轻量级虚拟化技术安全研究综述

孔 同[1,2,3], 王利明[1], 徐 震[1], 马多贺[1]

[1] 中国科学院信息工程研究所 北京 中国 100093
[2] 中国科学院大学 网络空间安全学院 北京 中国 100049
[3] 国家工业信息安全发展研究中心 北京 中国 100040

摘要 随着以容器技术为代表的轻量级虚拟化技术飞速发展, 其在云计算领域中的地位也越来越重要。轻量级虚拟化技术不为虚拟实例创建单独的操作系统, 而是使用各种内核机制来进行实现 CPU、内存、网络和文件系统的隔离, 可以更高效、灵活地实现硬件基础设施资源的充分利用、合理分配和有效调度, 为云计算带来了云原生等新的技术架构和运维模式。同时由于同一宿主机上的轻量化虚拟实例间共享操作系统内核、缺乏针对镜像库的有效检测手段等, 轻量级虚拟化技术相较于传统虚拟机技术安全隔离手段较弱且引入了新的安全风险, 为云计算技术带来了新的安全挑战, 引起学术界和工业界的广泛关注, 但其安全性缺少系统性的研究。为体系化了解轻量级虚拟化技术的安全研究进展和现状, 本文对轻量级虚拟化技术的安全问题以及解决方案进行了深入研究分析。首先对轻量级虚拟化技术的架构特点和应用场景进行了概述, 按照分层模型对轻量级虚拟实例层、宿主机层及硬件层等对象面临的攻击威胁进行了分类综述, 并概述了镜像库及其他配套系统存在的安全脆弱性。然后, 根据安全解决方案所属的系统层次对已有的安全防御方法和机制进行了深入介绍, 并对其防御原理、可应对的网络攻击类型、实现方案及优缺点进行了详细分析和总结。最后, 展望了轻量级虚拟化技术安全未来的发展趋势和后续的研究方向, 认为强化虚拟隔离、保障镜像安全检测、统一安全评估技术标准是提高轻量级虚拟化技术安全性的有效方法。

关键词 云计算; 轻量级虚拟化; 容器技术; 网络安全
中图法分类号 TP309 DOI 号 10.19363/J.cnki.cn10-1380/tn.2026.03.17

Survey on Lightweight Virtualization Technology Security

KONG Tong[1,2,3], WANG Liming[1], XU Zhen[1], MA Duohe[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
[3] China Industrial Control System Cyber Emergency Response Team, Beijing, 100040, China

Abstract With the rapid development of lightweight virtualization technology represented by container technology, its position in the cloud computing is becoming more and more important. Lightweight virtualization technology does not create independent operating systems for virtual instances but uses some kernel features to realize the isolation of CPU, memory, network, and file system, which can achieve the full utilization, reasonable allocation, and effective scheduling of hardware resources more efficiently and flexibly. It has brought new technical architectures and operation and maintenance models such as cloud-native to the cloud computing industry. Meanwhile, due to lightweight virtual instances on the same host machine sharing the kernel of the operating system and images in the public repository lack effective security detection, the security isolation mechanism of lightweight virtualization technology is weaker than traditional virtual machine technology. It also brought about new security risks and introduced new security challenges to cloud computing technology, which have received widespread attention in both academia and industry. But its security problems lack systematic research. To understand the security research progress of lightweight virtualization technology, this paper deeply studies and analyzes the security problems and solutions of lightweight virtualization technology. Firstly, we introduce the architecture and application scenarios of lightweight virtualization technology. And we summarize the attack threats of the lightweight virtual instance layer, host machine layer, and hardware layer by the hierarchical model, and generalize the security vulnerability of the image repository and other auxiliary systems. Then, the principle, implementation scheme, types of network attacks that can be defended against, advantages and disadvantages of the existing security defense methods and mechanisms are introduced and analyzed. Finally, this survey paper discusses the future work and suggested security research directions of lightweight virtualization technology. We believe that it is an effective method to improve the security of lightweight virtualization technology by enhancing virtual isolation, ensuring image security detection, and unifying security evaluation criteria.

Key words cloud computing; lightweight virtualization; container technology; network security

通讯作者: 王利明, 博士, 正高级工程师, Email: wangliming@iie.ac.cn。
本课题得到国家重点研发计划项目(No. 2019YFB1005200)的资助。
收稿日期: 2020-12-24; 修改日期: 2021-03-08; 定稿日期: 2023-08-14

1 引言

虚拟化技术是云计算中最为重要的核心技术。通过将物理设备划分为多个虚拟化设备, 多个用户可以共享相同的物理资源池, 灵活调配所需的计算任务。近年来, 轻量级虚拟化技术得到了快速的发展, 在很多应用场景中逐步替代了传统的虚拟机[1]。工业和信息化部2019年发布的《云原生技术实践白皮书》显示, 以容器技术为代表的轻量级虚拟化技术在国内云计算产业的普及率已经达到了90.9%。同时, 各大国际互联网公司也在积极开展相关服务, 既丰富了平台即服务(Platform as a Service, PaaS)的传统服务模式, 又诞生了容器即服务(Container as a Service, CaaS)的新的服务模式[2]。轻量级虚拟化相对于传统虚拟机有着更低的虚拟化开销和更短的启动时间, 同时提供了一致、可移植的软件环境。这使得云服务可以忽略不同平台之间的差异, 灵活地在任何地方运行[3-5]。这些优势进一步推动了微服务[6]体系架构的快速发展, 帮助云服务简化了更新、调度和扩展等操作过程, 对各种差异性服务场景提供了可靠的技术支持[7]。

伴随着轻量级虚拟化技术的飞速发展和大量应用, 其所面临的安全问题也逐渐显露, 得到了工业界和学术界的广泛关注[8-11]。大部分轻量级虚拟化实现方案采取了以 Linux 内核机制为基础的系统隔离方案或使用了精简的虚拟化层, 相对于传统虚拟机而言隔离性不够全面和彻底, 导致了虚拟化实例可能存在更多的安全漏洞, 面临着更大的安全威胁。另外, 由于轻量级虚拟化技术和传统的虚拟机技术所采用的技术架构具有较大的区别, 因此很多传统的虚拟化安全解决方法无法适用于新的基础设施。这就要求研究人员必须针对轻量级虚拟化技术并结合其所在的应用场景提出新的安全机制和防御方案[12-13], 抵御来源于多种不同渠道、使用各种不同方法的攻击行为。

尽管轻量级虚拟化的安全问题已经引起了安全界的广泛关注, 大量的相关研究成果也已经出现, 但是国内尚未有对于相关安全问题的全面总结和讨论。本文将以云计算环境中轻量级虚拟化技术面临的安全威胁和使用到的安全防护方法为基础, 综述轻量级虚拟化技术安全的研究进展。第 2 节介绍了轻量级虚拟化技术的技术特点和应用场景。第 3 节根据轻量级虚拟化系统中潜在的被攻击目标所处的层次对其面临的安全威胁进行了分类, 并总结了现有的攻击方式。第 4 节介绍了目前已有的安全防御方法和机制, 讨论了其适用范围及优缺点。第 5 节总结了轻量级虚拟化安全未来的发展趋势和后续的研究方向。第 6 节对全文进行了总结。

2 轻量级虚拟化技术概述

2.1 技术架构

轻量级虚拟化技术是一种操作系统级的虚拟化技术, 可以提供进程级别的隔离和管控。表 1 汇总了现有的轻量级虚拟化工具。和传统的虚拟机不同, 轻量级虚拟化引擎不需要 Hypervisors 层来实现虚拟化的硬件资源和运行平台, 也不需要为虚拟实例创建单独的操作系统, 而是使用宿主机的操作系统来管理系统资源和执行系统调用, 并使用各种内核机制来实现 CPU、内存、网络和文件系统的隔离, 在此基础上为同一个虚拟实例内的进程模拟一个几乎完整的操作系统[14]。图 1 对于两者之间的架构对比进行了展示。对这些机制使得轻量化的虚拟实例具有更快的处理速度和更低的系统负载, 进而实现跨平台的灵活迁移和运行[15-16]。

表 1 轻量级虚拟化工具汇总

Table 1 Summary of lightweight virtualization tools

类型	名称	实现技术
虚拟化引擎	Docker, Rocket, LXD, Podman, OpenVZ, Cri-o, Singularity, FreeBSD Jail, Linux Vserver, Solaris Container	容器
	Firecracker	MicroVM
	Kata, Nabla	容器+MicroVM
	gVisor	容器+沙箱
编排工具	Kubernetes, Openshift, Swarm, Mesos, Rancher	-
云平台	Azure Kubernetes Service, AKS Azure Container Instances, ACI Elastic Container Service, ECS Elastic Container Service for Kubernetes, EKS Google Kubernetes Engine, GKE Ali Elastic Container Instance, ECI	-

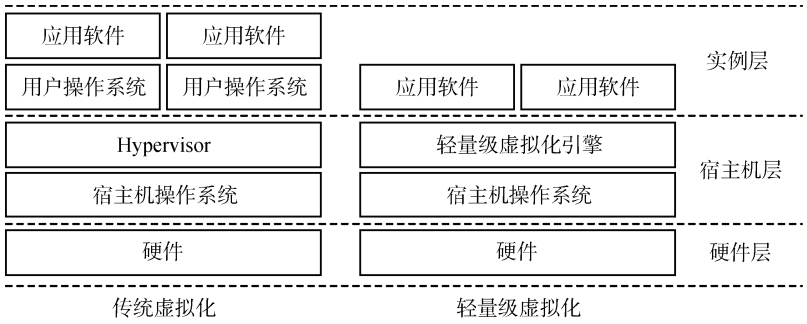

图 1　传统虚拟化和轻量级虚拟化架构对比

Figure1　The comparison between traditional and lightweight virtualization architectures

随着轻量级虚拟化技术的推广, 各类基于容器(Container)技术、轻量级虚拟机(MicroVM)技术和沙箱(SandBox)技术的实现方案层出不穷, 特别是以Docker[17]为代表的容器引擎和以 Kubernetes[18]为代表的容器编排管理工具被广泛应用, 并催生了微服务架构的产生[19]。微服务架构将传统的集成化的应用解构成为多个微服务, 每个微服务被装载在一个单独的容器中, 来提供一个简单明确的服务功能。在云场景中, 一系列的微服务互相配合来形成一个服务链, 为用户提供完整的功能。得益于轻量级虚拟化技术带来的灵活、轻量的特点, 微服务架构可以使云服务变得更加易于部署、升级和维护, 为更高频率的服务更新迭代提供了可能。

