访问权.

2.3 系统级隔离技术

系统级隔离技术主要是是通过对硬件进行安全扩展，并配合相应的可信软件从而在系统中构建一个相对安全可靠的可信执行环境(Trusted Execution Environment，TEE)[7]. 本节通过阐述 TEE 来叙述系统级隔离技术，后续章节会对各平台实现系统级隔离的机制作进一步的分析.

比如嵌入式领域，随着移动电子商务的飞速发展，我们正变得越来越依赖于移动计算设备并期望系统工作正常且足够安全. 然而嵌入式系统变得越来越复杂，系统漏洞已经无法避免. 如今移动应用程序为用户提供了前所未有的方便，无论何时何地，人们能方便地登录到邮箱接受邮件，核查信用卡余额，在线购物或者预定旅行等. 这一极大的方便也给用户的敏感信息，如信用卡信息，银行登录信息及用户个人信息等带来了前所未有的威胁. 同时，一些高质量内容需要保护，如高清视频和高清电影等，这些内容越接近发布时价值越高，但是这些内容又容易被盗版使用. 使用 TEE 可以在各种移动设备上保护这些高价值的内容，利用 TEE 来加密保护和传送这些内容，只有拥有特定密钥的 TEE 环境才能解密并使用这些内容. 另外，结合安全组件(Secure Element，SE)与 NFC(Near Field Communication)等，TEE 可以为消费者在其设备上提供一个可信环境来执行任何金融交易. 与此同时，政府和企业可以使用 TEE，在移动设备上进行机密信息的处理. 可以防止移动 OS 的软件攻击，以及控制对机密信息的访问权限. 这样，政府和企业可以雇员使用他们自己的移动设备(满足 BYOD)，不过需要保证机密信息和操作的处理在安全可信的 TEE 环境中进行.

TEE 是主处理器上的一个安全区域，它提供一个隔离的执行环境，可以保证程序的隔离执行、可信应用的完整性、可信数据的机密性及安全存储等. 并能保证加载到该环境内部的代码和数据的安全性、机密性和完整性. TEE 是与 ROS(Rich OS)并行运行的独立执行环境，为功能丰富的 ROS 提供安全服务，其内包含了一组应用程序接口(Application Programming Interface，API)来满足 REE(Rich Execution Environment)与 TEE 之间的通讯. TEE 内部由可信操作系统(Trusted OS，TOS)和其上运行的可信应用程序组成. TOS 有独立的初始化代码、安全服务、进程调度模块及内存管理模块等. REE 通讯代理为客户端应用和可信应用的消息传递提供了支持. 运行在 TEE 的可信应用可以访问设备主处理器和内存的全部功能，硬件隔离技术保护其不会遭受来自 ROS 的恶意攻击. 而 TEE 内部的软件和密码隔离技术可以保证其内可信应用程序之间的隔离. GlobalPlatform 和 TCG 近年来都在开展 TEE 方面的工作，前者以制定 TEE 的标准规范为主，后者试着将 TEE 规范与其可信平台模块规范进行结合以加强设备的安全和可信，并发布了最新的 TPM 2.0 规范[8].

2010 年 7 月份，GlobalPlatform 首次宣布了它们自己的 TEE 标准，如图 2 所示，提出 TEE Client API(与 TEE 交互的接口)，后来扩展到包含 TEE Internal API 以及一整套 TEE 系统体系规范. TEE Functional API 是对客户 API 的封装以便于开发者能在 ROS 中以标准的接口模式来开发调用安全服务的应用[3]. GlobalPlatform 包含的规范总结如下：TEE Systems Architecture 规范[9]描述了 TEE 下的软硬件体系结构；TEE Client API Specification 规范[10]定义了 ROS 的应用与 TEE 的可信应用如何通信；TEE Internal API Specification 规范[11]定义了如何开发能在 TEE 内部运行的可信应用，提供给其可信应用的编程接口；为了与用户进行友好的交互，TEE 还可以提供可信 GUI 用户接口，Trusted User Interface API Specification 规范[12]则是定义了可信 GUI(TGUI，Trusted GUI)，TGUI 可以在 TEE 中为用户的隐私数据提供安全显示，它完全独立于普通执行环境下运行的 ROS，因此不会遭受恶意 ROS 系统的攻击. 且 TEE 允许客户端应用和可信应用以共享内存的形式有效地进行数据通讯，这种机制很好地保证了客户端应用能调用可信执行环境内设计的各类可信应用. 一般 TEE 有访问 REE 中的资源的权限，反之则不然. 正因为 GlobalPlatform 的 TEE 规范构成 TEE 环境的基础，一般商业或者开源产品都会参考该规范，并按照其定义的各种功能接口进行规范实现，从而确保了不同厂商开发的可信执行环境的一致性.

商业界和学术界都陆续提出了一些 TEE，Trustonic 公司开发的 T-base①，捷德公司研发的 MobiCore②. 国外，格兰茨技术大学的 Andreas Fitzek 也开发了一款 ANDIX OS[13]，它是一款支持多任务、非抢占式的操作系统. 北卡罗莱纳州立大学的

① Trustonic. T-base. http://www.it-security-munich.net/wp-content/uploads/2014/12/07_Katzenberger.pdf

② Giesecke&Devrient. MobiCore. http://www.gi-de.com/en/trends_and_insights/mobicore/trusted-mobile-services.jsp

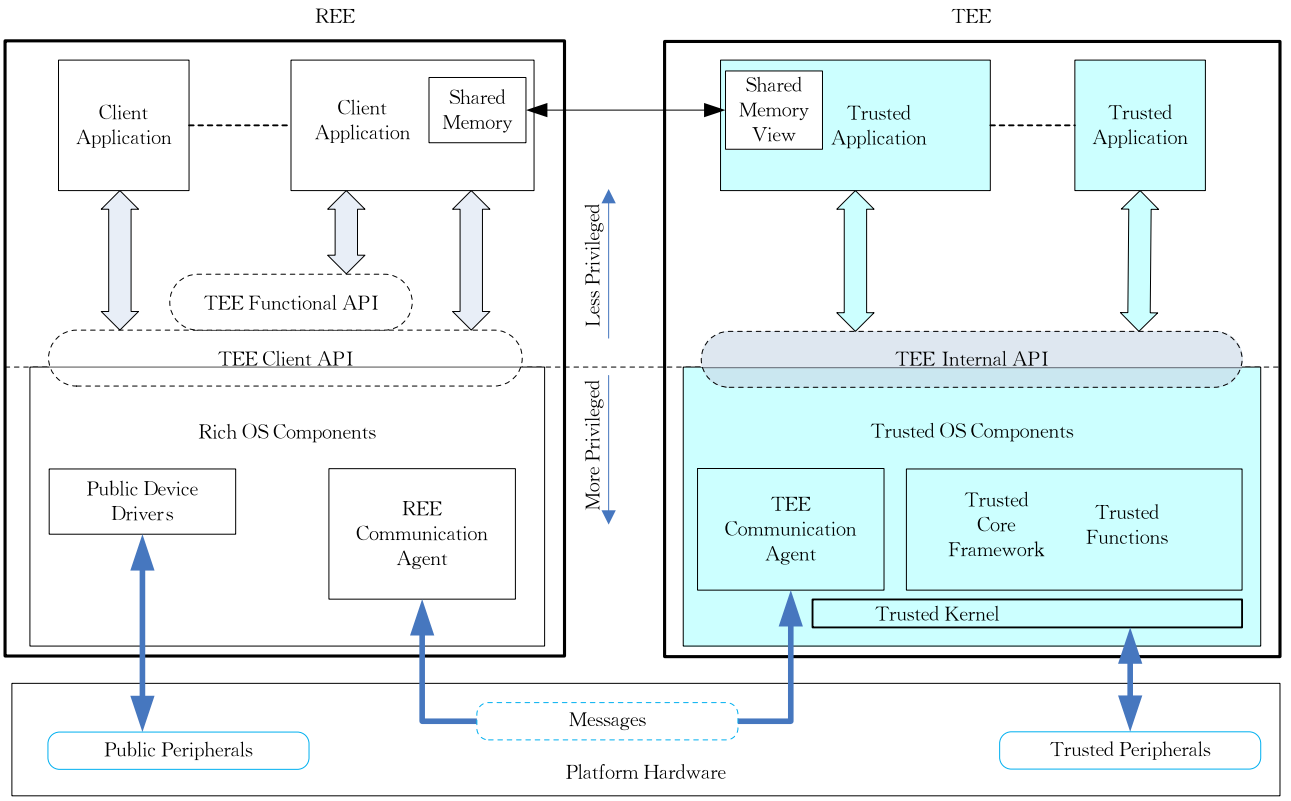

图 2 TCG TEE 系统架构[9]

Jitesh 利用 ARM TrustZone 技术开发一款安全 OS[14]. 国内,上海交通大学开发了 T6[15],T6 是基于 ARM TrustZone 安全扩展处理器开发的一个微可信操作系统,其普通执行环境支持运行 Linux 和 Android 等系统. 另外,Linux 社区开发了一款开源的 TEE 称为 OPEN-TEE[16];而 Sierraware 公司开发的 SierraTEE[17] 则提供了开源版本和商业版本两种,开源版本则缺少了一些关键组件.

3 安全隔离技术实现机制分析

本章主要分析以上各种安全隔离技术的实现机制,深入剖析这些技术所做的安全扩展和运行机理. 从上述章节陈述的实现方式可分为以下 3 个机制类别进行论述:硬件隔离机制、软件隔离机制及系统级隔离机制.

3.1 硬件隔离机制

为了在系统中处理一些关键数据或提供一些加解密模块,可以在 SoC 设计时在片内或片外专门设计一些安全模块 IP(Intellectual Property)来为系统提供安全服务.

外置安全硬件模块以智能卡为例,它是一个嵌入式集成电路卡,包含一个 CPU、内存和至少一个用来与主设备通讯的外设接口,如图 3 所示,可信域和不可信域之间有一个完好的物理隔离. 智能卡自身体积非常小,这些都使得智能卡具有较高级别的防篡改能力. 定义在可信域的可信服务非常安全,但是它的作用范围非常有限[18].

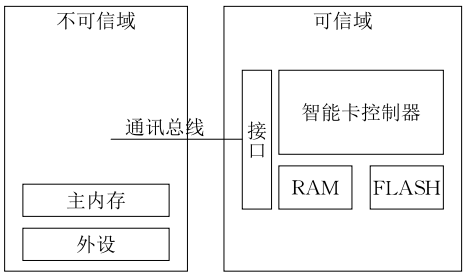

图 3 智能卡的系统架构

而安全组件是智能卡的变种,它是安装在移动设备中的一个嵌入式集成电路,通常结合近场通讯(Near Field Communication,NFC)使用. 另外,安全组件也能添加到任何设备的 Micro SD 卡或通用集成电路卡(UICC)中. 它能提供诸如数据保护、以小应用(Applets)的形式执行程序代码及硬件支持的加密操作,如 RSA、AES 及 SHA 等. 因为它能提供防篡改存储,因此它能很好地保证被存储的数据安全,防止它们被未授权软件访问或篡改.

硬件隔离机制提供了一个非常安全的隔离运行

环境，因为它利用硬件本身提供了完整性监控保护.尽管如此，现存基于硬件的安全隔离解决方案也存在一些限制.比如，一些研究者就提出了通过修改硬件的方式来实现事件驱动监控，但修改硬件是一个长期的目标，并不适合于现存的系统.

3.2　软件隔离机制

为了在系统中提供一个能够保护敏感信息，提供安全检测防御机制及隔离恶意进程，可以基于软件的方式实现一个相对安全的隔离运行环境，以免受到来自复杂系统中的各种恶意攻击.为此，各种软件隔离机制应运而生.本节着重分析虚拟化技术、主流容器技术 Docker、沙箱、蜜罐及微内核等软件隔离的实现机制.

