QINGTENG

容器安全指南及解决方案

青藤云安全

Adapted from NIST.SP.800-190 Application Container Security Guide

计算机系统技术报告

美国国家标准技术研究院（NIST）的信息技术实验室（ITL）通过为美国的测量和标准基础设施提供技术指导，促进了美国经济和公共福利的发展。ITL 开发测试、测试方法、参考数据、概念验证、以及技术分析，促进信息技术的开发和生产使用。ITL 的职责包括制定管理、行政、技术和物理标准以及指南，以经济有效地保护联邦信息系统中除国家安全相关信息以外的安全性和隐私。特别出版物 800 系列报告了 ITL 在信息系统安全性方面的研究、指南和探究工作，以及与行业、政府和学术组织的合作活动。

摘要

容器技术，也称为容器，是操作系统虚拟化与应用软件打包相结合的一种形式。容器提供了一种可移植、可重用的自动化方式来打包和运行应用。该出版物解释了与容器使用相关的潜在安全隐患，并对解决这些隐患提出了建议。

关键词

应用；应用软件打包；容器；容器安全；隔离；操作系统虚拟化；虚拟化

执行摘要

操作系统（OS）虚拟化为每个应用单独提供了一个操作系统虚拟化视图，从而使每个应用与服务器上的所有其它应用隔离开来。每个应用只能看到并影响自身。近年来，由于 OS 虚拟化易用性程度提高、并提高了开发人员敏捷性等关键效益，OS 虚拟化已变得越来越流行。当今的 OS 虚拟化技术主要致力于提供一种可移植、可重用的自动化方式来打包和运行应用（app）。术语“应用容器”或简称为“容器”经常用于指代这些技术。

该文档的目的是解释与容器技术有关的安全问题，并为规划、实施和维护容器时解决这些问题提出切实可行的建议。许多建议是针对容器技术架构（如图 1 所示）中的具体组件或层而提出的。

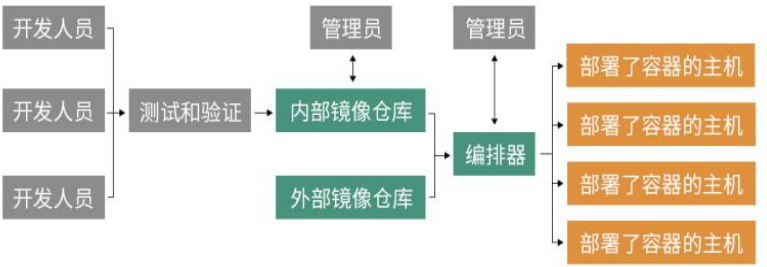


图 1：容器技术架构的层和组件

组织机构应遵循这些建议，以确保其容器技术的实施和使用安全：

（1）通过使用容器，可以调整组织机构的运营方式和技术流程，支持以全新方式来开发、运行和支持应用。

采用容器技术可能会破坏组织机构内的现有方式和软件开发方法。传统的开发实践、补丁技术和系统升级过程可能无法直接应用到容器化环境中，因此员工愿意适应这种新模型，这一点非常重要。应鼓励员工采纳本指南中涵盖的在容器内安全构建和运行应用的推荐做法，并且组织机构应愿意反思现有程序，充分利用容器。应向软件开发生命周期内所涉及的所有人提供有关技术和运行方式的教育与培训。

（2）使用容器专用主机操作系统而不是通用操作系统以减少攻击面。

容器专用主机操作系统是一种极简操作系统，仅设计用于运行容器，而禁用所有其它服务和功能，并且部署了只读文件系统和其它加固措施。当使用容器专用主机操作系统时，攻击面通常

比使用通用主机操作系统小很多，所以攻击和入侵容器专用主机操作系统的几率更小。因此，组织机构应尽量使用容器专用主机操作系统来降低风险。但重要的是要注意，专用主机操作系统随着时间的推移也会有漏洞需要修复。

（3）只将具有同样目的、敏感性和威胁态势的容器组合放在同一主机操作系统内核中，以提高纵深防御能力。

虽然大多数容器平台在将容器与容器隔离、将容器与主机操作系统隔离方面做的卓有成效，但在同一主机操作系统上运行不同敏感级别的应用时会存在不必要的风险。依据目的、敏感性和威胁态势来将容器分隔开来，可以提高纵深防御能力。组织机构按照这种方式对容器分组，入侵某一组容器的攻击者就很难横向移动到其它组容器。这提高了发现和控制入侵的可能性，并且确保任何残留数据，如缓存或加载临时文件的本地数据卷，保留在安全区域内。

在数以百计的主机和数以千计的容器这样大规模的环境中，自动化分组才具有现实可操作性。幸运的是，容器技术通常包含某种机制，能够将应用组合在一起，而且容器安全工具可以使用

像容器名称和标签等属性来对容器执行安全策略。

（4）采用容器专用的漏洞管理工具和过程，防止镜像遭到入侵。

传统的漏洞管理工具对主机持久性和应用更新机制及频率做了很多假设，但这与容器化模型完全不相符。例如，传统的漏洞管理工具常常假定，某一服务器长时间运行一组相同的应用，但不同的容器实际上可能是根据资源可用性情况，在任何给定时间在不同的服务器上运行。此外，传统工具往往无法检测容器内的漏洞，从而产生一种虚假的安全感。组织机构使用的工具应采用声明式、分步构建的方法，并将容器和镜像的不变性融入其设计中，从而提供更可行、更可靠的结果。