2.2　轻量级虚拟化的应用场景

轻量级虚拟化技术极大地提升了多种场景下系统的运行效率。例如容器技术为软件运行提供了一整套平台无关的标准化技术, 只要把需要使用的各种服务打包成标准的格式, 就可以在不同的平台环境下运行, 达到即插即用的效果。例如在 Web 应用服务中, 可以先将 Java 运行环境、Web 服务器直接打包成一个通用的基础容器镜像, 再将每个服务具体需要使用的应用代码程序或编译程序包加入到该镜像中产生一个新的应用镜像, 最后使用容器引擎加载镜像即可以容器的形式迅速启动所需的 Web 应用服务。

同时, 相关编排工具简化了系统资源的统一管理和调度, 为用户自动化地整理、打包配置文件和依赖关系, 方便用户快速构建实验、生产环境, 极大地降低了系统部署和运维流程的复杂度。这一优势在物联网[20-23]、边缘计算[24-25]和区块链[26]等新兴技术和平台中均得到显著的体现。这些相关平台本身都比较复杂, 需要烦琐的步骤进行搭建和维护, 利用轻量级虚拟化相关技术和平台可以减少开发者在系统建设上的投入, 使其将更多的时间和精力投入到上层应用的开发和业务功能的创新当中。

日益完善的生态链推动了新的软件开发模式的发展。DevOps 是一种新兴的软件开发模式, 涉及整个软件开发生命周期中的持续开发、持续测试、持续集成、持续部署和持续监控, 旨在较短的开发周期内开发高质量的软件。使用容器云平台可以实现“代码编写-git/svn 推送-代码自动下载-测试环境构建-单元/集成测试-新版本自动更新上线”整个过程的自动化运行, 最大限度地简化了运维成本, 同时保证线上、线下环境完全一致, 实现快速、安全和高质量的软件开发和发布[27-28]。因此, 结合轻量级虚拟化技术的 DevOps 开发模式已成为大多数组织和团队的发展方向。

3　轻量级虚拟化面临的安全挑战

轻量级虚拟化技术可以适用于多种多样的系统架构和应用场景中, 所面对的安全问题和挑战也是多种多样的。除了面临各种传统的网络安全威胁之外, 轻量级虚拟化技术还面临一些新的安全挑战, 这些挑战主要来源于以下两个方面。一方面, 传统的虚拟机拥有自己独立的内核和操作系统, 而轻量级虚拟实例则是和宿主机共享内核与操作系统, 所以宿主机内核出现漏洞等安全问题会直接影响其承载的轻量级虚拟化实例的安全性。另一方面, 目前而言, 轻量级虚拟化技术实现中的隔离机制并不完善, 存在各种跨实例的子系统共享和文件共享情况, 这使得实例间的同驻攻击变得更为简单。同时, 现有访问控制机制等保护措施的防护粒度不够, 难以针对两个容器需要读取同一个文件等系统运行场景进行防护, 导致了轻量级虚拟化技术的安全防护措施并不全面细致。

本文以被攻击目标在系统中所属的层次为依据对轻量级虚拟化系统面临的安全挑战进行了划分, 包含轻量级虚拟实例层、宿主机层和硬件层, 同时介绍了针对镜像和其他配套系统的安全威胁。图 2 汇总了各种现有的针对轻量级虚拟化架构的攻击方法。

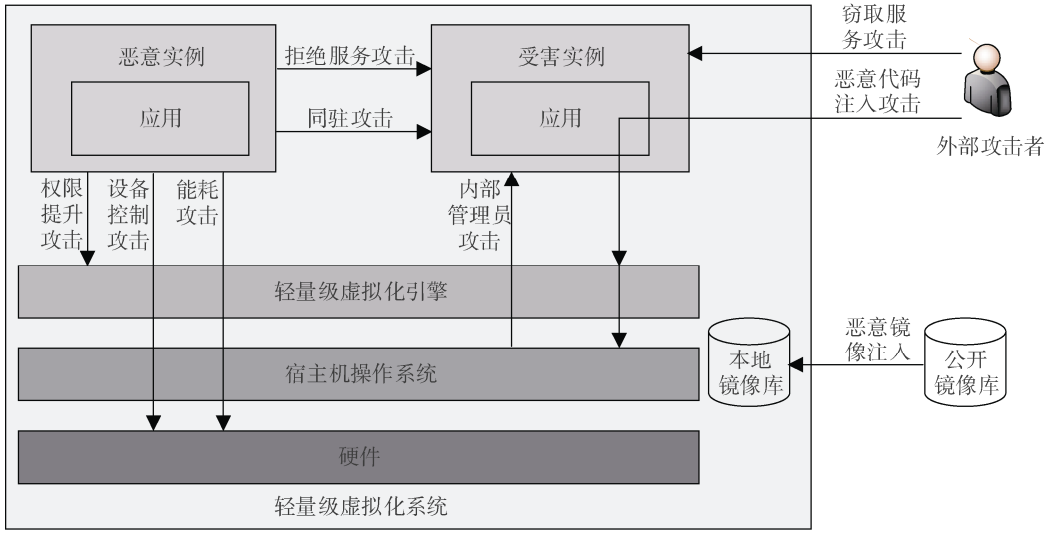

图 2　轻量级虚拟化架构面临的攻击方法

Figure 2　The attack methods used in lightweight virtualization architectures

3.1　针对轻量级虚拟实例层的攻击

轻量级虚拟实例是运行系统服务的具体节点, 也是系统中最直接受到安全威胁的层面。攻击者可以通过外部网络连接到实例内运行的应用程序, 并以此为基础进行攻击, 也可以通过实例租赁的方式直接获得公有云平台内部分实例的控制权, 进而对平台内部其他实例进行同驻攻击。其面临的安全威胁主要可以分为同驻攻击、拒绝服务攻击、窃取服务攻击等类型。

3.1.1　同驻攻击

为了支持云场景下多租户的服务模式, 容器技术设计了基于系统内核的虚拟实例间的隔离机制, 并实现了用户级别的、非特权的实例运行[14]。现有的隔离机制有效地限制了同驻实例间的相互感知和影响, 可以抵御大多数的恶意攻击。然而, 有些操作系统的子系统和实例应用的相关性较弱并且代码开发难度大, 现有的隔离机制并没有实现对于虚拟实例和宿主机、虚拟实例和虚拟实例之间这些子系统的隔离划分。攻击者可以利用这些脆弱性对在同一宿主机上的虚拟实例进行同驻攻击, 达到窃取隐私信息等攻击效果[29]。

一种方法是利用 Linux 基于内存的文件系统(Memory-based Pseudo File Systems)在虚拟化实现中的不完全隔离现象对同驻实例进行探测, 发现并窃取其敏感的数据信息。基于内存的文件系统可以通过正常的文件 I/O 操作来修改内核数据。这类系统包括 procfs,sysfs,devfs,securityfs,debugfs 等, 并广泛应用于虚拟实例和宿主机的交互之中, 但是并没有实现完全的隔离, 如图 3 所示。Gao 等人[30]设计了一种交叉验证工具对这些文件系统进行扫描, 发现了 17 个泄露同驻容器和宿主机信息的通道, 包含系统的内核、CPU、内存、运行状态等多个方面的敏感信息, 并证实了这些泄露通道在多个公有云平台上的存在。

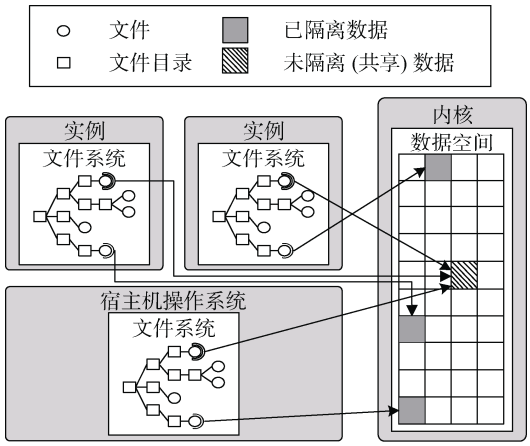

图 3　轻量级虚拟化的共享文件信息泄露

Figure 3　The information leakage of lightweight virtualization architectures

另外, 攻击者可以利用宿主机系统在缓存、内存和 CPU 调度机制中存在的漏洞实施侧信道攻击, 同样可以获得同驻实例的敏感信息[31]。Prime+ Probe[32]和 Flush+Reload[33]等攻击可以通过访问最后一级缓存(Last Level Cache, LLC)了解受害者实例的缓存使用模式。因为 LLC 是所有 CPU 核心之间共享的, 攻击者能够通过精心设计的 LLC 访问方案成功地推断出被攻击者的内部活动和秘密信息。Lipp 等人[34]提出了一种名为 Meltdown 的方法从用户空间读取内核

空间内存, 利用 CPU 乱序执行(Out-of-order Execution)机制的漏洞来读取任意的内核内存, 包括个人数据和密码。Meltdown 破坏了地址空间隔离以及半虚拟化环境所提供的安全保证, 能够在没有任何权限或特权的情况下读取云中其他进程或实例的内存, 该攻击已经在 Docker、LXC 和 OpenVZ 上得到证实。Kocher 等人[35]提出了一种利用 CPU 漏洞来进行侧信道攻击的方法, 可以绕过隔离机制读取内存和寄存器内容。在以容器技术为基础的微服务架构中, 多个程序可以同时或者分时在同一个 CPU 上执行, 由一个程序的行为引起的微服务状态变化可能会影响到其他程序。这种攻击方法利用 CPU 分支预测(Branch Prediction)和预测执行(Speculative Execution)机制的漏洞诱导 CPU 错误的执行命令序列, 进而通过侧信道获得其他容器的敏感信息。

3.1.2 拒绝服务攻击

拒绝服务攻击(Denial of Service Attack, DoS)是一种资源耗尽型攻击。传统的 DoS 攻击通过向目标实例发起大规模处理请求, 尝试耗尽系统资源, 从而导致正常服务的瘫痪。例如, 攻击者可以频繁地向容器集群漏洞发起远程登录和退出的操作以占用对应接口, 导致正常节点无法接入系统。还有的攻击方法可以绕过系统对于数据包的内容检查, 向虚拟实例发送超大内容的数据包(如 Gzip 轰炸), 导致实例内存被消耗, 达到拖慢实例运行速度甚至无法实现正常服务的攻击效果。