虚拟化技术[19-21]是在一个物理处理器虚拟出多个虚拟机(Virtual Machine，VM)，虚拟机的资源由虚拟机监控器(Virtual Machine Monitor，VMM)统一进行管理.其中 VMM 是高权限的软件层，为系统提供了一个良好的隔离环境，具备很多安全特性，它能保证各虚拟机之间能够相互隔离，互不影响，其隔离机制如图 4 所示.

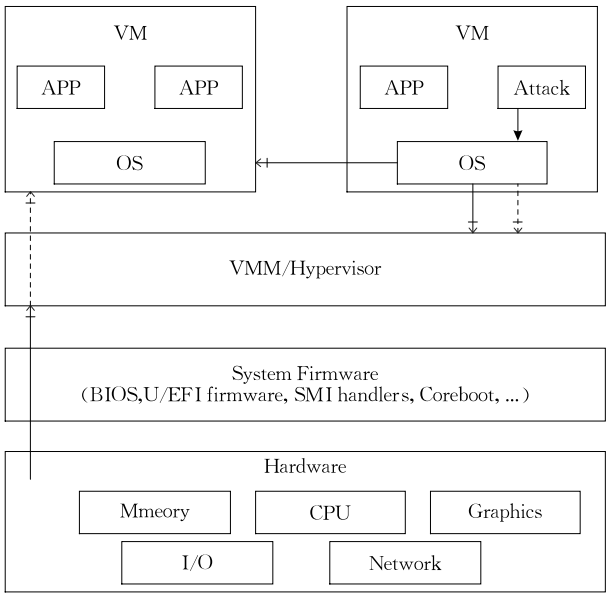

图 4　基于虚拟机的隔离机制

虚拟机利用 IOMMUs 来限制硬件对虚拟机的直接访问，也限制硬件对 VMM 的直接访问.某个虚拟机中的恶意程序会损害其自身的操作系统，然而该受损系统不能直接访问 VMM 和其它 VMs，因而虚拟机之间达到了较好的隔离效果.它能够控制所有的硬件资源并能捕获 VMs 中的中断和异常，因此利用该技术能够构建一个安全可靠的隔离执行环境.另外，带 MMU(Memory Management Unit)的处理器都能便捷地实现虚拟化，无需增加额外的硬件.这样，通用操作系统运行在普通执行环境下，而安全操作系统则运行在一个可信的执行环境中.

上述章节提及的操作系统层上的虚拟化、应用程序层上的虚拟化、库函数层上的虚拟化及编程语言层上的虚拟化都不能保证应用程序与操作系统之间完全透明和隔离[1].而基于硬件抽象层上的虚拟化则因为能够虚拟出硬件抽象层，所以能够保证应用程序与操作系统透明的运行环境.

对于容器技术，以主流的 Docker 为例来分析其软件隔离机制，由于它与虚拟化技术极其类似，故在此将它与虚拟化技术作对比分析进行陈述，如图 5 所示.Docker[22]为应用程序提供隔离的运行空间，它是从操作系统内部实现了进程的隔离，是一种操作系统层的虚拟化.每个容器独享一个完整用户环境空间，且一个容器的变动不会影响其它容器的正常运行.

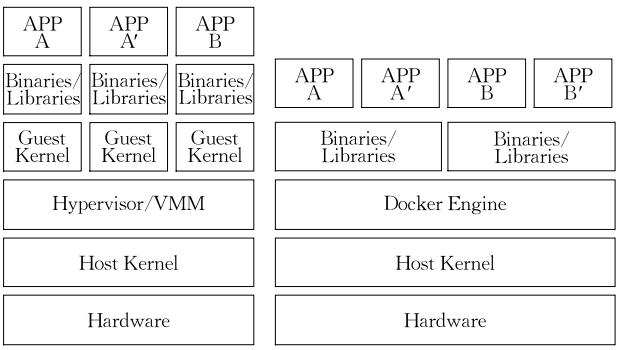

图 5　VMM 与 Docker 对比

Docker 可以自动将任何应用打包成轻量、可移植及自包含的容器引擎.开发者构建的应用可以一次构建在全平台运行[23].Docker 容器之间共享一个系统内核，因此启动非常快，然而同一个库有可能被多个容器同时使用，因此内存的使用效率会显著提高.另外，Docker 采用了一些现有的系统机制，如采用 Linux namespaces 来进行空间隔离，采用 cgroups 来确定每个容器可以使用多少资源，采用文件系统的挂载点来决定容器允许访问的文件等.

沙箱(SandBox)[24]也是一种软件隔离的运行机制，它按照严格的安全策略来限制不可信进程或不可信代码运行的访问权限，因此它能用于执行未被测试或不可信的应用.它的软件隔离机制如图 6 所示，沙箱内的应用需要访问系统资源时，它首先会发出读系统资源的请求，然后系统会核查该资源是否在它所操作的权限范围内，如果核查通过则完成读请求，否则系统会拒绝其操作.SandBox 能为不可信

应用提供虚拟化的内存、文件系统和网络资源等，也正是由于其内的资源被虚拟化，它能将不可信应用的恶意行为限制在其有限的机制内，这样能防止不可信应用可能损害其它应用甚至是威胁系统的安全.

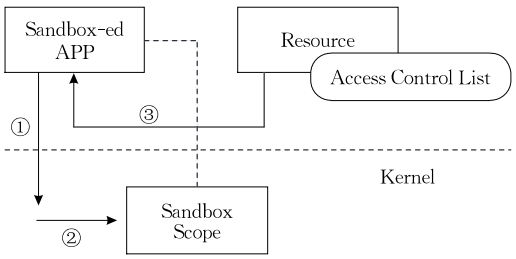

图 6 Sandbox 隔离机制

蜜罐技术(Honeypot)是另一角度的软件隔离机制，它是一种入侵检测机制，能收集系统所遭受的攻击，因此可以提高系统检测机制的检测能力[25]. 它通过诱骗攻击者使其误认为成功侵入目标主机或网络，从而对攻击行为进行分析并完善安全策略进而构建更加安全全面的入侵检测机制. 这种方式能够消耗攻击者的计算资源，也能够直接收集具有较高价值的攻击者信息，还能够捕获一些从未见过的攻击行为，却需要很少的系统资源. 然而，Honeypot 只能捕获与它直接交互的攻击行为，对其它的系统攻击行为则无能为力. 同时，它自身往往携带一定的预期特征或者行为，这样攻击者可能可以识别出其身份并对它实施攻击，甚至利用它危害系统的安全. 但因不同 Honeypot 构建和部署的需求和方式不一样，因此其风险具有个体差异.

还有一种基于瘦特权软件层而实现的隔离安全研究，如微虚拟机或经过验证的微内核，这些途径在隔离敏感任务是非常有效的. 以微内核为例，微内核操作系统采用多级安全架构，可以将任务划分成多种安全级别，这样就可以根据用户的安全需求进行设置. 可以将任务按照非密、秘密、机密和绝密的级别进行划分，也可以根据任务的安全等级进行更加细粒度的划分，进而采用相应的访问控制策略来实现隔离. 这样就可以隔离安全工具来验证每个操作并监控内核的完整性. 遗憾的是，通过这些途径来隔离内核组件从而阻止攻击者对系统关键数据结构的访问的方式完全远离了商业操作系统的设计. 文献[26]提出一个简单的瘦虚拟化层，从而大大减少攻击面并增加了系统的整体安全. 文献[27]在硬件虚拟化的基础上提出了一个轻量级的虚拟机 Hytux，它拥有比 Linux 内核更高的权限，从而能保证实现在 Hytux 中的防护系统内核的安全机制免于遭到一些恶意攻击. 然而，微虚拟机独占了硬件虚拟化扩展，这严重影响了这些途径的便携性和灵活性. 另外，微虚拟机和 hypervisors 都依赖于虚拟化扩展，因此，它们不是在所有的平台都是有效的. 文献[28]基于先进的微内核提出一个通用操作系统框架，它允许虚拟机与安全应用并行运行，并确保了它们之间的安全隔离. 文献[29]提交了一个全面且严格的方法来验证 seL4 微内核的某些安全性质，如由于软件漏洞产生的一些安全隐患等.

3.3 系统级隔离机制

基于目前纯软件和纯硬件的安全隔离机制都存在各自的局限性，为此业界和学术界都纷纷提出了系统级的安全隔离机制来保障系统的安全. 本节主要分析各种系统级隔离机制的实现原理和执行机制. 其主要包括嵌入式领域的 ARM TrustZone[30-31] 和 Texas Instruments M-Sheild[32]，PC 领域的 Intel TXT[33]、AMD SVM[34] 及 Intel SGX[35]，并基于这些平台发展起来了各种可信执行环境，本节也会着重进行描述.

ARM 公司推出一套系统级解决方案 TrustZone 技术，它提供了一个具有高度安全性的系统架构，而对于内核的功耗、性能和面积的影响微乎其微，其系统架构如图 7 所示.

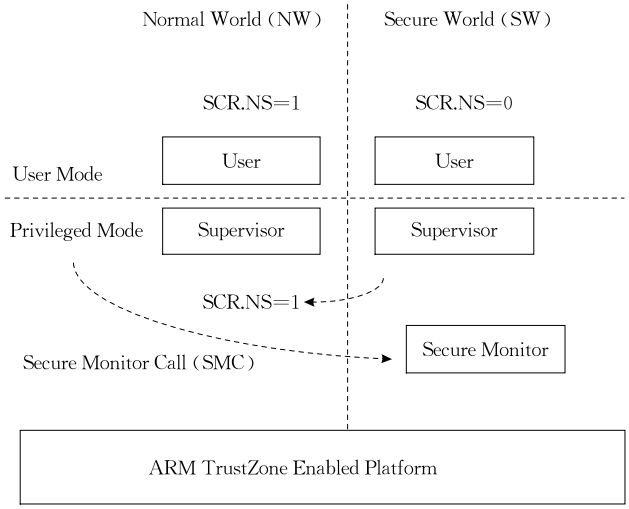

图 7 TrustZone 的系统架构

TrustZone 包含：硬件—提供代码隔离的保密环境，保密软件—提供基本的安全服务和其它安全环节上各部件间的接口，如智能卡、操作系统和普通的应用程序. TrustZone 在同一物理处理器上隔离了两个并行的运行系统：处理一般事务的普通执行环境和处理敏感事务的可信执行环境，两个执行环境都有一套独立的内存管理模块. 新引入的 Monitor

机制负责保存当前系统上下文状态来确保两个环境之间的安全切换.在没有得到 Monitor 的允许,系统不能随意地从一个环境切换到另一个环境,除非例外发生或调用 SMC(Secure Monitor Call)指令.

TrustZone 能根据应用需求将安全理念扩展到其它内存和外设上,NS(Non-Secure)位是 TrustZone 对系统的关键扩展,用来指示系统当前处于安全状态还是非安全状态.具体是在 Cache 的每个 tag 域中都扩展了一个 NS 位来将数据标记为安全与非安全,页表项和 TLB 相应地都扩展了一个 NS 位,这样系统通过动态验证所有的 NS 位来防止非授权操作去访问可信执行环境的资源.

TI M-Shield 是德州仪器(Texas Instruments)针对嵌入式领域推出的另一套系统级安全隔离方案,系统中敏感应用中的执行和关键数据的存储均在硬件支持的安全隔离环境中完成,这样可以保证高价值内容的安全传送和安全存储,其系统架构如图 8 所示.具体是它在硬件支持的安全隔离环境中设计了一个安全状态机(Secure State Machine,SSM)及安全的 ROM 和 RAM 等,其中 SSM 负责保证敏感应用在安全隔离环境中执行的安全策略.它也定义了安全 DMA,因此保证了 DMA 传输的安全性.同时,它确保了芯片互连的安全,也确保了安全软件能够安全访问外设和内存,从而保证了敏感数据传输的安全性.