这些工具和过程应考虑到镜像软件漏洞和配置设置。组织机构应采用工具和过程来验证并强制确保镜像符合安全配置最佳惯例。这应包括对每个镜像的合规性状态进行集中报告和监控，防止运行不合规镜像。

（5）考虑采用基于硬件的防范措施，为可信计算提供基础。

安全应扩展到整个容器技术的所有层。目前，实现这一目标的方法是将安全建立在硬件可信根之上，例如行业标准的可信平台模块（TPM）。在硬件可信根中存储了主机的固件、软件和配置数据的测量结果。在启动主机前用存储的测量结果验证当前的测量结果，这就保证了可以信任主机。硬件的可信链可以扩展到操作系统内核和操作系统组件，从而可以实现针对启动机制、系统镜像、容器运行时和容器镜像的密码验证。可信计算为构建、运行、编排和管理容器提供了一种安全方式。

（6）使用容器感知的运行时防御工具。

部署和使用能够在容器运行时预防、检测和响应威胁的专用容器安全解决方案。传统的安全解决方案，如入侵防御系统（IPS）和 Web 应用防火墙（WAF），通常无法对容器提供恰当防护。这些传统的安全解决方案可能无法运行在大规模的容器上，无法管理容器环境中的频繁变更，也无法检测容器内部的活动。利用容器原生安全解决方案能够监视容器环境，并能够精确检测其中的异常情况和恶意活动。

目 录

1. 引言 1
1.1. 目的和范围 1
1.2. 文档结构 1
2. 容器简介 2
2.1. 应用虚拟化和容器的基本概念 3
2.2. 容器和主机操作系统 7
2.3. 容器技术架构 12
2.3.1 镜像的创建、测试和验证 13
2.3.2 镜像存储和检索 15
2.3.3 容器部署与管理 16
2.4. 容器用途 19
3. 容器技术核心组件的主要风险 21
3.1. 镜像风险 22
3.1.1 镜像漏洞 22
3.1.2 镜像配置缺陷 23
3.1.3 嵌入式恶意软件 23
3.1.4 嵌入式明文秘钥 23
3.1.5 使用不可信镜像 24
3.2. 镜像仓库风险 24
3.2.1 与镜像仓库的连接不安全 24
3.2.2 镜像仓库中的镜像过时 25
3.2.3 认证和授权限制不足 25
3.3. 编排工具风险 25
3.3.1 管理访问权限不受限制 25
3.3.2 未经授权的访问 26
3.3.3 容器间网络流量隔离效果差 26
3.3.4 混合不同敏感度级别的工作负载 27
3.3.5 编排工具节点受信问题 27
3.4. 容器风险 28

3.4.1 运行时软件中的漏洞.....28
3.4.2 容器的网络访问不受限制.....28
3.4.3 容器运行时配置不安全.....29
3.4.4 应用漏洞.....30
3.4.5 流氓容器.....30
3.5. 主机操作系统风险.....31
3.5.1 攻击面大.....31
3.5.2 共享内核.....31
3.5.3 主机操作系统组件漏洞.....31
3.5.4 用户访问权限不当.....32
3.5.5 篡改主机操作系统文件系统.....32
4. 主要风险应对措施.....32
4.1. 镜像应对措施.....33
4.1.1 镜像漏洞.....33
4.1.2 镜像配置缺陷.....34
4.1.3 嵌入式恶意软件.....35
4.1.4 嵌入式明文秘钥.....35
4.1.5 使用不可信镜像.....36
4.2. 镜像仓库应对措施.....37
4.2.1 与镜像仓库的连接不安全.....37
4.2.2 镜像仓库中的镜像过时.....37
4.2.3 认证和授权限制不足.....38
4.3. 编排工具应对措施.....39
4.3.1 管理访问权限不受限制.....39
4.3.2 未经授权的访问.....39
4.3.3 容器间网络流量隔离效果差.....40
4.3.4 混合不同敏感度级别的工作负载.....40
4.3.5 编排工具节点受信问题.....42
4.4. 容器应对措施.....43
4.4.1 运行时软件中的漏洞.....43
4.4.2 容器的网络访问不受限制.....43

4.4.3 容器运行时配置不安全.....45
4.4.4 应用漏洞.....46
4.4.5 流氓容器.....47
4.5. 主机操作系统应对措施.....47
4.5.1 攻击面大.....47
4.5.2 共享内核.....48
4.5.3 主机操作系统组件漏洞.....49
4.5.4 用户访问权限不当.....50
4.5.5 篡改主机文件系统.....50
4.6 硬件应对措施.....50
5. 容器威胁场景案例.....52
5.1. 利用镜像中的漏洞.....53
5.2. 利用容器运行时.....53
5.3. 运行有毒镜像.....54
6. 容器技术生命周期安全考虑因素.....55
6.1. 起始阶段.....56
6.2. 规划设计阶段.....56
6.3. 实施阶段.....58
6.4. 运维阶段.....59
6.5. 处置阶段.....61
7. 青藤基于宿主机 Agent 的全生命周期容器解决方案.....61
7.1. 构建阶段.....64
7.2. 分发阶段.....65
7.3. 运行阶段.....66
7.3.1 运行准备阶段.....66
7.3.2 生产环境阶段.....66
8. 总结.....67