而在云环境中, 除了传统的攻击方法之外, 攻击者可以不直接向目标实例发起连接, 而是通过同驻检测的方法检测到目标实例所在的宿主机, 并在该宿主机上申请创建属于攻击者自己的虚拟实例。然后增加自身实例的对于共享系统资源使用量以阻断其他同驻实例对于系统资源的使用, 达到拒绝服务攻击的目的。Khalid 等人[36]介绍了一种利用网络堆栈打破系统隔离的攻击方法。具体来说, SofrIRQ 是一种用于管理延时数据包传输、指导数据包通过网络堆栈的软件中断机制。攻击者通过网络连接向属于自己的实例发送大量的数据包来加大 SofrIRQ 的执行时间和次数, 利用大量中断强制抢占其他容器 CPU 使用时间, 阻碍其他实例的正常运行。

3.1.3 窃取服务攻击

攻击者会利用实例内的应用漏洞或者恶意存放在系统镜像内的后门程序在实例内部注入木马或蠕虫病毒, 在用户不知情的情况下使用用户实例的计算资源为自己所用。近年来, 多个案例[37-38]已经证明, 黑客可以远程控制数百万个注入了蠕虫病毒的容器, 利用挖掘加密货币的程序进行“挖矿”, 严重威胁了正常用户对自身实例的控制和使用。

3.1.4 其他安全威胁

虚拟实例的运行依赖于宿主机操作系统内核, 同时宿主机控制着实例所使用的 CPU、内存、硬盘和网络设备, 所以一旦实例运行在了不受信任的宿主机上, 则可能产生严重的安全问题, 例如对实例内应用行为的探测、未授权的数据访问甚至控制实例运行等。特别是随着 CaaS 的出现, 租户可以向云服务提供商申请租借虚拟实例进行使用, 而此使用场景的前提是云服务提供商是安全可信的。如果云服务提供商存在恶意的系统管理员或者宿主机被攻击者渗透则可能对租户的虚拟实例造成严重的安全威胁。

此外, 各类应用的不正确配置也会带来安全隐患。例如数据库等应用软件的远程连接端口在缺少认证机制的情况下开放可能被攻击者利用, 窃取用户信息并对系统安全造成威胁。此外, 在有些应用场景下, 实例内运行的应用需要较高的系统权限, 这些应用也可能被攻击者利用来对系统进行探测和攻击。

3.2 针对宿主机层的攻击

宿主机承载着以 Root 权限运行的轻量级虚拟化引擎, 而虚拟化引擎是虚拟实例运行的基础, 为不同实例提供隔离环境并负责实例的创建、资源分配和运行管理等。然而由于虚拟化引擎、编排工具和操作系统漏洞的存在, 恶意用户可以通过一定的方式突破其隔离限制, 获得超过其自身的访问控制权限, 访问主机系统资源或执行恶意代码, 达到攻击或控制宿主机的目的。

3.2.1 恶意代码注入攻击

恶意代码注入攻击(Malware Injection Attack)利用虚拟化引擎和操作系统在函数执行、文件读写时的漏洞向特定代码段或文件内写入命令或代码, 以执行超出原有能力范围的操作, 进而威胁虚拟实例和宿主机的安全。

攻击者可以针对实例化的 Web 应用程序发起跨站点脚本攻击、SQL 注入攻击和远程代码执行等, 进而获得运行特定代码或程序的能力, 对系统进行破坏或者非法盗取证书信息和用户数据。破壳漏洞(CVE-2014-6271)就是通过 Apache 等第三方程序修改环境变量, 利用 Bash 在 ENV 命令实现时的输入检查漏洞在应用运行过程中执行指定的攻击代码脚本, 进而对系统进行攻击。虚拟化引擎和编排工具存在的漏洞也可能被攻击者所利用, 通过在正常的函数中插入精心设计的代码段使本地用户绕过保护机制,

执行任意命令。

3.2.2 权限提升攻击

轻量级虚拟化技术在发展过程中逐步引入了多种不同的机制来实现和完善虚拟实例和宿主机之间的权限隔离。其权限层级模型从权限等级由低到高可分为三层: 轻量级虚拟实例层、轻量级虚拟化引擎层和宿主机操作系统层。一般情况下, 轻量级虚拟实例层的用户所属的权限等级是非特权用户, 并且是经过命名空间系统访问控制系统限制的, 只能在其被赋予的权限范围内执行各种操作以及访问与自身相关的各类软硬件资源。但是攻击者依然可以利用各种系统漏洞绕过权限检查或突破权限隔离, 获得超过自身所有的各类权限进行操作执行和资源访问。

轻量级虚拟化系统中的权限提升主要分为两个层次, 一个是恶意的实例用户利用轻量级虚拟化引擎的漏洞获得部分引擎的操作权限; 另一个是攻击者完全绕过或突破引擎和操作系统的隔离限制进入宿主机, 甚至得到宿主机的 Root 权限。本小节将分别对以上两种层次的权限提升中可能存在的攻击手段和原理进行描述。

(1) 轻量级虚拟化引擎层

攻击者可以利用轻量级虚拟化引擎的漏洞绕过系统的权限检查机制来执行有利于自身的敏感操作。例如, 有些安全漏洞(例如 CVE-2017-5985)可以被攻击者所利用, 在本地主机上创建并命名网络接口界面。还有一些漏洞(例如 CVE-2019-16884)可以在 proc 系统文件内植入恶意的容器镜像。攻击者还可以进一步利用漏洞, 达到控制系统的访问控制模块的效果。例如, 攻击者利用数据卷的挂载配置错误(例如 CVE-2015-3631), 可以通过使用特制的容器镜像覆写文件系统中的文件, 设置任意的 Linux 安全模块(Linux Security Modules, LSM)策略。除此之外, 一些虚拟化引擎 API 调用的漏洞(CVE-2017-7297)还可以被攻击者用来直接禁用访问控制。

(2) 宿主机操作系统层

在轻量级虚拟化中虚拟实例和宿主机共用操作系统, 一旦攻击者突破虚拟化引擎的环境隔离限制和权限隔离限制就有可能直接接触到宿主机操作系统, 甚至得到系统的 Root 权限, 进一步攻击或控制宿主机。权限提升攻击可以通过多种类型的系统漏洞来实现, 主要包含以下几种情况。

- 轻量级虚拟化通常使用 Change Root(chroot)或 MNT Namespace 对文件系统进行隔离, 本质上是对虚拟实例访问文件系统的权限进行限制, 使之只能看到与之相关的目录及文件。但是这种方法并不完善, 攻击者可以通过各种方法绕过这种权限限制。攻击者可以在自身实例内访问一些系统文件(文件描述符、配置文件等)并对其进行修改, 插入特殊构造的语句。当虚拟化引擎或操作系统执行某些操作时会载入并运行该文件。根据构造语句的不同, 攻击者可以实现多种攻击效果, 例如遍历宿主机文件目录来获取敏感信息, 或者赋予实例用户更高的系统权限甚至逃逸到宿主机操作系统中, 完成权限提升攻击。
- 虚拟化引擎在执行 Root 用户操作和虚拟实例用户操作的状态切换过程中存在漏洞, 攻击者可以拦截具有 Root 权限的用户的连接, 并将自身替换为该用户, 达到权限提升的目的。
- 在轻量级虚拟化引擎的某些模块中存在缺少身份验证机制或访问控制强度不足的问题, 导致攻击者可以通过构造异常的字符串等方式绕过系统检查, 突破对于自身的权限限制来执行某些操作, 甚至获得操作系统的 Root 权限。
- 操作系统内核自身的漏洞如 Linux 内核函数引用错误、中断处理错误、内存竞争错误也可能被恶意容器用户利用, 达到本地用户提权的效果, 进而对宿主机安全造成威胁。
- 除此之外, 虚拟化引擎普遍存在对于容器镜像安全检查不足的情况。攻击者可以在镜像文件中插入硬连接或者恶意代码, 在镜像加载时这些内容会被触发, 导致直接以 Root 权限运行相应代码或者创建出有宿主机 Root 权限的虚拟实例。

3.3 针对硬件层的攻击

硬件设备安全是保证系统正常运行的基础, 一旦出现问题不但会影响系统管理员对于系统的正常操作, 还会危害整体信息系统的安全性。有的攻击方法(例如 CVE-2018-10892)可以绕过文件系统隔离, 访问到硬件设备的配置文件并对他们进行修改, 达到改变硬件设备配置和部分控制设备运行的目的。Gao 等人[30]提出的攻击方法可以通过收集容器的开机时间、电力消耗等看似不重要的信息来确定哪些容器的宿主机处在机房中相同的服务器机架和电源分配单元上, 并在短时间内提高这些容器的计算量以达到瞬间提高电力消耗的效果。机房电力系统检测到电力超负荷以后会触发断电保护机制导致服务器机架的强制断电。这也导致了所在机架上的服务器上承载的所有其他虚拟实例的宕机, 达到了能耗

攻击[39-40]的效果。

3.4 其他攻击

除了轻量级虚拟化系统自身面临安全威胁外, 容器镜像库、容器相关的架构组件和其他配套系统同样面临着各种安全挑战。

使用安全可信的镜像创建容器是容器安全的重要基础之一[41]。容器镜像通常存储在镜像仓库, 镜像仓库分为云端的公共仓库和本地的私有仓库两大类。公共仓库是面向全网用户的共享仓库, 主要代表是 Docker Hub 和 DaoCloud Hub。公共仓库中的镜像包括由软件厂商官方上传和第三方用户自制上传两种类型, 种类繁多但质量参差不齐。这些镜像通常没有经过安全扫描, 可能存在各种安全漏洞[42]。前文中提到的 CVE-2019-5021 就是因为官方上传的镜像中使用了硬编码的方式记录 Root 用户账户而被攻击者利用、进而实现提权攻击的。研究人员根据安全公司 Mitre 和美国国家漏洞数据库(National Vulnerability Database, NVD)提供的数据对 Docker Hub 上的镜像进行了扫描, 发现官方上传的镜像中超过 30% 的镜像易受到 Shellshock、Heartbleed 或 Poodle 等漏洞的影响, 而在由第三方上传的镜像中, 有超过 40%的镜像没有经过任何权威机构的认证[43]。此外, 虽然私有仓库主要是面向组织内部的, 难以被外部用户植入恶意镜像, 但是其中镜像的质量严重依赖于镜像编写人员的技术水平和安全意识, 并且可能同样未经过安全扫描和认证, 也会存在各种未被发现的安全隐患。当这些镜像被用户下载并使用时会严重威胁到用户系统的安全性。Kromtech 公司证实, 17 个包含带门罗币挖矿病毒的镜像已累计被下载超过 500 万次[38]。Unit 42 的研究人员也发现了一种包含名为 Graboid 的挖矿劫持蠕虫病毒的容器镜像, 已经被下载并感染了超过 2000 台 Docker 宿主机。图 4 展示了上述各类镜像在系统中的引入途径和相应实例的创建过程。