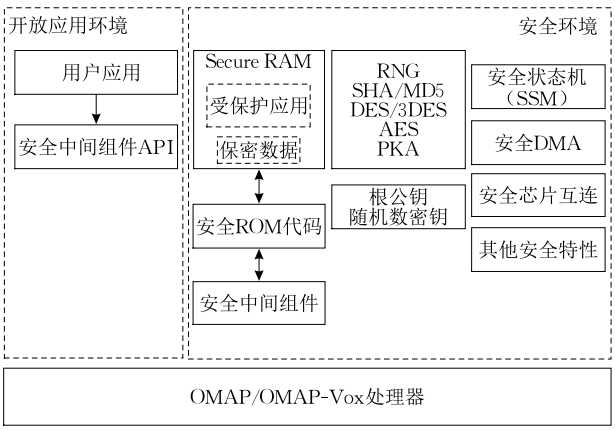

图 8　M-Shield 的系统架构

M-Shield 提供一个公钥基础架构来保证各种软件在执行前的真实性和完整性.它提供了基于硬件的 AES 和公钥加速器(PKA)及 DES/3DES,SHA 和 MD5 硬件加速器.其中硬件加速通过身份认证、快速解密和完整性核查提升了用户体验.在手机部署了有价值的安全服务时,该技术减少了对其未授权使用和诈骗.另外,为了便于开发安全应用该技术也加入了安全中间组件:一个安全基础框架和基于 TrustZone 标准的 API[32].

Intel SGX(Software Guard Extensions)[36-38]是 Intel 最近推出的一款新处理器技术,它能提供一个硬件支持的安全隔离区来保障系统关键代码和隐私数据的安全,其架构如图 9 所示.SGX 将可信软件的敏感操作放在一个 Enclave 中执行,防止其遭到恶意攻击,而不是隔离系统中所有的恶意软件.Enclave 提供一个隔离的可信区域,也可理解为一个 TEE,无论恶意软件有何种特权级别都无法访问它.操作系统、VMM 和 BIOS 也不能更改其中的代码和数据.这样,Enclave 能很好地保障用户的关键代码和数据的机密性和完整性.同时,SGX 也阻止 Enclaves 之间的互相访问,SGX 的安全边界(可信隔离环境与普通环境的边界)只包括 CPU 和它自身.

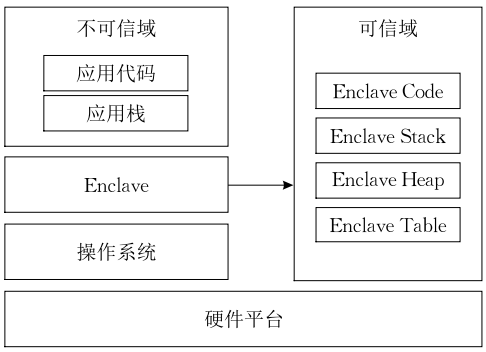

图 9　Intel SGX 的架构图

因为一个 Enclave 是用户地址空间的一部分,因此,为了保护它,防止主存受到软件和硬件攻击是十分必要的.为此,SGX 使用 EPC(Enclave Page Cache),EPC 是一组经过加密保护的内存,其页大小为 4 KB.为了加密主存,管理它的完整性及保护它与处理器之间的通讯安全,为此,使用了一个硬件单元—内存加密引擎(Memory Encryption Engine,MEE).处理器跟踪 EPC 中的内容,EPC 每页的安全属性被储存在一个独立实现的微结构 EPCM(Enclave Page Cache Map)中[18].

SGX 使用 ECREATE 指令创建 Enclaves,它分配一个空闲的 EPC 给 SECS(SGX Enclave Control Structure),并将它初始化在受保护的内存上.此后,页都必须通过 EADD 指令添加,即使是由 OS 分配的页,它们仍需被映射到物理内存上的 EPC 中.在硬件内按 EPC 页类型为单位跟踪每个 EPC 页,主要包括它被映射给哪个 Enclave,它在 Enclave 内的虚拟地址以及它的权限[18].

另外,研究者也提出了以可信硬件为基础的框架来为系统提供从启动到系统核心程序再到应用程序完整的信任链.由于它提供认证启动,启动序列的初始部分会被度量进可信平台模块(Trusted Platform Module,TPM),因此它能够有效抵御来自恶意代码的攻击.TCG(Trusted Computing Group)定义 TPM 如图 10 所示,它内含密钥发生器、存储设备及对称密钥保护器,它通常被用来存储系统状态,密钥,密码及证书等.TPM 是有具体要求的隔离组件并物理上连接到计算平台,即一个安全加密协处理器.由主机平台和外置的隔离组件增加一系列额外的可信根和可信计算功能,主机平台的软件和硬件状态的远程认证可通过结合隔离计算组件和可信根实现.然而,这种途径也有许多不足之处.在设备启动后,度量不能重复,从而不能阻止运行时的攻击;在开放的硬件平台,设备拥有者可以插入在硬件级别上实现内存访问的辅助硬件设备,而这些设备的固件和 DMA 使用不能在启动时被度量,无法保证它们的安全;它的安全性依赖于控制策略,在实际中也会因为过于严格的控制策略往往使得用户禁用 TPM 功能.在 2005 年,Intel 和 AMD 都基于 TCG TPM 设计一个概念"Late Launch",分别称为 AMD SVM 和 Intel TXT,并基于此发展起来许多可信系统.

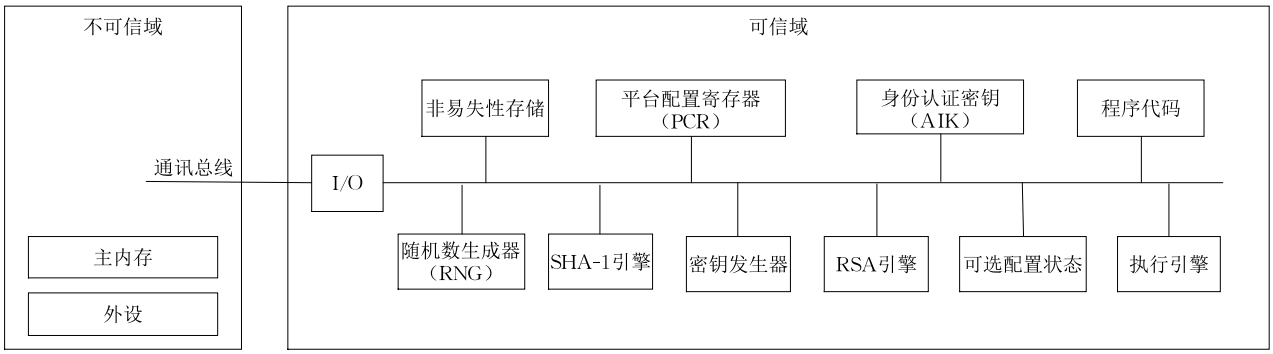

图 10 TPM 架构

Intel TXT(Trusted Execution Technology)是一系列 Xeon 处理器的硬件安全扩展,它提供动态可信根度量(Dynamic Root of Trust for Measurement,DRTM)以便能在运行时度量系统的完整性.TXT 依赖于 TPM 的平台配置寄存器(Platform Configuration Register,PCR)来存储完整性参考度量值,TPM 作为 TXT 的可信根,该可信根是计算平台成功评估所必须的基础结构.该技术通过创建一个度量启动环境来对启动环境中的关键组件做一个精确的度量,其为每个受到许可的启动组件都设置了一个唯一的经过加密的标识,并且提供基于硬件的增强机制制止未授权代码的启动.基于硬件的解决方案为建立可以防止威胁系统完整性、可信性、稳定性和可用性的软件攻击的可信平台提供了基础架构.一旦基本的可信根和安全度量被构建起来,它可以进一步扩展这些能力和技术,比如密封和保护内存中的密钥及提供系统配置的本地和远程认证等[33].

AMD 安全虚拟机(Security and Virtual Machine,SVM),其代码名"Pacifica",它极大地便利了虚拟化的开发和部署.SVM 的虚拟机架构应提供硬件资源,允许单个机器有效地运行多个操作系统,并要保证系统安全和资源管理的隔离.SVM 处理器提供一系列硬件扩展来实现经济高效的虚拟化系统,主要分为两大块:虚拟化支持和安全支持.虚拟化支持主要体现在:VMM 和 guest 之间的环境快速切换机制;拦截 guest 中被选择的指令或事件的能力;DMA 访问内存保护;协助中断处理和虚拟化中断支持;一个 guest/host 中被标记的 TLB 来减少虚拟化负载.并且引入一种新的处理器模式——Guest 模式,系统通过调用 VMRUN 指令进入到该模式当中.该技术也提供了外置访问保护,guest 能直接访问 I/O 设备,硬件会阻止一个 guest 的设备去访问另外一个 guest(或 hypervisor)拥有的内存空间.安全支持主要体现在:在 SVM 惯用模式,VMM 被映射在一个与 guest 不同的地址空间,而为了减少环境切换成本,TLB(Translation Lookaside Buffer)被添加一个地址空间标识符(ASID)来区别 host 空间与 guest 空间;VMM 能请求物理中断使运行的 guest 退出而允许 VMM 来处理这个中断,它也能向 guest 注入虚拟中断;为了进一步的安全初始化,SVM 提供了一些额外的系统支持,SKINIT 指令和协助系统支持(TPM)负责可信软件(如 VMM)的启动验证;复位时自动内存清除会擦除系统内存的

内容，这样可以阻止针对存储在内存中的密钥的简单重置攻击[34].

3.4 总　结

本章着重分析了以上 3 种安全隔离技术的实现机制，主要针对其系统架构中安全扩展作了详细说明，实际应用应根据安全需求进行选择. 基于它们已做了大量研究工作，后续章节深入分析调研了它们的研究现状，并总结了它们在解决系统安全问题方面的优势与不足之处.

4 基于安全隔离技术研究现状分析

目前基于安全隔离技术方面的研究工作已经成为一种主流趋势，基于这类技术构建的安全隔离环境来保障系统敏感操作和用户关键数据的安全是一种切实可行的设计方案. 为此，学术界和业界都提出了大量的基于安全隔离运行环境实现的一系列行之有效的系统安全解决方案. 本章对以上各种隔离机制在解决系统安全问题方面的研究现状进行了充分调研分析，并根据其解决的安全问题分析总结归纳出以下几方面的研究工作. 其中，纯硬件的方式主要是应用型，在前述章节已作具体介绍，在这只是对其应用作简单介绍，本章侧重对纯软件和系统级隔离的研究现状进行分析，并分析它们在保障系统安全方面的优势与不足.

4.1 硬件隔离技术的研究现状

基于硬件隔离机制目前比较主流的是 IBM 4758 加密协处理器和智能卡，前者主要应用于验证授权、银行服务器及工厂网络. 后者主要应用于 SIM 卡及机顶盒中智能卡等，提供安全应用隔离环境. 如智能卡能使用于强化信用卡和使能支付，SMIS 组织就在智能卡内实现了安全的数据库，它能保护和处理敏感数据. 多应用智能卡技术，如 MultOS 和 JavaCard，这种高端智能卡提供一个安全的环境，其威胁模型非常类似于加密协处理器. 如今，智能卡在金融领域也广泛被使用作为认证模块[18].

4.2 软件隔离技术的研究现状

当前，软件隔离技术的研究大部分围绕虚拟机展开. 虚拟机技术的安全目标通常可以概括为以下 5 个方面：(1) 抵御代码域注入攻击；(2) 抵御数据域注入攻击；(3) 抵御不可信内核扩展攻击；(4) 抵御权限提升攻击；(5) 抵御系统环境执行变量攻击. 本节以前述 5 个安全目标为线索，阐述软件隔离技术的研究现状.

4.2.1 抵御代码域注入攻击的研究

代码域注入攻击方式是攻击者使用恶意代码永久覆盖这段可写的内核代码或拦截这段内核代码并增加一些新的恶意指令来执行这段代码. 为了防止此类攻击，可以通过虚拟机提供的软件隔离环境来增强页表属性从而确保内核代码仅仅能执行但不可写.