1. 引言

1.1. 目的和范围

本文档的目的是解释与容器技术相关的安全性问题，并提出实用建议，解决在规划、实施和维护容器时遇到的问题。容器某些方面的内容可能因技术而异，但本文档中的建议适用于大多数、乃至所有容器技术。

除容器以外的所有形式的虚拟化，例如虚拟机，均不在本文档讨论范围之内。

本文档假定读者已经熟悉确保为容器技术提供支持并与之交互的技术，其中包括以下内容：

- **容器技术下的各层，包括硬件、虚拟机管理器和操作系统；**
- **在容器中使用应用的管理工具；**
- **用于管理容器内应用以及容器本身管理员终端。**

1.2. 文档结构

本文档的其余部分分为以下部分：

- **第 2 节为容器简介，包括其技术能力、技术架构和用途。**
- **第 3 节说明了容器技术核心组件的主要风险。**
- **第 4 节针对第 3 节中确定的风险提供了对策建议。**
- **第 5 节确定了容器的威胁情景示例。**
- **第 6 节介绍了用于规划、实施、操作和维护容器技术的可行信息。**
- **第 7 节为青藤全生命周期容器解决方案。**
- **第 8 节为本文档的总结。**

2. 容器简介

本节介绍了服务器应用（apps）的容器，首先定义应用虚拟化及容器的基本概念，这是理解本文档其余部分内容所需的。接下来，本节介绍容器如何与运行容器的操作系统进行交互。本节的下一部分说明了容器技术的总体体系结构，定义容器实现中通常用到的所有主要组件，并解释了组件之间的工作流。本部分的最后一部分介绍了容器的常用用法。

2.1. 应用虚拟化和容器的基本概念

NIST 特别出版物（SP）800-125 将虚拟化定义为“对运行软件或硬件的模拟抽象”。虚拟化已经使用了很多年，但以支持云计算而闻名。在云环境中，硬件虚拟化用于在单个物理服务器上运行许多操作系统（OS）实例场景，同时保持每个实例场景独立。这就提高了硬件的使用效率，同时也实现了多租户支持。

在硬件虚拟化中，每个 OS 实例都与虚拟化硬件交互。另一种形式的虚拟化称为操作系统虚拟化，其概念与此类似。它在单个实际 OS 内核之上提供了多个虚拟 OS。这种方法通常称为 OS 容器，并且自二十一世纪初期以来就存在各种 OS 容器，例如初期的 Solaris Zone 和 FreeBSD jails。[1] 2008 年，几乎所有现代发行版都内置了 Linux 容器（LXC）技术。OS 容器与本指南的主题— —容器有所不同，因为 OS 容器旨在提供一个类似于普通 OS 的环境，其中多个应用和服务可能共存。

近年来，由于其易用性取得重大进步，以及为给开发人员提供灵活性这一重要优势，应用虚拟化变得越来越普及。在应用虚拟化中，实际上是有多个单独的应用共享同一个 OS 内核。OS 组件将每个应用实例与服务器上的所有其它实例隔离开来。在这种

[1] 有关 jail 概念的更多信息，请参见 https://www.freebsd.org/doc/handbook/jails.html。

情况下，每个应用仅能查看 OS 及其本身，并且与可能在同一操作系统内核上运行的其它应用隔离开来。

OS 虚拟化和应用虚拟化之间的主要区别在于，通过应用虚拟化，每个虚拟实例通常仅运行一个应用。当今的应用虚拟化技术主要致力于提供一种可移植、可重用和自动化方式来打包和运行应用。“应用容器”或简称“容器”经常用来指代这些技术。该一术语是对船运集装箱的一个类比，它提供了一个标准化方式，将不同内容组合在一起，同时又将它们彼此隔离开来。

传统的应用架构通常将应用划分为几层（例如，Web、应用和数据库），并在每一层都有服务器或虚拟机，与此不同，容器架构通常会将应用划分为更多的组件，每个组件都有一个明确的功能，通常在自己的容器内运行。每个应用组件都在单独的容器中运行。在容器技术中，协同工作构成一个应用的一系列容器集称为微服务。通过这种方法，可提高应用部署的灵活性和可扩展性。由于内置功能也更加丰富，研发也更加简单。但是，还有更多对象需要管理和保护，这可能会导致应用管理以及安全工具和流程出现问题。

大多数容器技术都践行了不变性这一理念。换言之，容器本

身应作为无状态的实体来运行，可以部署但不能更改[2]。当需要升级正在运行的容器或更改其内容时，只需将容器销毁，然后用含有更新内容的容器来代替即可。这让开发人员和支持工程师提高了创建和修改应用的速度。

组织机构可能会从每个季度部署一个新版本的应用，发展到每周或每天部署新组件。不变性是容器和硬件虚拟化之间最根本的运营差异。传统的 VM 通常以有状态实体的形式运行，包括生命周期内的部署、重新配置和升级。传统安全工具和流程通常在很大程度上假定是静态运行，这可能需要进行调整，才能适应容器化环境中的变化频率。