表 2　已知的轻量级虚拟化安全攻击

Table 2　Known security attacks on lightweight virtualization

攻击层次	攻击类型	攻击原理	攻击实例
轻量级虚拟实例层	同驻攻击	利用轻量级虚拟化隔离性弱的特点读取共享的系统文件, 或者利用侧信道分析的方法获取各类敏感信息	Gao et al.[30], Liu et al.[32], Yarom et al.[33], Lipp et al.[33], Kocher et al.[34]
	拒绝服务攻击	向虚拟实例发起大规模数据请求以耗尽实例拥有的资源量, 或者在同驻实例上运行大量计算或网络操作以抢占系统资源	Khalid et al.[36], CVE-2016-6595, CVE-2017-14992, CVE-2017-11468, CVE-2018-20699
	窃取服务攻击	向目标实例内植入蠕虫或木马病毒, 并隐蔽地使用被控实例的计算资源或网络资源	Cryptojacking[37-38]
宿主机层	恶意代码注入攻击	利用虚拟实例内所执行代码的特定位置写入恶意代码, 通过运行该代码段发起攻击	CVE-2014-6271, CVE-2018-6764, CVE-2019-13139, CVE-2020-7606
	权限提升攻击	利用各种系统漏洞绕过权限检查或突破权限隔离, 获得超过自身所有的各类权限	CVE-2014-(3499, 6407, 6408, 9356-9358), CVE-2015-(3214, 3290, 3627, 3631), CVE-2016-(0728, 5195, 9223, 9962), CVE-2017-(5123, 5985,7297, 18641), CVE-2018-(9862, 15514), CVE-2019-(5021, 5736,14271,15752,16884), CVE-2020-(5239, 11492, 14298,14300, 15360)
硬件层	设备控制攻击	利用系统隔离漏洞或访问控制漏洞修改设备配置文件或运行状态, 控制设备的开关和调试	CVE-2015-3630, CVE-2018-6556
	能耗攻击	在受控实例上突然运行大量计算服务, 瞬间提高相关电力控制单元的用电量, 使其电力负载超标, 导致硬件设备强制断电	Gao et al.[30], Xu et al.[39], Li et al.[40], CVE-2018-10892
镜像仓库	后门攻击	在制作镜像时加入后门代码或病毒, 当用户使用该镜像创建虚拟实例时通过后门进入实例或激活病毒进行攻击	Henriksson et al.[41], Shu et al.[42], Banyan Team[43], CVE-2019-5021

此外, 容器相关系统还面临一些其他方面的安全挑战。容器引擎和编排工具的不正确配置、未经安全防护的容器网络和各种应用管理缺陷都可能给攻击者带来可乘之机。

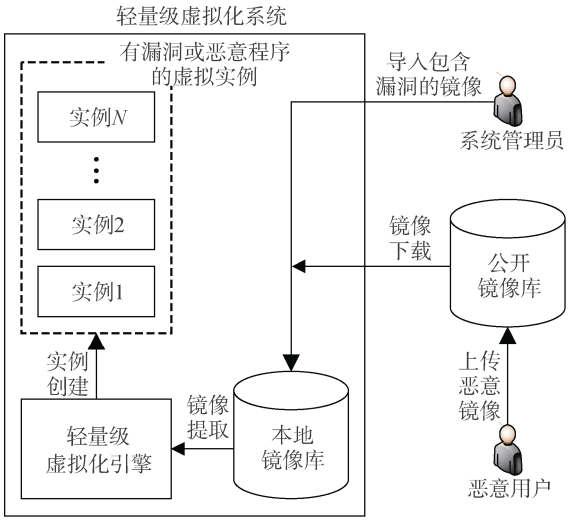

图 4　镜像安全问题的引入途径

Figure 4　The introduction of image security

4　轻量级虚拟化安全解决方案

本文以安全解决方案的实现机制在系统中所处的层次为依据对其进行了划分, 包括基于 Linux 内核机制的安全强化、基于硬件的安全强化和基于应用软件的安全强化, 同时介绍了容器编排工具、镜像系统等相关组件的安全防护方案, 并总结了这些方案的长处和弊端。

4.1　基于 Linux 内核机制的安全强化

轻量级虚拟化技术的实现在很大程度上是基于 Linux 内核的安全特性的, 其中最为关键的模块是 Namespace、Control Group 和 Capabilities, 分别用于虚拟实例间的运行环境隔离、系统资源隔离和系统权限分配。本小节主要对一些与系统隔离或系统安全相关的内核特征及以其为基础的安全强化方法进行介绍。

4.1.1　Namespace

命名空间 Namespace[44]最早在 Linux Kernel 2.4.19 版本被引入内核, 主要用于为特定进程隔离和虚拟化各类系统资源。这种机制被轻量级虚拟化技术所采纳, 实现了将一个虚拟实例内的进程和多个不同种类的 Namespace 互相映射, 使实例拥有独立的、定制化的系统资源管控。对于一个 Namespace 的修改只会影响与之对应的进程和系统资源, 不会引起整个系统范围的变化。当前的 Linux 内核主要包含六种 Namespace: UTS、NET、PID、IPC、MNT 和 USER[29]。UTS Namespace 为每个实例分配独立的主机名和域名, 因此可以将实例视为独立的节点。NET Namespace 为每个实例虚拟单独的网络设备、IP 地址、端口和路由表等。PID Namespace 负责对进程标识符(PID)进行虚拟化, 使实例内每个进程都有两个 PID: 一个在宿主机系统上, 另一个在其对应 PID 名称空间内, 一个实例中的进程只能感知到其对应 PID 名称空间中的进程。IPC Namespace 负责隔离进程间通信资源, 包括进程间信号、管道和共享内存等。MNT Namespace 用于隔离文件系统挂载点, 在不同的 MNT 名称空间中, 进程有不同的文件系统根节点和层次结构视图。USER Namespace 用来隔离用户 ID 和组 ID, 负责在实例内的 Root 用户与主机上的非特权用户之间创建映射, 以此实现虚拟实例在用户空间的非特权运行。

容器技术利用现有的 Namespace 机制隔离了大部分常用的计算机子系统, 实现了虚拟实例的可用性。但是由于利用率低和代码改编难度大等问题(如 2.2 节所述), 仍有一些子系统没有经过 Namespace 机制的隔离, 不能保障实例的安全性, 并给系统引入诸多漏洞。安全研究人员针对此问题提出了面向 Namespace 的安全隔离强化方法。Sun 等人[45]提出并实现了 Security Namespace, 放开了内核安全框架全局性和强制性的要求, 使之可以为单独的容器设定安全策略并作用于相关进程。该方法建立了各个容器安全策略间的冲突检查机制, 在容器执行操作时不仅要符合自身的安全策略, 并且不能影响其他容器安全策略的执行, 以此为容器提供独立的安全保障和自动化的安全控制。Gao 等人[30]提出了一种 Power-based Namespace, 为容器系统补充了基于能耗的命名空间。该方法通过系统监控和分析计算得到每个容器电力消耗等信息的真实使用情况并展示给容器用户, 避免宿主机相关敏感信息的泄露, 从而防止了能耗攻击等安全威胁的产生。Jian 等人[46]设计了一种面向 Namespace 的标签管理和状态检查机制, 在实例用户进程执行时对其所在的 Namespace 和记录在配置文件中的信息进行对比, 来判断用户是否有利用漏洞跳转到宿主机 Namespace 而实现容器逃逸的行为。

4.1.2　Control Group

Control Group[47](Cgroup)最早在 Linux Kernel 2.6.24 版本被引入内核, 是一种用来对系统进程进行分组化管理的内核机制, 可以隔离、限制和调整进程组(Process Groups)所使用的系统资源和服务, 如 CPU、内存、网络带宽和磁盘 I/O 等[11]。Cgroup 为这些资源各自建立一个子系统, 根据系统进程的族群关系(父子进程继承)建立树状的控制族群图并设定其资源使用量, 来实现系统资源的管理。轻量级虚拟化技术利用 Cgroup 为各个虚拟实例定量地分配各类系统资源和服务, 其目标是防止一个实例的进程占用主机上的所有可用资源而影响其余实例和进程

的正常运行, 防止 DoS 攻击等资源耗尽型安全威胁的产生。Chen 等人[48]提出了一种基于 Cgroup 的实时防御机制来抵抗 DoS 攻击。他们使用 Cgroup 机制为每一个任务分配一组 CPU 核心, 同时限制进程提高自身的优先级, 从而保护每个任务的内存等资源的使用量不被非法占用, 保证各个实例的可用性。

4.1.3 Capabilities

传统的以 UNIX 为基础的操作系统将用户分为 Root 用户和普通用户两类, Root 用户拥有所有的系统权限, 而普通用户的进程需要接受所有的权限检查。轻量级虚拟化实例和宿主机共享操作系统, 并且其运行通常需要一些普通用户没有的权限。但是如果为实例用户分配 Root 权限则会导致其可以控制整个宿主机, 违背了虚拟化应用的基本假设。

Capabilities[49]是为了解决 Root 用户权限过于集中而存在的风险所提出的权限拆分机制, Linux Kernel 从版本 2.2 开始把 Root 用户的特权划分为 37 项可以单独进行开启和禁止的 capability 权限。如表 3 所示, 轻量级虚拟化技术(例如 Docker)利用 Capabilities 机制[14], 并从功能性、可用性以及安全性多方面综合权衡, 为虚拟实例分配必要的 capability 权限[50], 实现了对 Linux 内核功能的细粒度权限控制。虚拟化实例中的进程在操作执行时, 内核会对该实例是否具有执行相对操作的 capability 权限进行检查。即使实例被攻击者利用或攻破, 也很难利用有限的权限来威胁到系统的其他模块, 从而加强系统的安全性。