文献[39]提出 NICKLE 实现 VMM 级的内核指令获取限制，最直接的途径是在内核内存页引入 W⊕X 机制来保护现存的内核代码并制止注入内核代码的执行. 因为商用操作系统存在混合内存页，它会使得 W⊕X 机制失效. 为了解决这个问题，该文基于 VMM 提出了 NICKLE，VMM 创建了两个内存区：the standard memory 和 the shadow memory，ROS 内核不能访问 the shadow memory. 当 VM 启动，VMM 对内核代码进行授权并动态地将授权内核指令从 the standard memory 复制到 the shadow memory. 运行时，任何执行在内核空间的指令必须从 the shadow memory 获取. 这样虽然内核 rootkit 能将恶意代码注入内核空间，但却不能被复制到 the shadow memory，由此便永远不可能被执行. 文献[40]提出 SecVisor，它利用新的硬件扩展来保证内核终身完整性并提供类似于 NICKLE 的安全保障. 它是基于 VMM 机制而提出增强内存页 W⊕X 的属性，能阻止未授权代码以内核级权限运行，也能阻止攻击者篡改内核代码或以内核权限执行注入代码. 但是 SecVisor 直接与混合内核页冲突，而该混合内核页又普遍存在于当前主流的操作系统当中，如 Linux 和 Windows. 然而，SecVisor 和 NICKLE 有两个主要的不同：SecVisor 的部署需要修改 OS 内核源码，也需要 MMU 和 IOMMU 虚拟化硬件支持，而 NICKLE 不依赖于客户端 OS 数据结构的保护，如全局描述符表.

4.2.2 抵御数据域注入攻击的研究

针对数据域的攻击主要分为两类：一类是使用恶意的双映射机制来修改内核数据，它具体是通过捕获内核漏洞来产生内核内存的用户空间映射，其典型攻击类型是 motochopper；另一类是敌手捕获内核漏洞来跟踪内核，从而直接修改内存中的一个或多个关键数据域，这类攻击的典型例子是 towelroot. 目前为了阻止攻击者对系统内存上关键数据的篡改来防止这类攻击，主要有两种比较可行的方案：地址空间布局随机化(Address Space Layout Randomization，

ASLR)和数据不可执行(Data Execution Prevention, DEP).然而,前者主要是通过随机排列程序的关键数据区域的位置,包括可执行的部分、堆、栈及共享库.它在保护内核空间抵御恶意攻击方面不是非常有效的.这是因为使用 ASLR 的方法必须在 64 位架构上实施才能有效阻止这类攻击,且仅仅适合用户空间.而后者是强制将数据段设置为可读写但不可执行,具体可以通过修改页表属性实现.

文献[4]在 x86-64 处理器上提出了嵌套的内核架构并命名为 PerspicuOS,PerspicuOS 将内核分离成两个特权域:the nested kernel 和 the outer kernel,它确保 the outer kernel 从来不使能写保护,限制 the outer kernel 域执行有效的和受保护的代码.由此保护内核和受保护的内存域与外界隔离,防止被恶意篡改.文献[41]提出的系统安全框架 SecPod 也做了类似的工作,它为安全工具提供了一个可信的执行环境.其主要包括两个关键的技术:在这个安全隔离环境内审计内核的页操作;对通过误用特权指令试图毁坏 SecPod 的操作实施拦截.文献[42]通过内存访问控制提出一种内核数据完整性保护机制 Sentry,它设计了访问控制策略和增强系统,从而有效地阻止了不可信内核组件更改内核使用的关键安全数据.该文也对内核数据布局进行划分和重组来维持内核性能.

4.2.3 抵御不可信内核扩展攻击的研究

当今商用操作系统都支持内核级扩展来扩展内核的功能.然而,攻击者能利用这些扩展接口来破坏内核的完整性.例如攻击者能安装恶意内核扩展,典型案例是通过安装内核 rootkit 来隐藏他们在系统中的恶意行为.另外,一些第三方设备也暴露许多的安全漏洞,这些漏洞也能被恶意攻击者利用来向内核空间注入恶意代码.因此,阻止不可信扩展来保证内核完整性是一个非常具有挑战性的问题.为了防止这类攻击目前主要有两种解决方案:一种是限制修改内核空间内存的内核功能扩展,但这对于内核的功能扩展有较大影响,往往偏离实际应用需求;另一种是对扩展模块进行审核后再载入内核中.

文献[43]提出一种基于虚拟机的系统完整性保护机制 HUKO,它来阻止攻击者利用不可信扩展来损害商用 OS 内核的完整性.在 HUKO 系统中,不可信内核扩展能安全地运行并提供预期的功能.因为不可信内核扩展会受到强制访问控制策略的约束,这项研究极大地限制了攻击者损坏内核完整性的能力.文献[44]提出一个轻量级虚拟机 Barrier 来增强个人计算机内核的完整性.具体是,它利用硬件虚拟化来隔离内核模块至不同的地址空间,所有这些地址空间上的交互必须受到严格预定义的访问控制策略的限制,并考虑了所有的内核模块,这极大地增加了攻击者损害内核完整性的难度.因此,它较之前那些仅仅隔离动态扩展的方案在保护内核完整性方面具有明显的优势.文献[45]基于硬件虚拟化而提出了一种 Harvard 结构 hvmHarvard,它有效地保护了商用 OS 可能遭受到的 rootkit 攻击,并在 x86 上得到了有效地实现.该 Harvard 结构有两块内存空间:一块是代码空间;另一块是数据空间.这种结构能强有力的防止现在操作系统中 rootkit 部署的代码注入攻击.该研究与之前提出的通过指令级重定向来虚拟化 Harvard 结构有着显著的不同,它采用页级敏感模式的机制达到了与之前同样的效果,却大大减小了系统的性能负载.

4.2.4 抵御权限提升攻击的研究

这类攻击通过捕获内核漏洞来执行非内核代码,从而将一个有效的用户空间地址写入内核指针,这样就能向内核空间注入来自用户空间的非法数据,甚至是执行用户空间代码.这类攻击的典型例子是 vroot,敌手通过在特权模式下修改内核函数指针跳转到恶意的用户空间代码中.文献[46]针对通过毁坏内存来实现权限提升的攻击设计了一种防御机制 KENALI,它利用数据流完整性来增强内核访问控制系统的安全参数,其主要包括两种新技术:一个是自动推理数据,这对于没有手动注释的访问控制系统是非常关键的;另一个是通过推理结果实施有效的数据流完整性增强.该研究在一个可接受的性能负载前提下阻止了大量部分权限提升攻击.

4.2.5 抵御系统环境执行变量攻击的研究

这类攻击并没有向内核注入代码,而是通过非法修改内核变量来改变内核行为.如存在于栈中的控制流数据,决定一些代码分支条件所使用的数据及页表的属性(Present, Read/Write, No eXection 标记等).攻击者通过修改栈中的数据来篡改内核的控制流,使其以错误的序列执行.如恶意行为通过修改栈帧能执行一个伪造参数的函数(或仅仅一些代码).这样能用原始代码在内存中的地址替换保存在栈中的程序计数器,这样做是为了改变执行流从而执行一段恶意代码.攻击者也可通过修改页表的属性来执行一个特殊页.另外,整数溢出,尤其是参考计数器溢出,都属于这类攻击类型[27].对此类攻击的防御保护在前面章节中提到的 Nested Kernel 和

SecPod 等研究工作中都有涉及，对他们针对执行环境变量的攻击提出的保护机制，在此不再赘述.

4.3 系统级隔离技术的研究现状

前文提到虚拟化自身不可避免的存在大量的安全漏洞，其本身也面对许多安全挑战. 总之，之前的研究也展示了使用 hypervisor 来进行内核保护有较大的安全隐患. 认识到内核和 hypervisor 受到的安全威胁后，主流硬件平台引入了一种新的系统级安全隔离环境，它被称作“安全世界”. 安全世界(the secure world)组成了一个代码非常小且更安全的软件层来管理安全服务，而且它与普通世界(the normal world)具有非常有限的接口且通过硬件保护阻止了来自普通世界或 hypervisor 的非法内存访问. 前面已经提到目前实现安全世界的主流技术主要有 Intel TXT 和 SGX，AMD SVM，TI M-Shield 及 ARM TrustZone. 可信系统 Flicker 和 TrustVisor 都是基于 Intel TXT 和 AMD SVM 发展起来的隔离系统，Open Virtualization、Open-TEE 和 T6 等都是基于 ARM TrustZone 发展起来的可信隔离系统. 本节主要分析总结系统级隔离技术在解决系统安全方面的研究现状，它除了能解决以上虚拟机在实现系统防御方面的功能之外，还能为系统安全提供一些新的解决思路. 通过分析总结将其分为：保障系统安全的研究、保护虚拟机安全的研究、保护应用程序安全的研究及实现安全启动的研究.

4.3.1 保护 hypervisor 安全的研究

前文提到 hypervisor 因自身的特点存在许多的安全隐患，因此基于软硬件结合实现的系统级隔离环境来保障 hypervisor 的安全性也是一种比较行之有效的方法，这方面的研究工作已经比较广泛.

文献[47]就提出一个用来度量系统中运行的 hypervisor 或其它最高权限软件层的完整性的新颖架构 HyperSentry. 它引入一个与 hypervisor 隔离的软件组件，该组件能够较隐匿的度量运行的 hypervisor的完整性，并能很好地保存度量上下文. 这种隐匿度量的方式保证了受损害的 hypervisor 没有机会隐藏其攻击痕迹，而保存上下文度量保证了能够很好地恢复一个成功的完整性度量所涉及的输入. 文献[48]提出一个硬件支持的恶意篡改检测框架 HyperCheck，它用来保护 VMM 的完整性. 具体是，HyperCheck 利用 x86 系统的 CPU SMM(System Management Mode)来安全地产生和传送被保护主机的全部状态到外置服务器. 使用 HyperCheck，我们能检测出一些影响 Xen 虚拟机和传统操作系统的完整性的 rootkit. 另外，HyperCheck 能强有力地阻止不使能或阻碍其自身操作的攻击. 早前文献[49]也提议一种使用 SMM 来监控虚拟机完整性的机制 HyperGuard，而 HyperCheck 较它系统性能已得到大幅度提高. 它们都依赖于 SMM 为完整性度量代码提供硬件保护，然而它们有严重的限制. 首先，它们触发完整性度量需要修改并告知 hypervisor，这使它们很容易遭受 scrubbing 攻击，受损的 hypervisor 可能在完整性度量以前清理攻击痕迹. 另外，它们没有解决运行度量代理在一个硬件虚拟化平台 SMM 模式下的一些技术问题，比如如果 SMM 中断一个客户 VM 而不是 hypervisor 则 hypervisor 文本将被隐藏在 CPU 中[47]. 文献[50]也基于 TrustZone 的安全隔离环境提出一个多层次安全架构概念来核查 hypervisor 的完整性，该概念架构涵盖了关键组件载入和运行时的验证，组件之间的强有力的隔离及虚拟机的异常自查. 然而，以上提到的系统安全架构无法监控内核中的事件从而可能导致系统瘫痪. 尽管它们对内核和虚拟机完整性做周期性检查，但由于只能在攻击发生后才能检测到，所以这起到非常小的作用. 另外，如果恶意攻击具有隐藏功能，这类机制将无法检测到. 因此，比较好的方式是能够在攻击发生前进行阻止.

4.3.2 保障系统安全的研究

系统存在诸多的安全问题，尽管基于 hypervisor 在解决系统的安全方面起到一定的作用，然而虚拟机自身的安全已无法得到保障，为此基于其系统安全解决方案存在诸多的安全隐患. 而目前基于系统级的隔离运行环境来保障系统的安全已经变成一种非常行之有效的措施. 前文已对系统防护的种类作详细分析，在此不再赘述，这里简单分析基于系统级隔离环境所做的一些系统防护研究工作.