容器的不变性也影响着数据的持久性。容器强调隔离的理念，而不是将应用与其使用的数据混合在一起。数据持久性不应通过简单地写入容器根文件系统来实现，而应通过使用外部持久性数据存储，例如数据库或集群的持久卷。容器使用的数据应存储在容器之外，如此一来，当用新版应用替代运行现有版本的容器时，所有数据仍可用于新版本。

现代容器技术出现时，DevOps 方法已在很大程度上得到采用，以便提高应用构建和运行的集成性，强调开发和运营团队之

[2] 注意，虽然容器的不变性真实可行，但这不是必需的，所以组织机构需要调整运行方式，充分利用这一点。

间的密切协作[3]。容器的可移植性特征之所以特别适于这些方法，是因为它们让组织机构能够维持开发、测试和生产环境的一致性。组织机构经常利用持续集成过程，将其应用直接放入创建过程的容器中。如此一来，从应用生命周期一开始，保证了其运行时环境的一致性。容器镜像（包括运行容器所需的文件的软件包）通常被设计为可在机器和环境之间移植，从而可以轻松地将在开发实验室中创建的镜像移至测试实验室中进行评估，然后复制到生产环境中进行运行，而无需进行任何修改。

这种方法的缺点在于保护容器的安全工具和流程不应假定具体的云服务提供商、主机操作系统、网络拓扑或其它容器运行时环境，因为这些都可能频繁变化。更关键的是，所有这些环境以及应用的整个生命周期的安全性，从开发到测试到生产，都要保持一致。

最近，诸如 Doker 和 rkt 等项目已经提供了额外功能，旨在提高操作系统组件的隔离特性的易用性和可扩展性。从 Windows Server2016 开始，在 Windows 平台上也有了容器技术。所有这些实现的基本架构基本一致，因此，本文档详细讨论容器，而不关注实现详情。

[3] 本文档是指 DevOps 人员执行的任务。提到对这些角色重点是正在执行的工作任务类型，而不是严格的职位名称或团队的组织结构。

2.2. 容器和主机操作系统

比较一下采用硬件虚拟化技术在虚拟机（VM）中部署应用（很多读者对此已经了然于心了），就很容易理解在容器中部署应用了。图 2 左图展示了 VM 部署情况，中间展示了没有 VM（安装在裸机上）时的容器部署，右图展示了在 VM 中的容器部署。

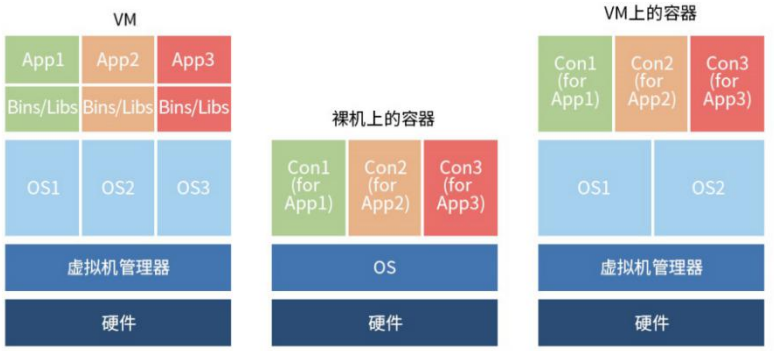


图 2：虚拟机和容器部署

VM 和容器都允许多个应用共享同一物理基础设施，但它们使用不同的隔离方法。VM 使用虚拟机管理器在 VM 之间提供硬件级别的资源隔离。每台 VM 能看到自己的虚拟硬件，除了应用及其数据之外，还包括完整的操作系统。VM 允许不同的操作系统（如 Linux 和 Windows）共享同一物理硬件。

对于容器来说，多个应用共享同一操作系统内核实例，但彼此相互隔离。操作系统内核是所谓主机操作系统的一部分。主机操作系统位于容器之下，为容器提供操作系统功能。容器特定于操作系统家族；Linux 主机仅能运行为 Linux 构建的容器，Windows 主机仅能运行 Windows 的容器。

此外，为一个操作系统家族构建的容器仅能运行于这个家族中最新版本的系统上。

用于运行容器的主机操作系统有两大类。通用操作系统，如 Red Hat Enterprise Linux、Ubuntu 和 Windows Server，可以用于运行很多类型的应用，并且可以将容器专用的功能加进去。容器专用操作系统，如 CoreOS Container Linux、Project Atomic 和 Google Container-Optimized 操作系统是极简操作系统，专门设计仅用于运行容器。这类操作系统通常不含有软件包管理器，而只含有核心管理工具的一个子集，并阻止在容器外运行应用。通常情况下，容器专用主机操作系统采用只读文件系统设计，降低攻击者能够在容器中保留数据的可能性，而且还采用了简化升级过程，并且很少关注应用兼容性方面的问题。

用于运行容器的每个主机操作系统都有对应的二进制文件，用于建立和维护每个容器环境，也称为容器运行时。该容器运行

时协调多个操作系统组件，隔离资源和资源使用量，这样每个容器只能看到自己专用的操作系统视图，与其它同时运行的容器彼此隔离。实际上，容器和主机操作系统通过容器运行时进行交互。容器运行时还提供了管理工具和应用编程接口（API），让 DevOps 人员和其它人设定如何在主机上运行容器。运行时消除了手动创建所有必需配置的需求，并简化了启动、停止和操作容器的过程。运行时的示例包括 Docker、rkt 和 Open Container Initiative Daemon。