表 3 Docker 默认开启的 capability 权限列表

Table 3 The list of Docker default capabilities

名称	功能概述
AUDIT_WRITE	将记录写入内核审计日志
CHOWN	对文件 uid 和 gid 进行任意更改
DAC_OVERRIDE	绕过文件的自主访问控制
FOWNER	绕过对操作的权限检查, 设置任意文件的访问控制列表
FSETID	当文件被修改时, 不清除 set-user-ID 和 set-group-ID 权限位
KILL	允许对不属于自己的进程发送信号
MKNOD	允许使用 mknod()系统调用
NET_BIND_SERVICE	将套接字绑定到小于 1024 的端口
NET_RAW	允许使用原始套接字
SETFCAP	为文件设置任意的 capability 权限
SETGID	允许改变进程的组 ID
SETPCAP	允许向其他进程转移 capability 权限以及删除其他进程的 capability 权限
SETUID	允许改变进程的用户 ID
SYS_CHROOT	允许使用 chroot()以及使用 setns 更改挂载的 namespace

4.1.4 其他内核机制的应用

除了以上提到的内核机制以外, 有些轻量级虚拟化的安全解决方案方法还采用了其他的内核机制来强化系统隔离和权限分配, 例如 SELinux[51]、AppArmor[52]、Seccomp[53]、TOMOYO[54]等。Mp 等人[55]研究了 SELinux、AppArmor 和 TOMOYO 等内核机制对 Docker 容器的影响, 并从概念上证明了它们在降低因系统漏洞和恶意代码等造成的安全威胁的有效性。

Security Enhanced Linux(SELinux)是 Linux 内核 2.6 版本开始提供的一种强制访问控制系统, 属于 Linux 安全框架[56](Linux Security Module, LSM)。通过 SELinux, 管理员可以设定安全策略, 规定所有进程(称为主体)对系统的其他部分(如文件、设备、端口和其他进程等, 称为客体)进行操作的权限, 进程只能访问其被允许访问的目标对象[57]。目前主流的轻量级虚拟化实现方法如 LXC、LXD、Docker 和 Rocket 都采用了此系统。Bacis 等人[58]还设计了一种方案, 将 SELinux 策略和容器镜像进行了绑定, 限制每个镜像建立的容器可访问的系统目标对象, 可以防止恶意容器利用系统漏洞进行越权访问和操作。

AppArmor 提供了一种基于路径名的访问控制, 在版本 2.6.36 被整合进 Linux 内核, 主要作用是设置可执行程序的访问控制权限, 可以限制程序读/写某个目录/文件或打开/读/写网络端口等[52]。AppArmor 通过配置文件来指定一个应用程序的相关权限, 与 SELinux 相比更安全, 访问控制粒度更小更灵活, 但存在着配置更复杂的问题。AppArmor 得到了广泛支持并应用于 Docker 和 LXC。Andreou 等人[59]提出了一种基于 AppArmor 的自动化安全防护系统, 在创建容器时以镜像相关参数以及其包含应用程序的功能为依据创建一组访问规则集合, 以限制恶意镜像带来的威胁。

安全计算模式 Secure Computing Mode(Seccomp)是 Linux 内核 2.6.10 版本引入的机制, 通过黑白名单的方式定义应用程序可以使用的系统调用, 广泛应用于多种容器和沙盒系统中[53,60]。Lei 等人[61]分析了容器在启动、运行等阶段所需系统调用的不同并扩展了 Seccomp, 使之可以在容器运行的不同阶段自动、动态地更新其可用的系统调用集合, 防止攻击者利用低频的系统调用进行攻击。Wan 等人[62]在容器正式运行前将其放入沙箱环境内进行测试, 并获取其测试期间的系统调用和配置文件创建行为。如果发现这些行为是正常的, 那么将在该容器正式运行时使用 Seccomp 只赋予它曾在测试时试用过的系统

调用并拒绝其他任何系统调用, 从而防止异常行为的产生。

TOMOYO 是一种强制访问控制机制, 旨在保护整个系统免受攻击者利用应用程序漏洞进行的攻击, 通常作为系统限制工具和监控工具使用[54]。TOMOYO 不需要用户提供现成的策略文件, 而是通过记录测试环境中所有进程的行为, 允许每个进程声明其完成任务所需的操作和资源并交于系统管理员审核, 然后在生成环境中对进程进行监控, 只允许申请过的行为在符合管理员要求的前提下运行。通过制定最低特权的策略, TOMOYO 可以防止应用程序受到威胁时对于系统造成的损害。

4.2 基于硬件的安全强化

与基于软件的防御方法相比, 基于硬件的防御方法更为稳定和安全, 所以有些学者提出结合硬件的安全设备和模块来进行安全防护, 目前讨论较为广泛的技术大多基于可信平台模块[63](Trusted Platform Module, TPM)或 Intel 的软件保护扩展模块[64] (Software Guard Extensions, SGX)来实现。

可信平台模块 TPM 是一种植入在计算机内部为计算机提供可信根的芯片, 是可信计算中用作加密处理器的硬件。TPM 为认证、数据封装, 安全启动和算法加速提供硬件支持[63,65]。轻量级虚拟化技术同样可以使用 TPM 对不同的虚拟实例提供安全保护。但是由于一台宿主机只有一个 TPM 芯片而同时承载多个虚拟实例, 系统需要对 TPM 进行改造以适应该使用场景, 因此诞生了虚拟 TPM[66] (vTPM)技术, 如图 5 所示。Perez 等人[66]对硬件 TPM 芯片进行了虚拟化, 使之可以对本计算机上无限数量的虚拟实例提供可信计算能力。Wan 等人[67]测试了使用 vTPM 的可信云计算, 并分析了现有的 vTPM 解决方案。Hosseinzadeh 等人[68]研究了 vTPM 的功能实现以及在传统虚拟化中的集成, 提出了第一个用于容器的 vTPM 实现方法, 并设计了两种将 vTPM 集成到基于容器的轻量级虚拟化模型中的架构解决方案, 当容器创建时为其创建并绑定一个 vTPM。还有一些研究[69-72]针对云计算场景下虚拟实例频繁迁移的问题进行了研究, 通过设计专门的安全协议等方法保障实例在迁移前后信任关系的恒定, 进一步支持了 TPM 相关技术的广泛应用。

2015 年, Intel 推出了与其 CPU 适配的 SGX 芯片, 增加了对安全域的支持, 在安全域内的代码无法被其以外的代码所访问, 因此可以保护应用程序代码和数据不被其他软件(包括具有较高特权的软件, 如虚拟化引擎和内核)访问[64]。这一特性非常适合用来对虚拟实例进行安全保护, 可以将每个实例相关的进程放在独立的安全域内执行, 保护其机密性和完整性。Arnautov 等人[73]提出了一种结合 SGX 芯片的安全解决方案来保护容器进程免受外部攻击, 设计了一个小型的可信计算基和安全的 C 语言标准库接口, 支持异步的系统调用和用户级的线程提供保护。Hunt 等人[74]提出了一种名为 Ryoan 的分布式沙箱, 可以在不受信任的计算平台上利用 SGX 提供的安全域建立独立的沙箱, 保护应用程序的安全性, 防止用户数据泄露或被他人篡改等情况的发生。Vaucher 等人[75]为运行在异构服务器集群上的容器设计了一种新的协调器, 该协调器利用 SGX 保障用户可以在无需信任提供商的情况下将他们的软件数据部署到云中运行。该研究扩展了 SGX Linux 驱动程序, 并可以配合 Kubernetes 在私有云上运行。

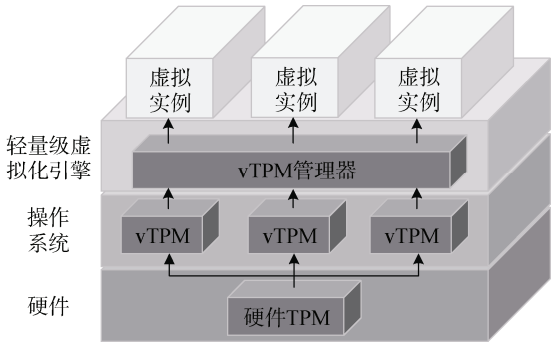

图 5 一种 vTPM 的实现实例

Figure 5 An example of vTPM implementation

4.3 基于应用软件的安全强化

为了解决轻量级虚拟化存在的诸多安全问题, 一些研究者设计了基于软件应用层的安全方案, 主要分为两种类型: 一种是对虚拟实例的权限和行为等进行监控, 针对异常情况进行告警、限制和处理。Khalid 等人[36]通过对 CPU 进行细粒度的监控和调度实现对于网络资源的强隔离。当实例绕过现有隔离机制占用其余实例网络时长时, 限制其网络通信行为, 保证其余实例网络资源的正常使用, 防止拒绝服务攻击的产生。Liu 等人[76]提出了一种针对实例间内存地址空间安全隔离的形式化定义方法和一套相对应的策略规则集, 以此控制实例用户对于内存地址空间的安全读写。Abed 等人[77]设计了一种基于宿主机的实时入侵检测系统, 利用 Linux strace 工具来检测实例内进程和宿主机之间的系统调用信息并对系统调用序列进行异常检测, 进而阻止实例用户可能存在的攻击行为。

另一种是利用主动防御的思想, 对系统的各类

配置或状态进行动态的变化, 以达到迷惑攻击者、加大攻击难度的效果。Ma 等人[78]设计实现了一个容器集群管理框架 Libra, 通过对系统网络链路的状态和各个容器的带宽需求等数据对网络共享问题进行理论分析, 从而将各个容器分配和放置到相对优化的位置, 为容器应用提供了带宽保障并减少了针对网络的 DoS 攻击的可能。主动防御的方法还可以用来对虚拟实例在数据中心宿主机上的位置进行合理