文献[51] 基于 TPM 提出一个可信的隔离系统 Flicker，它具有非常小的 TCB，能用来执行安全敏感的代码并提供了执行代码的远程认证功能. 然而，Ficker 的 rootkit 检测机制因为需要运行系统调用完整性度量使它较易于遭受 scrubbing 攻击，且其带来较高的性能负载. 为了减少 Flicker 的负载，文献[52]也基于 TPM 提出一个类似的可信隔离系统 TrustVisor，所有的合法应用都被传送到 TrustVisor 来工作，它也使用一个占用空间非常小的 hypervisor 来实施一些加密操作. 文献[53]基于 TrustZone 提出了一种实时内核保护机制 TZ-RKP，它使用新颖的技术剥夺普通世界对某些特权的系统功能的控

制,从而可以有效地阻止修改或注入二进制文件的攻击,也可以阻止修改系统内存布局的攻击,如通过内存双映射.文献[54]也基于 TrustZone 提出增强内核代码完整性的机制 SPROBES,它使能安全世界动态地中断普通世界的事务,并指定一个安全世界内的可信服务来处理这个事务;它能限制普通世界内核执行非法内核代码页,即使内核是 rootkit 控制.具体是:用户空间代码永远不能以特权形式执行;操作系统雇佣的 W⊕X 机制必须保持使能;页表基地址必须总是对应到合法的页表;对页表实体的任何修改必须劫持并被验证;MMU 必须保持使能以确保所有现存的内存保护功能正常.

4.3.3 保护应用程序安全的研究

随着系统应用程序涉及越来越多的敏感信息,为此如何保障应用程序自身的安全及怎样安全处理和存储应用程序中涉及的敏感信息变得尤为重要.而基于系统级安全隔离运行环境为解决以上的安全问题提供了新的思路,因此基于其的研究工作也逐渐成为一个研究的热点.

文献[55]提出 OverShadow,它是在 VMM 级采取了一种内存遮蔽技术来保护应用内存页防止被 OS 恶意修改.具体实现是,基于运行操作系统内的虚拟机加密隔离其中的应用,这样即使整个 OS 受到损坏也能为应用数据提供保护.然而,因其 VMM 自身存在许多的漏洞,所以这种方式存在许多的弊端.为此,基于系统级隔离环境实现应用程序的保护已然是一种比较行之有效的方法.文献[56]就基于 TrustZone 提供的安全隔离环境提出一种保护动态口令(One-Time Password,OTP)安全的机制 TrustOTP,该机制能在 ROS 遭受损害甚至毁坏的情况下保护 OTP 的完整性,也能保证 OTP 产生的可靠性并提供了 OTP 的安全显示.文献[57]也基于 TrustZone 平台提出一种在线移动广告认证的安全机制 AdAtterster,其主要提供两种新颖的安全原语:无法被篡改的点击和经验证的显示,该原型设计在 Samsung Exynos 4412 开发板上得到验证.文献[58]基于 TrustZone 机制提出一种安全有效的直接匿名认证(Direct Anonymous Attestation,DAA)机制 DAA-TZ.这些研究都是利用系统级安全隔离环境来隔离 ROS 中的敏感应用,以防止其中敏感操作和关键数据遭到来自 ROS 中的恶意攻击.

4.3.4 实现安全启动的研究

一个系统要实现真正的安全,必须从系统启动阶段就要保证其是安全的,基于一个安全的隔离环境来保障系统的启动安全已经变得切实可行,并已经展开了大量的研究工作.

上述文献[48]提出的 HyperCheck 中就利用静态的平台配置寄存器(Platform Configuration Registers,PCRs)来保证启动过程安全.文献[59]利用 SRAM 的物理不可克隆特性(Physically Unclonable Function,PUF)为 TrustZone 平台构建了一个可靠的可信根,它修复了 TrustZone 通过在片上系统固化设备密钥来作为可信根存在更新困难,且一旦泄露会导致整个系统无法使用的弊端.另外,基于 SRAM PUF 物理特性为系统提供可信根能够抵御一定硬件和软件攻击,如抵御逆向工程和有效防止克隆等,且不需要像设备密钥需要长期存储在设备上,并基于此构造随机数发生器,免去了硬件构造带来的成本、性能和功耗的影响.并在此基础上通过在 TrustZone 提供的安全隔离环境设计了 TPM 服务,以它作为普通执行环境下操作系统的可信根,ROS 中的应用也能调用安全隔离环境中的 TPM 服务从而将信任链扩展到 ROS 的应用层,从而构建了一条从设备上电到应用程序运行的完整信任链,实现了真正的可信启动.然而,本文采用了外置的 SRAM 来构建可信根,因此很难抵御系统可能遭受到的硬件攻击.

4.4 总 结

本章对基于各种安全隔离机制的研究现状进行了分析总结,以下章节会基于此基础上剖析以上 3 种安全隔离机制的优势与不足,并将它们作对比分析.

5 对比与分析

本章主要是基于以上对 3 种隔离机制的深入分析和对安全隔离技术研究现状的充分调研,具体剖析了这些机制的优势与不足,并进一步抽象出隔离性、安全性及性能 3 个指标将各种机制作了全面的对比分析.

5.1 硬件隔离机制的优势与不足

诸如智能卡和手机 SIM 卡等外置安全硬件模块可以将敏感数据保护在一个相对安全的物理外设中,该外设在设计时可以根据安全性要求采用更加先进的工艺和安全技术.然而,外置安全硬件模块加重了 SoC 设计的开销,增加系统的功耗和降低了处理器的性能,也因为外置安全硬件模块的作用范围受到限制,这样当软件在其保护外处理敏感数据时将无

法得到安全保证，其数据极易遭受各类恶意攻击[30].

而具有更低设计成本且便于集成的内置安全硬件模块相对外置安全模块提高了系统性能，它通过使用内置的安全硬件逻辑来阻止非法进程对敏感资源的访问. 然而，它的安全作用范围也受到限制. 其中，内置安全硬件模块包含一种专为系统安全需求设计的通用处理器，它有点类似 TrustZone 技术，也不尽相同[31]. 通用处理器需要设计一个专用的安全处理器，而 TrustZone 是在原有处理器上进行的安全扩展，这样前者需要以增大硅面积和增加系统功耗为代价. 且通用处理器需要与原有处理器保持通讯，这样就要实时刷新通常处于外置的共享内存中的数据，这样就需要消耗大量的时间，并带来较大的系统功耗. 另外，内置安全硬件模块需要在芯片设计时进行专门的硬件扩展，这无形中加重了设计和测试的难度，这种方式也没有考虑在调试和测试模式下的系统安全问题，而一旦关闭这两种模式必将使得软件调试更加困难[3].

总之，纯硬件的隔离机制尽管能将关键数据和操作保护在一个安全可靠的物理设备中，然而其也存在难以根据系统安全需求进行实时更新的缺点，且更新成本高，周期长，而且作用范围比较受限. 当今功能庞大的系统对安全需求日益丰富，攻击类型也日新月异，这种方法会显得捉襟见肘，实际应用时应根据安全需求进行选择.

5.2　软件隔离机制的优势与不足

本节主要从以上对纯软件隔离机制和现有的研究现状分析的基础上得知，虽然纯软件实现的隔离机制从一定程度上解决了一定的安全问题，但其自身存在许多的不足之处. 比如虚拟机化技术提供一个相对安全可靠的隔离环境，然而它需要管理系统和分配资源，其代码量越来越大，已经不断暴露出越来越多的漏洞，其自身安全性已经很难得到保证. 且它不能保证系统在调试和测试模式下所遭受的硬件攻击，它也需要不断模拟系统的关键指令从而增加了其执行负载. 更遗憾的是，类似直接内存存取（Direct Memory Access，DMA）和图形处理器（Graphics Processing Unit，GPU）等都能绕过虚拟机提供的保护机制[3].

Docker 是一个开源的引擎，它能为任何应用创建一个轻量级、可移植的及自给自足的容器. 且它能在很多平台上部署，诸如 VMs、Bare Metal、OpenStack 集群等. 然而，在最小化需要运行的容器上，开发者需要做好足够的权衡. 一方面是需要减少容器与系统之间的分离度，而虚拟机与主机的分离性比容器会更高. Docker 的最大问题是共享命名空间的问题，命名空间是系统内核所创建的组，它通常会为不同的源和区域指定不同的访问级别，而在 Docker 中并没有为容器提供不同的命名空间. 这样，倘若有数百个容器在运行，那么每个容器都需要有独立的命名空间. 如果一个容器想要共享存储，那么所有共享这个存储的命名空间必须使用显式访问. 另外，需要以 root 权限去运行 Docker，对于用户来说可能并不知道容器中运行的是什么，这也可能会引发安全问题. 同时，太过信任从互联网上下载下来的容器也将引发诸多安全问题.

容器和虚拟化技术极其类似，但仍有不同之处，在此通过将它们对比来进一步分析它们自身的优势与不足. 容器具有轻量级特性，所需的内存空间较少，提供非常快的启动速度，而虚拟机提供了专用操作系统的安全性和更牢固的逻辑边界. 如果是虚拟机，虚拟机管理程序与硬件对话，就如同虚拟机的操作系统和应用程序构成了一个单独的物理机. 容器中所有应用都是共享主机操作系统的内核和某些库，因此它相对虚拟机提供了更高级别的隔离措施. 容器和虚拟机都具有高度可移植性，但方式不一样. 就虚拟机而言，可以在运行同一虚拟机管理程序（如 VMware ESX、Microsoft Hyper-V、Zen 或 KVM）的多个系统之间移植. 而容器不需要虚拟机管理程序，因为它与操作系统绑定在一起. 只需要有一个操作系统副本，容器中的应用程序就可以移植. 创建容器的速度要比虚拟机快的多，那是由于虚拟机必须从存储系统检索 10 GB 至 20 GB 的操作系统. 容器中的工作负载是直接使用主机服务器的操作系统内核，避免了这一步. 另外，与 VMM 相比，容器不需要虚拟化整个硬件抽象层，而只需抽象应用，所以会更加高效. 细粒度的虚拟化使得容器技术不会浪费过多的资源，这样对 CPU、内存和存储等硬件资源要求会比较低. 因此，容器在控制资源开销和管理应用运行环境是一个非常高效的工具. 而虚拟化技术能够为每个用户分配虚拟化的 CPU、内存和 IO 设备资源，因此它能为每个用户提供一个与其他用户完全隔离的系统环境.

SandBox 则是按照预定的安全策略尽可能限制不可信应用或恶意代码去损害其它应用和系统的安全，这样可以保证攻击发生在极其有限的范围内. SandBox 的 TCB（Trusted Computing Base）非常大，很难保证不存在漏洞，从而使其失效甚至被攻击

者利用.而且它依赖于严格的安全策略,而定制合适的安全策略和判定一个程序是否会损害系统的安全都是非常困难的.另外,在沙箱中也很难权衡和兼顾系统的安全与应用程序的功能完整性.

Honeypot 则是通过故意诱骗攻击者在其部署的有限范围内实施攻击从而捕获攻击者的信息和攻击行为,从而能为系统制定更加全面完善的防御机制提供可靠的来源.然而,Honeypot 自身会携带一定的预期特征或行为,可能因此遭受到恶意攻击者的攻击和利用,在应用它们的过程中如何保证其自身的安全变得尤为重要.

微内核因为简单,所以其安全可以更容易地被验证和保证.如果网络驱动不执行安全功能,那它就不被当作 TCB 的组成部分,也就不用遵循内核或其他关键安全组件的安全标准.一个安全的微内核可以在整个系统的架构中拥有多个兼容 UNIX 的文件系统实例,每个实例服务于与其他安全域相隔离的安全域.在一个域中的关键应用程序将不会因另一个域的不良软件而遭受拒绝服务的攻击.多实例服务器不需要太多的内存资源,安全微内核应当为各个域提供共享该文件服务器静态、只读部分的能力.而单内核操作系统使系统软件,如网络协议栈、文件系统和复合设备驱动等,都共享同一个内存空间,并执行于特权模式,这生成了巨大的 TCB,使得攻击者有大量的机会来发现并利用漏洞.

5.3 系统级隔离机制的优势与不足

本节分析各类系统级隔离机制的优势与不足,并将它们作对比分析.