容器运行时确保主机操作系统提供的技术功能示例包括以下内容：

- 命名空间隔离限制了可以与容器交互的资源，这包括文件系统、网络接口、进程间通信、主机名、用户信息和进程。命名空间隔离可确保容器内的应用和进程只看到分配给该容器的物理和虚拟资源。例如，如果您在某个主机上运行 Apache 的容器中运行“ps-A”，即使这个主机上还有很多运行其它应用的其它容器，您也只能在结果列表中看到 httpd。命名空间隔离为每个容器提供了自己的网络堆栈，包括独立的界面和 IP 地址。Linux 上的容器使用像带有掩码的进程标识等技术来实现命名空间隔离，而在 Windows 上则使用对象命名空间。

● 资源分配限制了给定的容器可以使用多少主机资源。例如，如果您的主机操作系统有 10GB 的总内存，您可能希望给九个独立的容器中的每一个分配 1GB。任何容器不应该能够干预另一个容器的活动，所以资源分配确保每个容器仅能利用分配给它的资源量。在 Linux 上这主要通过控制组（cgroups）实现[4]，而在 Windows 上，作业对象可实现类似的目标。

● 文件系统虚拟化让多个容器可以共享相同的物理存储，而无需具有对其它容器存储进行访问或变更的能力。虽然可以说类似于命名空间隔离，但文件系统虚拟化是单独调用的，因为它也经常涉及到优化，以确保容器通过诸如写时拷贝等技术有效地使用主机的存储。例如，如果使用同一镜像的多个容器在单个主机上运行 Apache，文件系统虚拟化可确保，仅在磁盘上存储一份 httpd 二进制代码。如果其中一个容器修改了其内部的文件，则只会将具体更改的数据写入磁盘，这些更改将只对执行变更的容器可见。在 Linux 上，这些功能是由像高级多层统一文件系统（AUFS）等技术实现的，而在 Windows 上，这些功能是 NT 文件系统（NTFS）的扩展功能。

容器的技术能力因主机操作系统家族的不同而不同。容器基

[4] cgroups 是可以独立管理的进程集合，让内核能够以软件方式对子系统（如内存、处理器使用和磁盘 I/O）进行测量。管理员可以手动或以编程方式控制这些子系统。

本上是给每个应用一个单独操作系统视图的机制，所以这种分离的实现工具很大程度上依赖于操作系统家族。例如，Linux 和 windows 在进程隔离上所使用的方法是不同的。然而，尽管底层的实现可能不同，但常用的容器运行时提供了一种通用的接口格式，很大程度上将这些差异与用户隔离开。

虽然容器提供了强度很高的隔离，但容器并没有提供像 VM 那样具体明确的安全边界。由于容器共享相同的内核，并且可以在主机上使用不同的功能和特权来运行，因此容器之间的隔离程度远远小于虚拟机管理器对 VM 的隔离。因此，配置环境不细致可能会导致容器能够比同一主机上的多个 VM 更容易、更直接地与其它容器以及主机交互。

虽然有时候容器被视为虚拟化的下一个阶段，超越硬件虚拟化，但事实上大多数组织机构更愿意发展而不是革命。容器和硬件虚拟化不仅能够，而且经常会和谐共处，并且实际上能够彼此强化对方的能力。VM 提供很多益处，如强隔离、操作系统自动化、以及广泛和深入的解决方案生态环境。组织机构无需在容器和 VM 之间做出抉择。与此相反，组织机构可以使用 VM 来部署、划分和管理硬件，同时使用容器来打包应用，并提高每个 VM 的使用效率。

2.3. 容器技术架构

图 3 展示了容器技术架构的五个层：

1. 开发人员系统（生成镜像并将其发送给测试和验证）

2. 测试和验证系统（验证并核实镜像的内容、镜像签名，并将镜像发送到镜像仓库）

3. 镜像仓库（存储镜像并将镜像按需分发给编排工具）

4. 编排工具（将镜像转换为容器，并将容器部署到主机）

5. 主机（根据编排工具的指示运行和终止镜像）

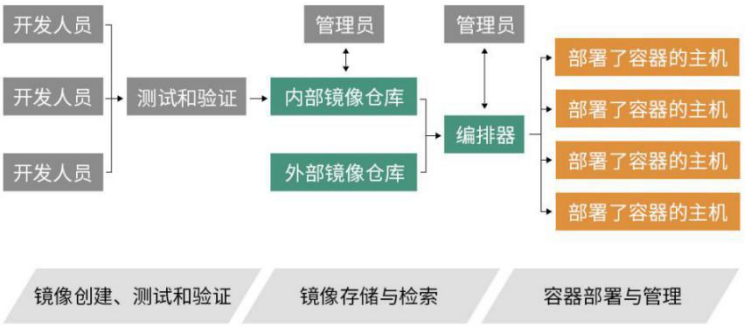


图 3：容器技术架构的层、组件和生命周期

尽管在整个过程中涉及许多管理者系统角色，但该图仅描述了内部镜像仓库和编排工具的管理者系统。

灰色的系统（开发人员系统、测试和验证系统以及管理员系统）不在容器技术架构范围之内，但确实与容器技术架构有着重要的交互。在大多数使用容器的组织机构中，开发和测试环境也利用容器，这种一致性是使用容器的主要优点之一。本文档不着重介绍这些环境中的系统，因为保护系统安全的建议与针对生产环境的建议大体相同。绿色的系统（内部镜像仓库、外部镜像仓库和编排工具）是容器技术架构的核心组件。最后，橘色的系统（有容器的主机）是容器使用的地方。