表 4 现有轻量级虚拟化防御机制对比

Table 4 Comparison of defense mechanisms on lightweight virtualization

机制所属层次	防御机制	是否需要修改内核	是否需要硬件支持	防御原理	可防攻击	实现方案	不足
Linux 内核层	Namespace	是	否	系统环境隔离	同驻攻击 窃取服务攻击 恶意代码注入 权限提升攻击 远程设备控制 能耗攻击	Babar et al.[14], Gao et al.[30], Sun et al.[45], Jian et al.[46]	难以区分虚拟实例和宿主机边界, 实现复杂
	Control Group	是	否	系统资源隔离	拒绝服务攻击 窃取服务攻击 能耗攻击	Babar et al.[14]	难以区分用户进程和内核进程依赖关系, 实现复杂
	Capabilities	是	否	系统权限隔离	同驻攻击 恶意代码注入 权限提升攻击	Babar et al.[14], Docker Default Capabilities[50]	虚拟实例中应用功能复杂多样, 难以自动化配置
	访问控制强化	是	否	系统权限细粒度管理分配	同驻攻击 恶意代码注入 权限提升攻击	Bacis et al.[58], Andreou et al.[59], Lei et al.[61], Wan et al.[62]	虚拟实例中应用功能复杂多样, 难以自动化配置
硬件层	TPM	是	是	可信认证, 可信计算	同驻攻击 恶意代码注入	Perez et al.[66], Wan et al.[67], Hosseinzadeh et al.[68]	需要硬件支持, 难扩展推广
	SGX	是	是	提供安全域, 防止代码篡改和数据泄露	同驻攻击 恶意代码注入	Arnautov et al.[73], Huntet al.[74], Vaucher et al.[75]	仅支持 Intel 部分型号的 CPU, 难扩展推广
应用层	异常检测	不定	否	监控系统行为, 发现、阻断攻击	同驻攻击 窃取服务攻击 恶意代码注入 权限提升攻击	Khalid et al.[36], Liu et al.[76], Abed et al.[77]	难以发现新的攻击方法
	主动防御	不定	否	动态变化系统状态, 阻断扫描和攻击	同驻攻击 拒绝服务攻击 窃取服务攻击 恶意代码注入 远程设备控制	Ma et al.[78], Han et al.[79], Moon et al.[80], Miao et al.[81], Kong et al.[82], Bohare et al.[83], Huang et al.[84], Azab et al.[85,86]	配置复杂, 易带来额外的系统负载

放置和动态迁移, 以此来破坏不同用户实例间同驻关系, 从而阻断攻击者的攻击路径、解决轻量级虚拟化隔离性较弱而可能产生的侧信道攻击和敏感信息泄露等问题。Han 等人[79]提出了一种针对云环境中同驻攻击的度量方法并比较了常见实例放置策略下攻击者实现同驻的难度, 设计了一种新的实例放置策略并在 Openstack 平台上实现和测试。与此类似, Moon 等人[80]设计了一种用于减少系统总体数据泄露量的贪心算法, Miao 等人[81]设计了一种针对同驻率、系统性能和资源约束等条件的多目标优化自适应算法, 用于降低系统整体的同驻率, 最小化侧信道攻击带来的信息泄露问题。Kong 等人[82]通过一种针对容器云中的同驻攻击模型来测量每个用户和其他用户的同驻情况, 并判断各个用户之间可能因同驻攻击而带来的威胁性。当发现某个用户对其他用户的威胁超过定义的安全指标时, 即对该用户的容器根据提出的基于遗传算法的策略进行迁移调度, 达到破坏和其他用户的同驻环境、阻断同驻攻击的效果。为达到相同目标, Bohare[83]通过对容器所在物理服务器 syslog 日志记录的信息进行检测来发现系统的异常行为, 并以此为依据触发容器的迁移。除此之外, Huang 等人[84]采取了定时将容器迁移到与容器当

前所在宿主机差异性最大的其他宿主机的策略, Azab 等人[85-86]利用了随机化的算法对容器在云内的宿主机上的位置进行动态迁移, 以实现实时、高效、可伸缩的移动目标防御, 并通过理论分析证明了方案对于增加攻击者定位目标、攻击目标的难度的有效性。

4.4 其他安全防护

除了对于轻量级虚拟化架构的各个层面进行安全强化外, 其整个技术生态系统其余方面的安全也同样值得关注和研究, 例如对于轻量级虚拟化编排系统和镜像库的安全防护。

4.4.1 编排工具安全

轻量级虚拟化编排工具管理着承载各类服务的虚拟实例集群、支撑了其核心业务, 所以对编排工具的安全保护和合理配置同样对系统安全有着重要的意义。Kubernetes 社区提供了一种名为 Pod Security Policy[87]的机制对于集群资源进行安全规范和管理, 从而控制实例用户的权限分配、数据卷使用和文件系统选择等。同时, Kubernetes 还对集群各个节点和 API 提供了访问认证和访问授权等机制, 保证服务间访问的安全性。Swarm 内置了 Docker Secret[88]指令来保护账号密码、安全传输层协议(Transport Layer Security, TLS)证书和密钥、SSH 密钥和各类敏感信息, 同时提供了有加密的分布式集群存储、安全集群接入令牌和一套简化数字证书管理的公钥基础设施(Public Key Infrastructure, PKI)。合理使用和配置这些安全强化机制可以在一定程度上提高系统在实际使用中的安全性。

4.4.2 镜像安全

镜像是虚拟实例运行的基础, 镜像的安全在整个安全生态中占据着重要位置, 对于其安全的保护主要体现在镜像加密认证和镜像内容扫描两个方面。图 6 介绍了各类镜像安全检测机制在系统中的运行流程。

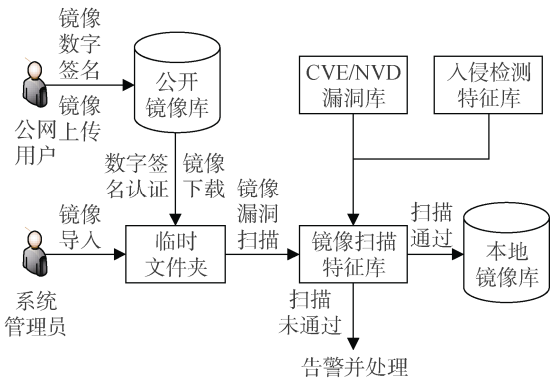

图 6 镜像安全检测机制

Figure 6 The image security detection mechanism

容器厂商 Docker 在自己的容器生态系统中设计了 Content Trust 机制, 保障在公有镜像仓库 Docker Hub 上下载的容器镜像的合法性[89-90]。该机制在用户上传镜像到镜像仓库时使用该用户的密钥对镜像进行加密, 之后其他用户在镜像仓库下载该镜像即可对签名进行检查, 保障镜像的来源可信并且没有被篡改。开源项目 Notary 同样使用数字签名来保证服务器和客户端之间的可信连接和交互, 也被广泛应用于构建镜像仓库, 来支持对容器镜像签名、镜像源认证和镜像完整性检查等安全需求。Giannakopoulos 等人[91]优化了宿主机间的镜像传输方法, 针对容器镜像分层的特点在镜像传输时对整体镜像进行分割, 只传递上层的包含隐私信息的镜像层并对镜像层使用 AES-256 算法进行加密, 既减少了数据传输量又保证了镜像数据的隐私性。Xu 等人[92-93]提出了基于区块链的 Docker 信任解决方案, 为容器镜像提供签名验证服务的同时利用区块链去中心化的特点降低了系统被 DoS 攻击的风险, 并在多个数据中心上对其进行了性能评估, 证明了方案的效果和扩展性。

另外, 为了防止系统漏洞和恶意代码引入镜像进而影响系统的安全, 从公共镜像仓库下载镜像时需要对镜像进行安全检查, 防止系统内启动的实例自身存在安全问题。目前已经出现多个可以应用于镜像漏洞扫描的开源项目, 例如 Clair[94]、Anchore[95]、Dockerscan[96]和 Trivy[97]等, 这些方案通过对镜像进行分层扫描、提取其中包含各类软件的版本号等特征信息并与 CVE、NVD 等权威漏洞数据库对比等方法, 发现其中包含的漏洞并进行报警。

Docker 官方的 Docker Security Scanning[98]服务在此基础上增加了对于每一镜像层的二进制级别分析, 并且把二进制下隐含的标记与系统显示的版本做匹配, 不仅能根据软件版本号等信息发现有漏洞的组件, 还能正确识别已经打过补丁或者被恶意修改的组件, 提高了扫描检测的准确性。除此之外, 还有一些方案(例如 Dockerscan)针对镜像中的环境变量、操作命令以及端口开放信息等进行扫描, 并根据实际应用场景来判断其是否存在安全隐患。

5 未来研究展望

随着影响力的不断扩大, 轻量级虚拟化已经融入计算机和网络的各个方面, 特别是在云原生等应用场景中起到了至关重要的作用, 因此其安全问题也受到了越来越多的关注[9-12,99]。轻量级虚拟化技术面临的安全问题不仅仅是如何应对具体实现方案中可能存在的漏洞, 还包括如何在其整个运行生命周

期中安全地进行使用和维护。结合前文所述轻量级虚拟化技术的发展趋势及当前相关安全方面的最新研究进展, 可以发现现有轻量级虚拟化保护方案仍然存在一些不足, 未来的科研工作可以重点关注以下几个方面。

强化虚拟隔离, 防止各类同驻攻击的产生。多租户共享宿主机是云场景下经典的服务模式之一, 因此必须强化虚拟实例和宿主机以及其他同驻实例间的隔离机制。一方面需要关注宿主机硬件资源的隔离, 防止恶意用户抢占超过其配额的资源量, 保障各类系统功能和应用服务的正常运行。另一方面需要关注宿主机内核和文件系统等操作系统层面的隔离, 防止攻击者对于系统内各类信息的窃取, 阻断因隔离性不足而带来的攻击威胁。

保障镜像安全, 支持镜像溯源, 防止漏洞引入。虚拟实例大多数是基于相应镜像创建的, 所以镜像是轻量级虚拟化实现方案运行的基础。应当建立统一的容器镜像签名机制, 在镜像上传和下载时进行数字签名和签名检测, 保证镜像来源的可验证性。同时需要健全镜像的安全扫描机制, 检测其是否包含系统漏洞或恶意代码等安全隐患, 及时阻断不安全镜像的使用, 保障系统的安全性。

建立评估标准, 全面覆盖生态系统安全检查。以轻量级虚拟化技术为基础的云平台和微服务架构需要一系列软硬件产品的支持, 并依赖使用者设定的机制和策略来运行。虽然现在已经有研究开始关注其生态系统和生命周期的整体安全运行, 但是大多是针对特定系统的安全检测和评估, 适用性较差, 无法进行大范围的推广。因此未来互联网厂商和研究人员需要合作建立相应的系统建设标准和安全检查标准, 对系统部署、通信协议和评估方法的标准化进行进一步的研究, 希望可以通过自动化的解决方案来实现安全检查和防护。