嵌入式领域主要代表 TI M-Shield 和 ARM TrustZone.前者提供了开放的移动安全框架,也提供一个足够安全的隔离环境,从而为使用 OMAP 与 OMAP-Vox 处理器的手机创造了可靠而灵活的应用环境.然而,该技术只限于 OMAP 与 OMAP-Vox 处理器,应用平台比较受限制,另外为了便于开发者开发,它也扩展了 TrustZone 的 API.后者是在尽量不影响系统功耗和面积的前提下对原有主处理器进行了安全扩展,并配套了可信软件支持扩展硬件,它能够提供一个非常安全的隔离环境,系统中的敏感数据和操作都可以切换到这个隔离环境进行处理,从而保障其执行的安全.

TrustZone 技术在提高嵌入式系统安全方面拥有大量的技术和商业上的优势,主要分为以下几个方面进行阐述.首先,它可以为片上的保密数据提供安全的隔离环境,而这种处理方式也是目前保密的最佳途径.例如,如果想用 SoC 上的一个 CPU 来处理 SIM 卡中的密钥,必须确保在 SoC 环境中有个完全安全的区域,一个不怎么安全的 OS 是不可以做这些操作的.其次,性能一直是某些保密系统中难以克服的问题,特别是在片上处理器和片外存储器之间需要频繁传递信息时,这些信息必须经过加密.此时 TrustZone 便可发挥作用,因为它对整个存储空间都可以保证完全的总线带宽,而在其安全缓冲区数据却可以以明文的形式存储从而实现快速访问.加密过的数据则可以以普通方式存放在 FLASH 存储器中,这样可以使用一些成本低的,容量大的,灵活的存储方式.另外,TrustZone 系统架构是软硬件的合理组合,即使在 SoC 设计完成后,它依然可以保证用户能够灵活地定制和升级保密系统.最后,TrustZone 在嵌入式系统中定义了一个安全的隔离环境,该独立环境中包含一些直接的外设通道,用户界面,SIM 卡,智能卡及音频输出等.对于非保密部分,TrustZone 可以通过完整性检查机制为 SoC 器件中的所有部分提供安全保护.例如,解码后的 DRM 音频数据被传送到非保密区域中时可以通过操作系统有关部件的完整性检测来受到保护[3].

尽管 TrustZone 在系统安全方面存在显著的优势,但其仍有许多不足之处.TrustZone 没有提出怎样实现安全存储,由于缺少安全存储和,这极大地减少了 TrustZone 作为可信计算硬件的有效性.可以采用加密的方式将持续状态加密后存储到不可信存储上,但这仍然不是非常有效的方式.一方面,这需要安全存储密钥;另一方面,加密不能阻止 rollback 攻击.大多数可信系统使用加密技术,但 TrustZone 没有提供一个安全的熵源或单调递增的计数器.另外,两个世界以一种稳定的方式共享处理器应使用虚拟化技术,尽管 ARM 提供了虚拟化扩展,然而 TrustZone 没有授权给它们.TrustZone 也缺少安全时钟.其次,当 TrustZone 访问受保护的内存和中断时会提供安全外设,但这是不足够的,因为它并没有保护总线控制器.一般外设控制器能被普通世界编程,因此其外设安全很难得到保证,即使它的中断和内存域仅仅被映射到安全世界,恶意代码也能以某种方式编程这个外设使它不安全,如一些外设能在调试模式下被任意访问.再者,大多数 SoC 硬件厂商不愿意提供对他们固件的访问,这样导致许多开发者和研究者不能部署他们的系统或原型在 TrustZone 硬件上,这严重的阻碍了开发者采用 TrustZone 作为一个可信计算机制.

同时设备需要匹配可靠的可信根以保证系统的安全启动，而 TrustZone 是通过固化密钥的方式作为可信根，此方法会存在密钥更新困难，且一旦泄露会导致整个系统不可信甚至不可用的弊端，因此为其系统实现一个可靠的可信根是保证其可信执行环境安全的必要条件. 基于 SRAM PUF 技术提取密钥会是一种比较行之有效的方式，克服现存固化密钥的不足[3]. 另外，该技术只是提供了一个可信的执行环境，但并没有向外界提供其可信的证明. 为此，将 TCG 定义的 TPM/MTM 构建在其可信环境，从而能为其提供本地和远程证明，而 TrustZone 提供的隔离环境保护 TPM/MTM 的安全性，这种方式也变得切实可行. 该技术也只能防止软件攻击，而难以防止各种物理攻击. 与此同时，TrustZone 结构设计时没有授权对安全世界资源的访问. 为了访问安全世界的资源，REE 中一个合法的进程使用一个 REE 和 TEE 的信号通道，该通道由 REE 进程创建，它同步地调用具体的指令，参数被写入分配在 REE 中的共享内存. 合法的 REE 进程能通过这个通道发送请求到 TEE 并得到响应. 然而很难验证 REE 中请求调用 SMC 指令的进程是否是合法的，对一个拥有 REE 内核级权限的攻击者能创建一个恶意进程来持续发送经过精心设计的参数请求来发现 TEE 的漏洞. 另外，共享内存能被攻击者访问、修改或重分配. 当前没有有效的机制验证共享内存的完整性. 再者，REE 和 TEE 的切换是经过调用 SMC 指令，而 SMC 指令有可能被攻击者恶意替换、修改或者删除. 尽管此类攻击难度比较高，可一旦成功这样会使 TrustZone 机制瘫痪.

Intel SGX 是 Intel 最近推出的一套安全隔离技术，系统能将敏感信息和敏感操作放置在其设计的 Enclave 进行处理，从而能保证其不被攻击. 它也是 Intel 处理器的一系列扩展，为运行的应用级代码与系统的其他部分隔离开而构建了一个沙箱机制. SGX 的认证机制直接由 CPU 支持，类似于 TPM，远程实体能验证可信域和 TEE 的完整性. 远程实体能加密地验证一个 Enclave 并创建一个安全通道来传送密文. 通过这种方式，远程者能在一个 Enclave 中载入可信代码，并且后续能验证它是否已经恰当地例化在目标平台上. SGX 也能保证Enclaves之间及它们与远程实体的安全通讯，因为 SGX 提供了本地认证和远程认证两种机制. 它也能保证当一个 Enclave 被例化，硬件会对它其中的数据提供机密性和完整性保护. 当 Enclave 进程退出，这个 Enclave 将会销毁，并将其中的安全相关的数据清零. 如果这些数据需要被再次使用，则将这些数据安全地存储在 Enclave 外.

Intel SGX 与 TrustZone 类同的是，SGX 应用能创建 Enclave，Enclave 与 OS 和其余运行在这个平台上的软件都隔离开了，故能得到很好的保护. 由前述机制分析可知，所有分配给 Enclave 的内存都硬件加密了，这点与 TrustZone 有所不同. 然而，SGX 不提供 I/O 支持，所有的中断都由不可信代码处理，这点也与 TrustZone 不同. 当代码执行在 Enclave 内能加密它的状态，但加密不能防御 rollback 袭击. SGX 也没有提供一个安全计数器，然而安全计数器是构建安全系统的非常重要的一部分. 同 TrustZone，它也缺少一个安全时钟. 另外，SGX Enclave 仅仅保护代码在 Ring 3，这意味着不可信 OS 需要服务于资源管理任务，这便为边信道攻击预留了大量的攻击面. SGX 与 TrustZone 不同，前者中一个 CPU 能运行多个 Enclave 且可并发执行，而后者是将 CPU 划分为两个隔离的执行环境，两个环境是通过调用 SMC 指令实现切换. 当然，在 TrustZone 的安全世界内部实现多个相互隔离的安全服务亦可达到同样的效果. SGX 能将安全应用依赖的 TCB 减小到仅包含 CPU 和安全应用本身，将不可信的复杂 OS 和 VMM 排除在安全边界之外.

Intel TXT 和 AMD SVM 都是基于 TPM 发展起来的可信隔离系统，它构筑一条从 BIOS、MBR (Master Boot Record)、操作系统启动引导程序、操作系统内核到应用层的完整信任链，它能有效地阻止特权代码的攻击，从而增强了系统的安全性.

然而，它并没有得到普及，因为这种方式过于依赖于硬件配置. 另外，其安全性也依赖于安全控制策略，用户往往也因为这种过于严格的控制策略而禁用 TPM 的功能. 而且 TPM 需要安全非易失性内存，如果主机不具备抗干扰存储，就不能达到部署 TPM 的要求. 此时，常常部署一个软件 TPM. 另外，TPM 并不适合于对体积和功耗要求苛刻的嵌入式领域，然而可以在其隔离环境部署软件 TPM，这也是一种很好的部署可信策略的方式. 这样，可以利用隔离环境保障软件 TPM 的安全，而 TPM 则为系统提供丰富的可信计算功能.

另外，基于系统级隔离机制构建的可信执行环境是基于硬件与软件的合理配合，对系统的性能影响较小，而且易于根据不同安全需求进行再设计. 因此，该隔离技术是目前比较主流的安全隔离技术，也

是未来保障系统安全,解决系统中存在的安全问题比较好的解决方案.然而,在使用的过程中需要保证结合具体的应用场景和需求进行设计,保证其 TCB 足够小,以确保其足够安全.同时,可信执行环境在系统掉电的情况下无法发挥其安全优势,如何防止在离线状态其可能遭受的各类物理攻击也变得势在必行,特别是针对嵌入式设备易丢失且易遭盗窃的特点.这样可以防止攻击者对可信执行环境中关键代码的篡改及获取其中存储的关键数据.因此,可信执行环境虽然为系统提供了一个良好的隔离环境,但其自身也存在很多限制与不足之处,合理设计 TEE 也十分关键.

5.4 各种隔离机制对比分析

本节主要将以上 3 种隔离机制整体作对比分析,具体是抽象出隔离性、安全性及性能等 3 个关键指标来进行整体比较,具体如表 1 所示.

表 1 各类隔离机制对比

隔离机制类型	隔离性	安全性	性能(包括功耗和面积)
硬件隔离技术	隔离性中:纯硬件的隔离模块能很好地将敏感数据保护在可靠的物理设备中,并可采用更加先进的防篡改技术.	安全性中:当软件运行在外置安全硬件模块和内置加密模块之外时,安全范围受到限制,而提供专用的通用处理器没有考虑在调试和测试模式下的系统安全.但因为设计成本高和周期长,不能实时满足新的安全需求.	性能中:纯硬件加密模块会增加系统的功耗和硅设计面积,通用安全处理器则因为需要与主处理器通讯而影响系统性能.故硬件隔离给系统的性能带来一定的影响.
软件隔离技术	隔离性低:纯软件的机制能在系统中提供一个相对安全的隔离环境.然而,它的隔离性会受到其它软件或系统安全的影响.	安全性低:纯软件的隔离机制,如沙箱集成在内核中的安全机制,会因为 TCB 过大,导致自身极易受到恶意攻击,而虚拟化技术自身需要管理许多系统资源,自身代码量已经非常庞大,存在较大的安全隐患.容器也存在类似的问题.故软件隔离技术自身的安全性得不到保证.	性能低:对系统的功耗没有什么影响,然而会一定程度上增加系统的负载,它也不需要重新设计硬件,因此开发成本比较小且周期短.然而,软件实现的隔离机制运行速度较硬件会慢很多,因此性能比较低.
系统级隔离技术	隔离性高:基于软硬件的合理配合能够在系统中提供一个相对稳定且安全的隔离环境,硬件实施访问限制,并能隔离敏感代码,实现安全启动及构筑可信链等,其隔离性很好.	安全性高:其中 Intel SGX 和 ARM TrustZone 能够分别将 Enclave 和 TOS 与平台上的软件和都隔离开,TI M-Shield 也能提供类似 TrustZone 的隔离保护,然而它只针对限定的处理器,而基于 TPM 的 Intel TXT 和 AMD SVM 也能达到同样的效果,但其要求严格的访问控制策略.故系统级隔离技术安全性最高,它能保护系统中的关键数据和操作的安全.	性能高:ARM TrustZone 在原有处理器上进行安全扩展,对系统功耗、面积和性能影响小.而基于 TPM 构建的隔离运行环境则对系统功耗和性能都有一定的影响,其对硬件配置依赖性强.故系统级隔离技术的设计对系统的性能影响比较小,相对软件隔离和硬件隔离,性能最高.