理解容器技术架构的另一种方法是考虑容器生命周期的阶段，如图 3 底部所示。下文将会对这三个阶段进行更详细的介绍。

由于组织机构通常一次性构建和部署很多不同的应用，所以，在一个组织机构内，这些生命周期的阶段通常同时进行，而不应被视为成熟度的渐进式阶段。相反，应将其视为连续运行引擎中的循环。按照这个比喻，每个应用都是引擎中的一个汽缸，不同的应用可能同时处于这个生命周期的不同阶段。

2.3.1 镜像的创建、测试和验证

在容器生命周期的第一阶段，创建应用的组件，并将其放置在一个镜像中（或者可能放置在多个镜像中）。镜像是一个软件

包，包含运行容器所需的所有文件。例如，运行 Apache 的镜像包括 httpd 二进制文件，以及相关的库和配置文件。镜像应该只包括应用本身所需的可执行文件和库；所有其它 OS 功能由底层主机操作系统中的 OS 内核提供。镜像通常使用诸如分层机制和写时复制机制等技术（其中共享的主镜像是只读的，并将变更记录到不同的文件中），最大限度地减少其对磁盘的占用并提高运行效率。

因为镜像以分层的方式构建，所以，所有其它组件构建于其上的底层通常被称为根镜像层。根镜像层通常是 Ubuntu 和 Windows Nano Server 等通用操作系统的极简版本，不含操作系统内核。用户从其中一个根镜像层开始构建他们的完整镜像，然后添加应用框架及其自定义代码，从而形成其独特应用的完整部署镜像。容器运行时支持使用来自同一操作系统家族的镜像，即使具体的主机操作系统版本并不不同。例如，运行 Docker 的 Red Hat 主机可以运行在任何 Linux 根镜像层上创建的镜像，如 Alpine 或 Ubuntu。但是，容器不能运行用 Windows 根镜像层创建的镜像。

镜像创建过程由将应用打包、转交给测试的开发人员管理。镜像创建通常使用能够自动化管理工具（例如 Jenkins 和 TeamCity）来协助所谓的“持续集成”过程。这些工具获取应用

的各种库、二进制文件和其它组件，对其进行测试，然后依据开发人员创建的应用镜像流程清单，组合镜像。

大多数容器技术采用声明式方法来介绍应用的组件和要求。例如，用于 Web 服务器的镜像不但包括 Web 服务器的可执行文件，而且还包括计算机可解析的数据，用来描述 Web 服务器的运行方法，如侦听端口或服务器使用的配置参数。

镜像创建后，组织机构通常进行测试和验证工作。例如，自动化测试工具和人员会使用所创建的镜像来验证最终应用的功能，安全团队会对这些镜像进行认证。

为应用创建、测试和认证完全相同的复制件，保持其一致性，这是容器运行和安全性的主要效益之一。

2.3.2 镜像存储和检索

镜像通常存储在中央位置，以便各主机对其进行控制、共享、查找和重用。镜像仓库是一种服务，方便开发人员在创建镜像后，将镜像进行存储、签名和分类，实现镜像标识和版本控制，有助于发现和重用，以及查找和下载其它人创建的镜像。镜像仓库可以自托管，也可以作为一种服务来使用。镜像仓库示例包括 Amazon EC2 Container Registry、Docker Hub、Docker Trusted Registry 和 Quay Container Registry。

镜像仓库提供 API，确保能够实现常见镜像相关任务的自动化。例如，组织机构可能会在镜像创建阶段设置触发机制，一旦测试通过会自动将镜像推送到镜像仓库中。镜像仓库可能设置进一步的触发机制，可以在添加新镜像后自动部署新镜像。这种自动化功能可以提高迭代速度，实现更一致的效果。

一旦存储到镜像仓库中，就可以轻松地获取镜像，然后由 DevOps 人员在其运行容器的任何环境中运行镜像。这是容器可移植性效益的另一个示例；也可以在公有云上创建镜像，将镜像推送到托管在私有云中的镜像仓库，然后从此处分发镜像，在最终位置运行该应用。

2.3.3 容器部署与管理

编排工具让 DevOps 人员或自动化工具，能够从镜像仓库中获取镜像，将这些镜像部署到容器中，并管理正在运行的容器。该部署过程会形成应用的一个可用版本，能够运行并准备响应请求。将镜像部署到容器中时，镜像本身不会更改，而是将镜像的一个副本放到容器中，并将一组应用静止代码转换为运行实例。常见的编排工具有 Kubernetes、Mesos 和 Docker Swarm。