关注新兴领域, 支持新场景下的安全运行。轻量级虚拟化技术具有灵活、低功耗的特点, 所以被物联网和边缘计算等新兴产业所关注, 进入到了更多的实际使用场景之中。而在这些场景中使用的设备通常不同于传统的服务器和个人计算机, 所以在这些新场景中的轻量级虚拟化技术的安全问题也是未来重点的研究方向之一。

同时, 现有的研究工作还面临着一些困难和问题。轻量级虚拟化技术和传统的虚拟化技术在使用过程中的显著区别源于其灵活、高效的轻量级特点。以轻量级虚拟化技术为基础的系统架构的组件更新频率可能高达每天上百次, 极大地提升了系统的动态性和复杂性, 这给系统带来了更多不可预测的安全问题, 也更加难以进行针对性的安全防御。另一方面, 现有的针对轻量级虚拟化技术的安全解决方案大多是将原有对于主机的或者传统虚拟化技术的安全技术进行改造、优化和拓展, 然而在一些场景下并不完全适应轻量级虚拟化的应用模式。如何针对轻量级虚拟化相关的云原生、微服务等场景设计部署适应度更高的安全系统和防御策略也是当前面临的一个关键问题。

6 结论

轻量级虚拟化技术以其灵活、高效、易配置的特点在互联网产业中迅速占领了一席之地。云计算环境中的多租户、共享资源池等特点也扩大了轻量级虚拟化技术隔离性较弱等薄弱点带来的安全隐患, 引起了人们对其安全问题的广泛关注。本文重点介绍了轻量级虚拟化技术中各类已知的安全挑战, 然后对现有防御机制的技术路线、实现方法和不足之处进行了综述。随着轻量级虚拟化技术在实际场景中的应用普及, 其面临的安全问题可能会更加复杂和多样, 针对相关问题不断提出新的安全防御技术和解决方案仍是未来研究的重点。

致 谢 在此向对本文工作提出指导的各位老师和帮助收集整理材料的各位同学表示感谢, 以及对提出建议的评审专家表示感谢。

参考文献

[1] Soltesz S, Pötzl H, Fiuczynski M E, et al. Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors[C]. *The 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 2007: 275-287.

[2] Kaur K, Dhand T, Kumar N, et al. Container-As-a-Service at the Edge: Trade-off between Energy Efficiency and Service Availability at Fog Nano Data Centers[J]. *IEEE Wireless Communications*, 2017, 24(3): 48-56.

[3] Tay Y C, Gaurav K, Karkun P. A Performance Comparison of Containers and Virtual Machines in Workload Migration Context[C]. *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops*, 2017: 61-66.

[4] Kozhirbayev Z, Sinnott R O. A Performance Comparison of Container-Based Technologies for the Cloud[J]. *Future Generation Computer Systems*, 2017, 68: 175-182.

[5] Wang B, Song Y, Cui X, et al. Performance Comparison between Hypervisor- and Container-Based Virtualizations for Cloud Users[C]. *2017 4th International Conference on Systems and Informatics*, 2018: 684-689.

[6] Thönes J. Microservices[C]. *IEEE Software*, 2015: 116.

[7] Dragoni N, Lanese I, Larsen S T, et al. Microservices: How to make your application scale[C]. *Proceedings of the 11th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 2017: 95-104.

[8] Combe T, Martin A, Di Pietro R. To Docker or not to Docker: A Security Perspective[J]. *IEEE Cloud Computing*, 2016, 3(5): 54-62.

[9] Lin X, Lei L G, Wang Y W, et al. A Measurement Study on Linux Container Security: Attacks and Countermeasures[C]. *The 34th Annual Computer Security Applications Conference*, 2018: 418-429.

[10] Manu A R, Patel J K, Akhtar S, et al. Docker Container Security via Heuristics-Based Multilateral Security-Conceptual and Pragmatic Study[C]. *2016 International Conference on Circuit*, *Power and Computing Technologies*, 2016: 1-14.

[11] Manu A R, Patel J K, Akhtar S, et al. A Study, Analysis and Deep Dive on Cloud PAAS Security in Terms of Docker Container Security[C]. *2016 International Conference on Circuit*, *Power and Computing Technologies*, 2016: 1-13.

[12] Chandramouli R. Security assurance requirements for Linux application container deployments. Technical Report. NISTIR 2017-8176, USA National Institute of Standards and Technology.

[13] Yu D J, Jin Y K, Zhang Y Q, et al. A Survey on Security Issues in Services Communication of Microservices-Enabled Fog Applications[J]. *Concurrency and Computation*: *Practice and Experience*, 2019, 31(22): e4436.

[14] Ali Babar M, Ramsey B. Understanding Container Isolation Mechanisms for Building Security-Sensitive Private Cloud. Technical Report, CREST 2017, University of Adelaide.

[15] Felter W, Ferreira A, Rajamony R, et al. An Updated Performance Comparison of Virtual Machines and Linux Containers[C]. *2015 IEEE International Symposium on Performance Analysis of Systems and Software*, 2015: 171-172.

[16] Morabito R, Kjällman J, Komu M. Hypervisors Vs. Lightweight Virtualization: A Performance Comparison[C]. *2015 IEEE International Conference on Cloud Engineering*, 2015: 386-393.

[17] Docker. https://www.docker.com. Dec. 2020.

[18] Kubernetes. http://kubernetes.io. Dec. 2020.

[19] Dragoni N, Giallorenzo S, Lafuente A L, et al. Microservices: Yesterday, Today, and tomorrow[M]. Present and Ulterior Software Engineering. Cham: Springer, 2017: 195-216.

[20] Celesti A, Mulfari D, Fazio M, et al. Exploring Container Virtualization in IoT Clouds[C]. *2016 IEEE International Conference on Smart Computing*, 2016: 1-6.

[21] Khazaei H, Bannazadeh H, Leon-Garcia A. SAVI-IoT: A Self-Managing Containerized IoT Platform[C]. *2017 IEEE 5th International Conference on Future Internet of Things and Cloud*, 2017: 227-234.

[22] Morabito R, Petrolo R, Loscrì V, et al. Lightweight Virtualization as Enabling Technology for Future Smart Cars[C]. *2017 IFIP/IEEE Symposium on Integrated Network and Service Management*, 2017: 1238-1245.

[23] Morabito R, Cozzolino V, Ding A Y, et al. Consolidate IoT Edge Computing with Lightweight Virtualization[J]. *IEEE Network*, 2018, 32(1): 102-111.

[24] Morabito R. Virtualization on Internet of Things Edge Devices with Container Technologies: A Performance Evaluation[J]. *IEEE Access*, 2017, 5: 8835-8850.

[25] Morabito R, Farris I, Iera A, et al. Evaluating Performance of Containerized IoT Services for Clustered Devices at the Network Edge[J]. *IEEE Internet of Things Journal*, 2017, 4(4): 1019-1030.

[26] IBM-container-service.https://github.com/IBM-Blockchain-Archive/ibm-container-service. Dec. 2020.

[27] Ebert C, Gallardo G, Hernantes J, et al. DevOps[J]. *IEEE Software*, 2016, 33(3): 94-100.

[28] Balalaie A, Heydarnoori A, Jamshidi P. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture[J]. *IEEE Software*, 2016, 33(3): 42-52.

[29] Gao X, Steenkamer B, Gu Z S, et al. A Study on the Security Implications of Information Leakages in Container Clouds[J]. *IEEE Transactions on Dependable and Secure Computing*, 2021, 18(1): 174-191.

[30] Gao X, Gu Z S, Kayaalp M, et al. ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds[C]. *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2017: 237-248.

[31] Azar Y, Kamara S, Menache I, et al. Co-Location-Resistant Clouds[C]. *The 6th edition of the ACM Workshop on Cloud Computing Security*, 2014: 9-20.

[32] Liu F F, Yarom Y, Ge Q, et al. Last-Level Cache Side-Channel Attacks are Practical[C]. *2015 IEEE Symposium on Security and Privacy*, 2015: 605-622.

[33] Yarom Y, Falkner K. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack[C]. *The 23rd USENIX conference on Security Symposium*, 2014: 719-732.

[34] Lipp M, Schwarz M, Gruss D, et al. Meltdown: Reading Kernel memory from user space[C]. *in Proceedings of the 27th USENIX Security Symposium*, 2018: 973-990.

[35] Kocher P, Horn J, Fogh A, et al. Spectre Attacks: Exploiting Speculative Execution[C]. *2019 IEEE Symposium on Security and Privacy*, 2019: 1-19.

[36] Khalid J, Rozner E, Felter W, et al. Iron: Isolating Network-Based CPU in Container Environments[C]. *The 15th USENIX Conference on Networked Systems Design and Implementation*, 2018: 313-328.

[37] Attackers Cryptojacking Docker Images to Mine for Monero. https://unit42.paloaltonetworks.com/cryptojacking-docker-images-for-mining-monero/. Dec. 2020.

[38] Cryptojacking Invades Cloud. How Modern Containerization Trend Is Exploited by Attackers. https://mackeeper.com/blog/post/cryptojacking-invades-cloud-how-modern-containerization-trend-is-exploited-by-attackers. Dec. 2020.

[39] Xu Z, Wang H N, Xu Z C, et al. Power Attack: An Increasing Threat to Data Centers[C]. *Proceedings 2014 Network and Distributed System Security Symposium*, 2014: 1-15.

[40] Li C, Wang Z H, Hou X F, et al. Power Attack Defense: Securing Battery-Backed Data Centers[C]. *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture*, 2016: 493-505.

[41] Oscar H, Michael F. Static vulnerability analysis of docker images. Technical Report. Henriksson2017StaticVA 2017. Blekinge Institute of Technology.

[42] Shu R, Gu X H, Enck W. A Study of Security Vulnerabilities on Docker Hub[C]. *The Seventh ACM on Conference on Data and Application Security and Privacy*, 2017: 269-280.

[43] Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities. https://blog.banyansecurity.io/blog/over-30-of-official-images-in-docker-hub-contain-high-priority-security-vulnerabilities. Dec. 2020

[44] Linux Man. Namespaces-overview of Linux namespaces. http://man7.org/linux/man-pages/man7/namespaces.7.html. Dec, 2020

[45] Sun Y Q, Safford D, Zohar M, et al. Security Namespace: Making Linux Security Frameworks Available to Containers[C]. *The 27th USENIX Conference on Security Symposium*, 2018: 1423-1439.

[46] Jian Z Q, Chen L. A Defense Method Against Docker Escape Attack[C]. *The 2017 International Conference on Cryptography, Security and Privacy*, 2017: 142-146.