另外,也有针对安全技术的隔离性进行评估验证方面的研究,Jeanna 等人[60]就提出了一个隔离的基准测试套件,它的目的是量化虚拟化系统,并限制一些行为不端的虚拟机影响其它与其运行在同一物理机上行为正常的虚拟机的能力.该测试套件主要包括:CPU 密集测试,内存密集测试,磁盘密集测试,网络发送和接收密集测试及 fork 炸弹(fork bomb)测试,并展示了全虚拟化的 VMware Workstation,半虚拟化实例 Xen 和操作系统级虚拟化实例 Solaris 和 OpenVZ 的测试结果.测试结果展示全虚拟化系统在所有的情况下提供了完整的隔离,半虚拟化系统达到了同样的隔离效果,只是在磁盘密集测试中有至多 1.7%的下降.操作系统级的虚拟化系统测试结果则是多种多样的,这也说明了在一个紧耦合的操作系统中实现所有资源隔离的复杂性.

6 安全隔离技术研究和应用需求展望

系统环境中构建一个安全的隔离运行环境已经成为一种主流的趋势,在敏感数据处理和关键数据保护方面起着极其重要的作用.本章针对系统的现有安全问题,展望了安全隔离技术未来的研究方向及发展趋势,总结归纳出以下 3 个方面的工作.

6.1 安全隔离解决云计算安全问题

随着当今对海量数据和复杂计算的日益需求,计算模式已经从集中式向分布式演变,云计算作为一项新兴的计算模式逐渐成为学术界和产业界的热点和焦点.云计算可以根据用户的特定需求并通过整合分布式资源来构建满足各种服务要求的计算环境,并可以通过网络分别对其进行访问.它能有效地实现资源共享,从而便利了系统管理与维护,也确保了服务的正常使用.然而,确保这些服务和数据的机密性、完整性、可用性及隐私性仍然是其核心需求,需要针对云计算系统及应用的特点进行安全防护.它的防护不应是开发全新的安全理念或体系,而应从传统安全管理角度出发,结合其特点将现有成熟的安全技术及机制延伸到云计算应用及安全管理中,满足云计算应用的安全防护需求.

而安全隔离技术提供的隔离运行环境能够解决云计算所面临的一系列安全问题,用户的敏感操作和隐私数据都能隔离在相对独立的执行环境,用户之间互不影响,这样可以有效的保护用户的隐私,它也是保证云计算环境安全的有效措施.如采用虚拟机化技术,它能在逻辑上实现系统、存储及网络等的

安全隔离.硬件比纯软件实现逻辑隔离的更为安全，它能实现各个环节的高效隔离来保证用户隐私数据的安全性.文献[61]中就提到当前云计算安全隔离架构主要围绕硬件和软件隔离两个方面来实施，其目的是主要为用户提供云计算整个链路层的隔离.虚拟化技术是实现云计算的关键核心技术，使用虚拟化技术的云计算平台上的云架构提供者必须向其客户提供安全性和隔离性.

6.2　安全隔离与可信计算结合

基于安全隔离提供隔离运行环境，可以将可信计算模块构建在该隔离环境中，利用隔离的安全优势保障可信模块的安全，而可信模块则为系统提供丰富的可信计算功能，这种结合已是一种切实可行的方案.

TPM 已应用于 PC 领域，然而它并不适合对功耗和面积要求苛刻的嵌入式领域.虽然 TCG 随后也针对嵌入式和移动领域发布了 MTM(Mobile Trusted Module)规范.然而，限于移动平台形态的多样性，TCG 在发布相应可信平台移动平台规范时并没有明确 MTM 的具体实现方式，也未对 MTM 的实现给出明确的定义，MTM 仅是功能而非硬件实现.幸运的是，可基于以上提到的各种隔离技术来构建可信移动平台，如前述章节提到的智能卡 JavaCard 或 ARM TrustZone.具体是，基于 JavaCard 实现 MTM 能利用智能卡自身的硬件保护机制防范攻击者对 MTM 功能实现的篡改，且继承了智能卡固有的安全属性.同时，对智能卡安全性评估目前已经有非常成熟的方法，这也为基于智能卡实现 MTM 的安全性检测评估提供了便利.与此同时，MTM 的两种功能 MRTM(Mobile Remote-Owner Trusted Module)和 MLTM(Mobile Local-Owner Trusted Module)是以 Java 小应用程序的形式并发地运行于 JavaCard 内部的虚拟机上，利用虚拟机提供的安全机制能够确保它们的隔离运行[62].基于 TrustZone 实现的 MTM，其硬件级别的隔离运行环境能够保证 MTM 自身的安全性，又可以通过环境切换使非安全域的应用程序能够访问 MTM 功能.同时，基于 TrustZone 的 MTM 所构建的可信移动平台能够实现安全启动.具体是，系统上电后首先执行固化在 ROM 里的加载程序 Secure Bootloader，该程序由 TrustZone 硬件保证不被篡改，随之将控制权交给 MTM，然后 MTM 对后续加载程序执行先度量再传递控制权的过程，最终构建一个完整的移动平台安全启动链.文献[63]就将 TPM 2.0 与 TrustZone 结合实现相应功能，具体使用 TrustZone 提供的硬件隔离环境保护 TPM 2.0 的安全，而 TPM 则为平台提供丰富的可信计算功能，从而构筑一个可信的计算平台.

6.3　安全隔离实现系统防护

系统现在部署的传统操作系统，如 Linux 或 Windows，继承了它们的丰富功能的和无法避免的安全威胁.这是因为这些操作系统代码量已经非常大，很容易存在潜在的漏洞而被攻击者利用.同时，这些系统有许多特权进程，一旦受到攻击损坏，很容易安装内核 rootkits，这样敌手就能轻易访问系统的敏感数据，甚至完全控制整个操作系统的执行，比如 Android 系统中特权进程的漏洞也曾有过很多报道.为此，基于系统的各类隔离运行环境来防护系统的安全，保护系统的敏感数据和关键操作不被攻击，仍是目前研究的热点.

如基于硬件抽象层虚拟机就能够很好地在主操作系统中隔离一个相对可信的环境，通过在该隔离环境中构建检测防御机制能有效地检测并阻止主操作系统中特权代码实施的各类攻击.文献[64]就提出一种轻量级的安全可信的虚拟执行环境，通过利用 CPU 的系统管理模式(SMM)提供的硬件隔离特性来隔离应用程序的安全敏感代码.然而，由于虚拟层只能获取硬件层的信息，如何将这些信息转化成操作系统所需要的语义信息也是亟待解决的问题.也因为虚拟机是在操作系统外新引入的一个高权限的软件层，它给当前的系统安全结构引入了新的问题.一方面，一些攻击者能够针对当前虚拟机来设计攻击，甚至是伪造一个虚拟机，这样 OS 将无法完全发现和防范这类攻击.另一方面，引入了 VMM 的软件层后，如何维持 TPM 构建的信任链，也是一个非常棘手的难题.

因此，基于硬件支持的可信执行环境去解决系统存在的安全问题变得更加切实可行，如 ARM TrustZone 或 Intel SGX，通过在隔离环境设计检测和防御机制来保证 ROS 的安全.这些检测和防御机制与 ROS 完全隔离，所以即使在 ROS 受到损害或者毁坏也能保证自身的安全，敏感数据、系统关键功能及关键操作都可以放在这个可信执行环境中执行.同时，也可以根据具体的安全需求和应用场景在可信执行环境中设计特定的安全服务来确保 ROS 的安全.另外，可以在其隔离环境中实现一套与 ROS 完全隔离开的安全显示与安全输入来保证用户信息输入和显示的安全.之前章节提到的 TZ-KRP 就基

于 TrustZone 的隔离运行环境实现了系统的是实时内核防护的机制，解决了一些系统存在安全漏洞的问题.

同时，针对移动设备容易丢失和遭窃取的特性，这使得它极易受到廉价的内存攻击，比如冷启动攻击，如使用总线监控器来监控内存总线，并实施 DMA 攻击. 因此，防范设备可能遭受到的物理攻击以保护系统关键数据的安全也成为未来研究的重点内容. 文献[65]就提出一种数据保护机制，它能防止内存遭受到的冷攻击，该系统允许应用和 OS 组件存储它们的数据和代码在 SoC 而不是 DRAM. 另外，基于 SRAM PUF[3] 提取密钥代替固化密钥作为可信根也是研究的热点，它能使设备克服一定的物理攻击，如抵御逆向工程和有效防止克隆等.

7 结 论

当设备集成各种安全敏感的服务，如钱包、访问控制令牌、远程银行甚至是盗窃保护装置，增加设备的可信性是非常重要的. 而我们的日常生活越来越依赖于这些服务，它给我们带来了极大便携性的同时，也产生了非常广泛的安全问题. 因此，操作系统安全性需要加强，应用代码的远程配置，固件更新及关键数据也需要可靠的执行. 数据的备份和恢复，安全应用数据存储及商业和个人使用之间的防火墙都是常用的功能要求. 用户及应用编程人员的可用性也是另一项非常重要的功能. 为此，在系统中提出一个稳定可靠安全的隔离运行环境是至关重要的，对保障系统关键功能及敏感数据是极其必要的. 本文首先对各类安全隔离技术作了一个整体概述，接着细致分析了各种安全隔离机制的实现方式，对它们的研究现状进行了分析总结，并基于此基础上对它们各自在解决系统安全方面的优势与不足作了深入的剖析，同时将各类安全隔离机制通过抽象出 3 个指标进行了整体对比分析，最后结合目前信息安全领域中存在的各种安全问题，展望了安全隔离技术的未来发展趋势、学术研究及应用情况.

致 谢 评审专家和编辑部老师为本文提出了宝贵的意见和建议，作者在此表示衷心的感谢！

参 考 文 献

[1] Wen Yan. Research on the Key Technologies of Isolated Execution Environment[Ph. D. dissertation]. Graduate School of National University of Defense Technology, Changsha, 2008(in Chinese)
(温研. 隔离运行环境关键技术研究[博士学位论文]. 国防科学技术大学研究生院，长沙，2008)

[2] Jan-Eric E. Securing Software Architecture for Trusted Processor Environments[Ph. D. dissertation]. Aalto University, Finland, 2013

[3] Zheng Xian-Yi, Li Wen, Meng Dan. Analysis and research on TrustZone technology. Chinese Journal of Computers, 2016, 39(9): 1912-1928(in Chinese)
(郑显义，李文，孟丹. TrustZone 技术与分析. 计算机学报，2016，39(9)：1912-1928)

[4] Nathan D, Theodoros K, et al. Nested kernel: An operating system architecture for intra-kernel privilege separating// Proceedings of the 20th International Conference on Architecture Support for Programming Languages and Operating Systems. New York, USA, 2015: 191-206

[5] David K, Mike K. Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development. Holland: Elsevier, 2012

[6] Srivaths R, Anand R, et al. Security in embedded systems: Design challenges. ACM Transactions on Embedded Computing Systems, 2004, 3(3): 461-491

[7] Azab A M, Swidowski K, et al. SKEE: A lightweight secure kernel-level execution environment for ARM//Proceedings of the Network & Distributed System Security Symposium 2016. San Diego, USA, 2016: 1-15

[8] TCG. TCG Specification TPM 2.0 Mobile Reference Architecture. White Paper, 2014

[9] GlobalPlatform Inc. GlobalPlatform Device Technology TEE System Architecture Version 1.0. White Paper, 2012

[10] GlobalPlatform Inc. GlobalPlatform Device Technology TEE Client API Specification Version 1.0. White Paper, 2010

[11] GlobalPlatform Inc. GlobalPlatform Device Technology TEE Internal API Specification Version 1.0. White Paper, 2011

[12] GlobalPlatform Inc. GlobalPlatform Device Technology Trusted User Interface API Version 1.0. White Paper, 2013

[13] Andreas F. Development of an ARM TrustZone Aware Operating System ANDIX OS[M. S. dissertation]. Graz University of Technology of Austria, Styria, Austria, 2014