请注意，操作简单的轻量型容器的实现也可以无需功能齐全的编排工具。在以下这些情况下，编排工具也可能被省略或无需

存在。例如，主机可以直接联系镜像仓库，从中获取镜像创建容器运行时。为了简化本出版物中的讨论内容，将假定使用编排工具。

编排工具提供的抽象性让 DevOps 人员可以便捷地设定某个镜像运行所需的容器数量、以及每个容器需要分配的资源，如内存、处理能力和磁盘。编排工具了解集群中每个主机的状态，包括每个主机的可用资源，并确定哪些容器将在哪些主机上运行。然后，编排工具从镜像仓库中提取所需的镜像，并在指定资源下用容器运行镜像。

编排工具还负责监控跨主机的容器资源消耗、作业执行和机器健康状况。根据其配置，如果最初运行容器的主机出现故障，编排工具可能会在新主机上自动重启容器。许多编排工具可以实现跨主机的容器组网和服务发现。大多数编排工具还包括一个软件定义网络（SDN）组件，称为覆盖网络，可用于隔离共享同一物理网络的应用之间的通信。

当容器中的应用需要更新时，不会去更改现有的容器，而是将其销毁，并从更新的镜像中创建新的容器。这是使用容器关键一项不同之处：初始部署的基础软件不应随时间而改变，而是通过一次性替换整个镜像来完成更新。这种方式具有重大的潜在安全价值，因为它能够让组织机构在构建、测试、确认和部署每个

阶段配置完全一样的软件。当对应用进行更新时，组织机构可以确保使用最新的版本，通常是利用编排工具来实现。编排工具通常配置为从镜像仓库中获取最新版本的镜像，以便应用始终是最新的。这种“持续交付”的自动化方法让开发人员只需为其应用构建一个新版本的镜像，测试镜像，将其推送到镜像仓库，然后通过自动化工具将其部署到目标环境。

这意味着，在创建新的镜像版本时，开发人员通常会关注所有的漏洞管理，包括补丁和配置设置。对于容器，通常让开发人员而不是运营团队负责应用和镜像的安全性。与以前相比，这种职责的变化往往需要人员之间加强协调与合作。采用容器的组织机构应确保为每个相关者团体制定明确的工作流和团队职责。

容器管理包括安全管理和监控。但为非容器环境设计的安全控制通常不适用于容器。例如，想一想考虑到 IP 地址的安全控制。这对于 VM 和物理服务器来说很有效，因为其静态 IP 地址几个月或几年内都保持不变。与此不同，容器通常是通过编排工具来分配 IP 地址的，因为容器的创建和销毁频率要比 VM 频繁得多，这些 IP 地址也会随着时间的推移而频繁更改。因此，使用依赖静态 IP 地址的安全技术（如基于 IP 地址的防火墙规则来过滤通信）来保护容器非常困难或者无法实现。此外，容器网络还可以包含同一节点、跨不同节点，甚至是跨云的容器之间的通信。

2.4. 容器用途

与其它任何技术一样，容器不是万能的。在许多场景下，容器都是很有价值的工具，但不一定是每个场景的最佳选择。例如，一个拥有大量传统成品软件的组织机构，不可能利用容器来运行大部分软件，因为供应商可能不支持。但是，大多数组织机构会将容器用于多种有价值的用途。示例包括：

● 需要经常更新和部署应用的敏捷式开发。容器的可移植性和声明性让频繁更新更有效率、更容易测试。这使得组织机构能够加快其创新速度并加快软件交付速度，从而可以修复应用代码中的漏洞，并提高更新后软件的测试和部署速度。

● 环境的一致性和区域分隔，让开发人员可以拥有相同但独立的环境来构建、测试和运行应用。容器让开发人员能够在研发用的笔记本电脑系统上本地运行完整的生产应用副本，从而降低了对测试环境的协调和共享需求，并消除了陈旧测试环境的困扰。

● 在“扩容”场景下，应用可能需要根据负载情况在给定时间点快速部署或退役许多新实例。鉴于容器的不变性，可以更容易地可靠扩容实例，因为每个实例都与其它实例完全一样。另外，由于容器通常是无状态的，所以更容易在不再需要时退役容器。

● 对于云原生的应用，开发人员可以从一开始就构建一个微服务架构，确保更高效地迭代，并简化部署。

容器还提供其它效益；例如，由于容器镜像具有不变性，可以提高构建管道的效率。容器会改变生产代码的安装时间和位置。在非容器系统中，需要在生产环境中（即在服务器运行时）安装应用，通常是通过运行手写脚本来管理服务器上应用代码的安装（例如，编程语言运行时、依赖的第三方库、初始化脚本和操作系统工具）。这意味着在生产前构建管道（以及开发人员的工作站）中，任何测试人员都不测试实际的生产环境，而是测试相似环境。这种相似会随着时间的推移逐渐偏离真实生产情况，特别是当生产管理团队和构建系统团队不是同一团队时。这种场景就是“在我电脑上好好的”问题的体现。

使用容器技术时，构建系统会将应用安装在它所创建的（即编译时）镜像中。该镜像是应用的所有用户需要的不会发生改变的快照。在生产环境下，只需要下载和运行构建系统创建的容器镜像。这就解决了“在我电脑上好好的”这一问题，因为开发人员、构建系统和生产系统都运行着相同的不会发生改变的制品。