[47] Cgroup_namespaces-overview of Linux cgroup namespaces. http://www.man7.org/linux/man-pages/man7/cgroup_namespaces.7.html. Dec. 2020.

[48] Chen J Y, Feng Z W, Wen J Y, et al. A Container-Based DoS Attack-Resilient Control Framework for Real-Time UAV Systems[C]. *2019 Design, Automation & Test in Europe Conference & Exhibition*, 2019: 1222-1227.

[49] Capabilities. http://man7.org/linux/man-pages/man7/capabilities.7.html. Dec. 2020

[50] Docker Default Capabilities. https://github.com/moby/moby/blob/master/oci/defaults.go. Dec. 2020.

[51] What is SELinux?. https://www.redhat.com/en/topics/linux/what-is-selinux. Dec. 2020.

[52] AppArmor. https://www.apparmor.net. Dec. 2020.

[53] Seccomp. https://man7.org/linux/man-pages/man2/seccomp.2.html. Dec. 2020.

[54] TOMOYO. https://tomoyo.osdn.jp. Dec. 2020.

[55] Mp A R, Kumar A, Pai S J, et al. Enhancing Security of Docker Using Linux Hardening Techniques[C]. *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology*, 2017: 94-99.

[56] Linux Security Module Usage. https://www.kernel.org/doc/html/latest/admin-guide/LSM/index.html. Dec. 2020.

[57] SELinux Project. https://github.com/SELinuxProject. Dec. 2020.

[58] Bacis E, Mutti S, Capelli S, et al. DockerPolicyModules: Mandatory Access Control for Docker Containers[C]. *2015 IEEE Conference on Communications and Network Security*, 2015: 749-750.

[59] Loukidis-Andreou F, Giannakopoulos I, Doka K, et al. Docker-Sec: A Fully Automated Container Security Enhancement Mechanism[C]. *2018 IEEE 38th International Conference on Distributed Computing Systems*, 2018: 1561-1564.

[60] Seccomp security profiles for Docker. https://docs.docker.com/ engine/security/seccomp/. Dec. 2020.

[61] Lei L, Sun J, Sun K, et al. SPEAKER: Split-Phase Execution of Application Containers[C]. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2017: 230-251.

[62] Wan Z Y, Lo D, Xia X, et al. Mining Sandboxes for Linux Containers[C]. *2017 IEEE International Conference on Software Testing, Verification and Validation*, 2017: 92-102.

[63] Morris T. Trusted Platform Module[M]. Encyclopedia of Cryptography and Security. Boston, MA: Springer, 2011: 1332-1335.

[64] Intel Software Guard Extensions. https://software.intel.com/ content/www/us/en/develop/topics/software-guard-extensions.html. Dec. 2020.

[65] Felten E W. Understanding Trusted Computing: Will Its Benefits Outweigh Its Drawbacks?[J]. *IEEE Security & Privacy*, 2003, 1(3): 60-62.

[66] Berger S, Cáceres R, Goldman K A, et al. VTPM: virtualizing the trusted platform module[C]. *In Proceedings of the 15th conference on USENIX Security Symposium*, 2006: 305-320.

[67] Wan X, Xiao Z T, Ren Y. Building Trust into Cloud Computing Using Virtualization of TPM[C]. *2012 Fourth International Conference on Multimedia Information Networking and Security*, 2013: 59-63.

[68] Hosseinzadeh S, Laurén S, Leppänen V. Security in Container-Based Virtualization through VTPM[C]. *The 9th International Conference on Utility and Cloud Computing*, 2016: 214-219.

[69] Danev B, Masti R J, Karame G O, et al. Enabling Secure VM-VTPM Migration in Private Clouds[C]. *The 27th Annual Computer Security Applications Conference*, 2011: 187-196.

[70] Liang X L, Jiang R, Kong H F. Secure and Reliable VM-VTPM Migration in Private Cloud[C]. *2013 2nd International Symposium on Instrumentation and Measurement, Sensor Network and Automation*, 2014: 510-514.

[71] Zhou H, Wang J A, Zhang H G. A Trusted VM-VTPM Live Migration Protocol in Clouds[C]. *The The 1st International Workshop on Cloud Computing and Information Security*, 2013: 299-302.

[72] Wan X, Zhang X F, Chen L, et al. An Improved VTPM Migration Protocol Based Trusted Channel[C]. *2012 International Conference on Systems and Informatics*, 2012: 870-875.

[73] Arnautov S, Trach B, Gregor F, et al. SCONE: Secure Linux Containers with Intel SGX[C]. *The 12th USENIX conference on Operating Systems Design and Implementation*, 2016: 689-703.

[74] Hunt T, Zhu Z T, Xu Y Z, et al. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data[C]. *The 12th USENIX conference on Operating Systems Design and Implementation*, 2016: 533-549.

[75] Vaucher S, Pires R, Felber P, et al. SGX-Aware Container Orchestration for Heterogeneous Clusters[C]. *2018 IEEE 38th International Conference on Distributed Computing Systems*, 2018: 730-741.

[76] Liu Q A, Li X Y, Du J A, et al. Security Isolation Strategy Mechanism for Lightweight Virtualization Environment[J]. *ITM Web of Conferences*, 2017, 11: 01001.

[77] Abed A S, Clancy C, Levy D S. Intrusion Detection System for Applications Using Linux Containers[C]. *International Workshop on Security and Trust Management. Springer International Publishing*, 2015: 123-135.

[78] Ma S Y, Jiang J J, Li B, et al. Maximizing Container-Based Net-

work Isolation in Parallel Computing Clusters[C]. *2016 IEEE 24th International Conference on Network Protocols*, 2016: 1-10.

[79] Han Y, Chan J, Alpcan T, et al. Using Virtual Machine Allocation Policies to Defend Against Co-Resident Attacks in Cloud Computing[J]. *IEEE Transactions on Dependable and Secure Computing*, 2017, 14(1): 95-108.

[80] Moon S J, Sekar V, Reiter M K. Nomad: Mitigating Arbitrary Cloud Side Channels via Provider-Assisted Migration[C]. *The 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015: 1595-1606.

[81] Miao F B, Wang L M, Wu Z L. A VM Placement Based Approach to Proactively Mitigate Co-Resident Attacks in Cloud[C]. *2018 IEEE Symposium on Computers and Communications*, 2018: 285-291.

[82] Kong T, Wang L M, Ma D H, et al. A Secure Container Deployment Strategy by Genetic Algorithm to Defend Against Co-Resident Attacks in Cloud Computing[C]. *2019 IEEE 21st International Conference on High Performance Computing and Communications*; *IEEE 17th International Conference on Smart City*; *IEEE 5th International Conference on Data Science and Systems*, 2019: 1825-1832.

[83] Moving Target Defense Using Live Migration of Docker Containers. https://arxiv.org/pdf/1611.03065.pdf

[84] Huang R, Zhang H Q, Liu Y, et al. RELOCATE: A Container Based Moving Target Defense Approach[C]. *Proceedings of The 7th International Conference on Computer Engineering and Networks — PoS*, 2017: 1-7.

[85] Azab M, Eltoweissy M. MIGRATE: Towards a Lightweight Moving-Target Defense Against Cloud Side-Channels[C]. *2016 IEEE Security and Privacy Workshops*, 2016: 96-103.

[86] Azab M, Mokhtar B M, Abed A S, et al. Smart Moving Target Defense for Linux Container Resiliency[C]. *2016 IEEE 2nd International Conference on Collaboration and Internet Computing*, 2017: 122-130.

[87] Pod Security Policies. https://kubernetes.io/docs/concepts/policy/pod-security-policy. Dec. 2020.

[88] Manage sensitive data with Docker secrets. https://docs.docker.com/engine/swarm/secrets. Dec. 2020.

[89] Mirantis Secure Registry. https://www.mirantis.com/software/docker/image-registry. Dec. 2020.

[90] Content trust in Docker. https://docs.docker.com/engine/security/trust. Dec. 2020.

[91] Giannakopoulos I, Papazafeiropoulos K, Doka K, et al. Isolation in Docker through Layer Encryption[C]. *2017 IEEE 37th International Conference on Distributed Computing Systems*, 2017: 2529-2532.

[92] Xu Q Q, Jin C, Bin Mohamed Rasid M F, et al. Blockchain-Based Decentralized Content Trust for Docker Images[J]. *Multimedia Tools and Applications*, 2018, 77(14): 18223-18248.

[93] Xu Q Q, Jin C, Bin Mohamed Rasid M F, et al. Decentralized Content Trust for Docker Images[C]. *The 2nd International Conference on Internet of Things*, *Big Data and Security*, 2017: 431-437.

[94] Clair. https://github.com/quay/clair. Dec. 2020.

[95] Anchore. https://anchore.com. Dec. 2020.

[96] Dockerscan. https://github.com/cr0hn/dockerscan. Dec. 2020.

[97] Trivy. https://github.com/aquasecurity/trivy. Dec. 2020.

[98] Docker Security Scanning safeguards the container content lifecycle. https://www.docker.com/blog/docker-security-scanning. Dec. 2020.

[99] Sultan S, Ahmad I, Dimitriou T. Container Security: Issues, Challenges, and the Road Ahead[J]. *IEEE Access*, 2019, 7: 52976-52996.

孔同 于 2015 年在北京交通大学信息安全专业获得学士学位。现在中国科学院信息工程研究所攻读博士学位。CCF 学生会员。研究领域为云计算安全、网络与系统安全。研究兴趣包括:虚拟化安全、网络安全、移动目标防御。Email: kongtong@iie.ac.cn

王利明 于 2007 年中国科学院软件所计算机科学与技术专业获得博士学位。现任中国科学院信息工程研究所第五研究室正高级工程师, 博士生导师。研究领域为云计算与网络安全、移动通信系统安全、大数据安全分析、关键基础设施安全。Email: wangliming@iie.ac.cn

徐震 于2005年在中国科学院软件所计算机科学与技术专业获得博士学位。现任中国科学院信息工程研究所第五研究室主任, 正高级工程师, 博士生导师。研究领域为云计算安全、可信计算、移动互联网安全、网络与系统安全等。Email: xuzhen@iie.ac.cn

马多贺 于 2015 年在中国科学院信息工程研究所信息安全国家重点实验室获得博士学位。现任中国科学院信息工程研究所副研究员。CCF 高级会员, CISSP。研究领域为移动目标防御、智能安全、云安全、网络与系统安全。Email: maduohe@iie.ac.cn