[14] Jitesh H S. ARMMithril: A Secure OS Leveraging ARM's TrustZone Technology[M. S. dissertation]. North Carolina State University, Raleigh, America, 2012

[15] Li Wen-Hao. T6, an Operating System for TrustZone Based Trusted Execution Environment(TEE) in ARM-Based Systems. White Paper, 2014

[16] Brian M, Tanel D, Thomas N. Open-TEE—An open virtual trusted execution environment//Proceedings of the 14th IEEE International Conference on Trust, Security, and Privacy in Computing and Communications, TrustCom 2015. Helsinki, Finland, 2015: 1-11

[17] Sierraware. Open Virtualization Build and Boot Guide for ARM V7 and ARM V8. White Paper, 2014

[18] Javier G. Operating System Support for Run-Time Security with a Trusted Execution Environment [Ph. D. dissertation]. IT University of Copenhagen, Copenhagen, The Kingdom of Denmark, 2015

[19] Azab A M, Peng Ning, et al. HIMA: A hypervisor-based integrity measurement agent//Proceedings of the 25th Annual Computer Security Applications Conference. Oahu, Honolulu, 2009: 461-470

[20] John C, Andrew L, et al. Secure virtual architecture: A safe execution environment for commodity operating systems// Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles. New York, USA, 2001: 351-366

[21] Tal G, Ben P, et al. Terra: A virtual machine-based platform for trusted computing//Proceedings of the 19th ACM Symposium on Operating Systems Principle. New York, USA, 2003: 193-206

[22] Dirk M. Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014, 2014 (239): 1-3

[23] Carl B. An introduction to docker for reproducible research. ACM SIGOPS Operating Systems Review, 2015, 49(1): 71-79

[24] Justin C, Armon D, et al. Retaining sandbox containment despite bugs in privileged memory-safe code//Proceedings of the 17th ACM Conference on Computer and Communications Security. New York, USA, 2010: 212-223

[25] Iyatiti M, Michele A. Honeypots: Concepts, approaches, and challenges//Proceedings of the 45th Annual Southeast Regional Conference. New York, USA, 2007: 321-326

[26] Udo S, Bernhard K. NOVA: A microhypervisor-based secure virtualization architecture//Proceedings of the 5th European Conference on Computer Systems. New York, USA, 2010: 209-222

[27] Eric L, Vincent N, Yves D. Enforcing kernel constraints by hardware-assisted virtualization. Journal in Computer Virology, 2011, 7(1): 1-21

[28] Matthias L, Steffen L, et al. L4Android: A generic operating system framework for secure smartphones//Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. New York, USA, 2011: 39-50

[29] Gerwin K, Kevin E, et al. SeL4: Formal verification of an OS kernel//Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. New York, USA, 2009: 207-220

[30] ARM Limited. ARM Security Technology Building a Secure System using TrustZone Technology. White Paper, 2009

[31] Tiago A, Don F. TrustZone: Integrated hardware and software security enabling trusted computing in embedded system. Government Information Quarterly, 2004, 3(4): 18-24

[32] Texas Instruments. M-Shield Mobile Security Technology: Making Wireless Secure. White Paper, 2008

[33] Intel Corporation. Intel Trusted Execution Technology: Hardware-Based Technology for Enhancing Server Platform Security. White Paper, 2012

[34] AMD Inc. AMD64 Virtualization Codenamed "Pacifica" Technology: Secure Virtualization Machine Architecture Reference Manual. White Paper, 2005

[35] Intel. Software Guard Extensions Programming Reference. White Paper, 2013

[36] Frank M, IIya A, et al. Innovative instrustions and software model for isolated execution//Proceedings of the 2nd International Workshop on Hardware and Architecture Support for Security and Privacy. New York, USA, 2013: 1-8

[37] Matthew H, Reshma L, et al. Using innovative instructions to create trustworthy software solutions//Proceedings of the 2nd International Workshop on Hardware and Architecture Support for Security and Privacy. New York, USA, 2013: 9-16

[38] Ittai A, Shay G, et al. Innovative technology for CPU based attestation and sealing//Proceedings of the 2nd International Workshop on Hardware and Architecture Support for Security and Privacy. New York, USA, 2013: 17-23

[39] Ryan R, Jiang Xu-Xian, Xu Dong-Yan. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing//Proceedings of the 11th International Symposium, RAID. Cambridge, USA, 2008: 1-20

[40] Arvind S, Mark L, et al. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes// Proceedings of the 21st ACM SIGPOS Symposium on Operating Systems Principles. New York, USA, 2007: 335-350

[41] Wang Xiao-Guang, Chen Yue, et al. SecPod: A framework for virtualization-based security systems//Proceedings of the USENIX Annual Technical Conference. Santa Clara, USA, 2015: 346-360

[42] Abhinav S, Ikpeme F, Jonathon G. Kernel data integrity protection via memory access control. Atlanta, Georgia: Georgia Institute of Technology, Technical Report: GT-CS-09-04, 2009

[43] Xiong Xi, Tian Dong-Hai, Liu Peng. Practical protection of kernel integrity for commodity OS from untrusted extensions //Proceedings of the 18th Annual Network and Distributed System Security Symposium. San Diego, America, 2011: 1-17

[44] Hua Jing-Yu, Kouichi S. Barrier: A lightweight hypervisor for protecting kernel integrity via memory isolation// Proceedings of the 27th Annual ACM Symposium on Applied Computing. New York, USA, 2012: 1470-1477

[45] Michael G, Wang Zhi, et al. Transparent protection of commodity OS kernels using hardware virtualization// Proceedings of the 6th International ICST Conference, SecureComm. Singapore, 2010: 162-180

[46] Song Cheng-Yu, Lee B, et al. Enforcing kernel security invariants with data flow integrity//Proceedings of the Network and Distributed System Security Symposium 2016. San Diego, USA, 2016: 1-15

[47] Ahmed M A, Peng Ning, et al. HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity//Proceedings of the 17th ACM Conference on Computer and Communication Security. New York, USA, 2010: 38-49

[48] Wang Jiang, Angelos S, Anup G. HyperCheck: A hardware-assisted integrity monitor//Proceedings of the 13th International Symposium, RAID. Ottawa, Canada, 2010: 158-177

[49] Rutrowska J, Wojtczuk R. Preventing and detecting Xen hypervisor subversions. Las Vegas, NV: Invisible Things Lab, Technical Report: Blackhat Briefings USA 2008, 2008

[50] Tamas K L, Thomas K, et al. Multi-tiered security architecture for ARM via the virtualization and security extensions//Proceedings of the 25th International Workshop on Database and Expert Systems Application. Munich, Germany, 2014: 308-312

[51] Jonathan M M, Bryan P, et al. Flicker: An execution infrastructure for TCB minimization//Proceedings of the ACM European Conference on Computer Systems. Glasgow, UK, 2008: 315-328

[52] Jonathan M M, Li Yan-Lin, et al. TrustVisor: Efficient TCB reduction and attestation//Proceedings of the 2010 IEEE Symposium on Security and Privacy. Washington, USA, 2010: 143-158

[53] Ahmed M A, Peng Ning, et al. Hypervision across worlds: Real-time kernel protection from the ARM TrustZone secure world//Proceedings of the 21st ACM Conference on Computer and Communication Security. New York, USA, 2014: 90-102

[54] Ge Xin-Yang, Hayawardh V, Trent J. SPROBES: Enforcing kernel code integirt on the TrustZone architecture//Proceedings of the 3rd Workshop on Mobile Security Technologies (MoST). San Jose, USA, 2014: 1-10

[55] Chen Xiao-Xin, Tal G, et al. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems//Proceedings of the 13th International Conference on Architecture Support for Programing Languages and Operating Systems. New York, USA, 2008: 2-13

[56] Sun He, Sun Kun, et al. TrustOTP: Transforming smartphones into secure one-time password tokens//Proceedings of the 22nd ACM SIGSAC on Computer and Communications Security. New York, USA, 2015: 976-988

[57] Li Wen-Hao, Li Hai-Bo, et al. AdAttester: Secure online mobile advertisement attestation using TrustZone//Proceedings of the 13th International Conference on Mobile Systems, Application, and Services. Florence, Italy, 2015: 1-14

[58] Yang Bo, Yang Kang, et al. DAA-TZ: An efficient DAA scheme for mobile devices using ARM TrustZone//Proceedings of the 8th International Conference, TRUST 2015. Heraklion, Greece, 2015: 209-227

[59] Zhao Shi-Jun, Zhang Qian-Ying, et al. Providing root of trust for ARM TrustZone using on-chip SRAM//Proceedings of the 4th International Workshop on Trustworthy Embedded Devices. New York, USA, 2014: 25-36

[60] Jeanna N M, Hu Wen-Jun, et al. Quantifying the performance isolation properties of virtualization systems//Proceedings of the 2007 Workshop on Experiment Computer Science. New York, USA, 2007: 1-9

[61] Lin Chuang, Su Wen-Bo, Meng Kun, et al. Cloud computing security: Architecture, mechanism and model evaluation. Chinese Journal of Computers, 2013, 36(9): 1765-1784(in Chinese)
(林闯, 苏文博, 孟坤等. 云计算安全: 架构、机制与模型评价. 计算机学报, 2013, 36(9): 1765-1784)

[62] Feng Deng-Guo. The Theory and Practice of Trusted Computing. Beijing: Tsinghua University Press, 2013(in Chinese)
(冯登国. 可信计算理论与实践. 北京: 清华大学出版社, 2013)

[63] Raj H, Saroiu S, et al. fTPM: A Firmware-based TPM 2.0 Implementation. White Paper, 2015

[64] Chen Hao, Sun Jian-Hua, Liu Chen, Li Hai-Wei. A lightweight secure and trusted virtual execution environment. China Science, 2012, 42(5): 617-633(in Chinese)
(陈浩, 孙建华, 刘琛, 李海伟. 一种轻量级安全可信的虚拟执行环境. 中国科学, 2012, 42(5): 617-633)

[65] Patrick C, Zhang Jia-Wen, et al. Protecting data on smartphones and tables from memory attacks//Proceedings of the 20th International Conference on Architecture Support for Programming Languages and Operating Systems. New York, USA, 2015: 177-189

ZHENG Xian-Yi, born in 1986, Ph. D. candidate. His main research interests include computer architecture and network and system security.

SHI Gang, born in 1974, Ph. D., professor. His main research interests include computer architecture and embedded system.

MENG Dan, born in 1965, Ph. D., professor. His main research interests include computer architecture, cloud computing, network and system security.

Background

Despite recent advances in systems security, attacks that compromise ROS kernel still pose a real threat. Such attacks can access system sensitive data, hide malicious activities, escalate the privilege of malicious processes, change the OS behavior or even take control of the system. However, monolithic commodity operating systems, like Windows, Mac OS X, Linux, and FreeBSD, lack sufficient protection mechanisms with which to adhere to these critical information protection design principles. As a result, these OS kernels define and store access control policies in main memory which any code executing within the kernel can modify. In order to realize the real kernel protection and the key data protection, the security mechanism should be isolated and protected, so an isolated operating environment is essential.

So far, researchers around the world have done some research work on this topic and have proposed some security strategies for protecting system security based on security isolation technology. However, none of them have yet made any efforts on analyzing and evaluating these isolation technologies. It is important how to apply the technology to more system architectures and more application fields, but it also make more difficulties to researchers and developers.

In this paper, we firstly introduce kinds of security isolation technologies, analyze their security extensions in detail and divide these security isolation technologies into three categories. Then, we make thoroughly analysis to their security mechanism. We also make a summary to the existing researches and commercial applications of secure isolation technology at home and abroad from application and academic perspective. At the same time, we analyze and compare these isolation methods, and indicate their advantages and disadvantages. Finally, we prospect the development direction and application requirement of secure isolation technology combining with the existing research ideas and the existing system security problems.

This work is supported by the National High Technology Research and Development Program (863 Program) of China under Grant No. 2012AA01A401 and the National Science and Technology Major Project of China under Grant No. 2013ZX01029003-001.