现代容器技术通常也强调重复使用，这样，开发人员创建的容器镜像可以很容易地供其它开发人员共享和重用，无论是同一

组织机构还是在其它组织机构的开发人员。镜像仓库服务提供了集中式的镜像共享和发现服务，让开发人员能够轻松地查找和重复使用其它人创建的软件。鉴于这种易用性，许多流行软件供应商和项目使用容器，以此让客户能够轻松地找到并快速运行其软件。例如，与其直接在主机操作系统上安装 MongoDB 等应用，用户只需运行 MongoDB 的容器镜像即可。此外，由于容器运行时将容器与容器、容器与主机操作系统隔离开来，这些应用可以更安全可靠地运行，用户无需担心这些应用会干扰底层的主机操作系统。

3. 容器技术核心组件的主要风险

本节介绍和分析容器技术核心组件（例如镜像、镜像仓库、编排工具、容器和主机操作系统）的主要风险。由于分析仅关注核心组件，因此，无论是否采用容器技术、采用何种主机操作系统平台或位置（公有云、私有云等），本节的分析适用于大多数容器部署。本节考虑了两种类型的风险：

（1）镜像或容器遭到入侵。

使用了 NIST SP800-154 中所述的以数据为中心的系统威胁建模方法来评估该风险。要保护的主要“数据”是镜像和容器，其中可能包含应用文件、数据文件等。要保护的第二种数据是共享

主机资源（如内存、存储和网络接口）中的容器数据。

（2）滥用容器攻击其它容器、主机操作系统、其它主机等。

涉及核心组件的所有其它风险，以及涉及非核心容器技术组件（包括开发人员系统、测试和认证系统、管理员系统、主机硬件和虚拟机管理器）的风险不在本文档的讨论范围之内。

3.1. 镜像风险

3.1.1 镜像漏洞

由于镜像实际上是静态存档文件，其中包含运行给定应用的所有组件，因此镜像中的组件可能缺乏重大安全更新或已过时。使用最新组件创建的镜像可能在创建后的几天或几周内不存在已知漏洞，但在某些时候，会在一个或多个镜像组件中发现漏洞，这时，镜像将不再是最新镜像了。

在传统的操作模式中，所部署的软件在其运行的主机上“现场”更新，与此不同，容器则必须在上游的镜像中进行更新，然后重新部署。因此，容器化环境的常见风险是由于用于创建容器的镜像版本存在漏洞，因而所部署的容器存在漏洞。

3.1.2 镜像配置缺陷

除了软件缺陷，镜像也可能存在配置缺陷。例如，镜像没有使用特定用户账户进行配置来运行，因而，运行时所需的权限更高。另一个示例是，镜像可能包含一个 SSH 守护进程，从而让容器面临不必要的网络风险。这很像传统的服务器或 VM，配置不当仍然可能会让全新的系统面临攻击危险，所以，即使包含的所有组件都是最新的，镜像配置不当也还是会增加风险。

3.1.3 嵌入式恶意软件

由于镜像只是打包在一起的文件集合，其中可能会有意或无意地包含恶意文件。此类恶意软件与镜像中的任何其它组件具有相同能力，因此可以用来攻击环境中的其它容器或主机。嵌入式恶意软件的一个可能来源是所使用的根镜像层，以及通过第三方提供的来源不完全清楚的其它镜像。

3.1.4 嵌入式明文秘钥

许多应用需要秘钥才能实现组件之间的安全通信。例如，Web 应用可能需要用户名和密码才能连接到后端数据库。嵌入式秘钥的其它示例包括连接字符串、SSH 私钥和 X.509 私钥。将一个应用被打包成镜像时，这些秘钥可以直接嵌入到镜像文件系统中。然而，这种做法造成了安全风险，因为能够访问镜像的任何

人都可以轻易对其进行分析，获取秘钥。

3.1.5 使用不可信镜像

在任何环境中，一个最常见的高风险场景是运行不可信软件。由于容器可移植、易于重复使用，更可能诱使团队运行可能无法妥善验证、或不可信的外部来源的镜像。例如，对于一个Web 应用进行故障排查时，用户可能会发现第三方提供的镜像中存在另一版本的应用。使用外部提供的镜像会造成与外部软件通常面临的相同类型的风险，如引入恶意软件、数据泄漏、或包含有漏洞的组件。

3.2. 镜像仓库风险

3.2.1 与镜像仓库的连接不安全

镜像中经常包含敏感组件，例如组织机构的专有软件和嵌入式秘钥。如果与镜像仓库的连接通过不安全的通道进行，镜像的内容会与用明文传输任何其它数据一样存在同样的保密性风险。还会提高中间者攻击的风险，导致可能会截取用于镜像仓库的网络流量，偷取流量中开发人员或管理员的凭证，给编排工具等提供虚的或过时的镜像等。

3.2.2 镜像仓库中的镜像过时

因为镜像仓库通常是组织机构部署的所有镜像的来源位置，随着时间的推移，这些镜像可能包含许多有漏洞的过期版本。虽然这些有漏洞的镜像并不会因为仅仅存储在镜像仓库中就直接构成威胁，但他们增加了意外部署已知晓含有漏洞版本的可能性。

3.2.3 认证和授权限制不足

因为镜像仓库中可能包含用于运行敏感或专有应用以及访问敏感数据的镜像，认证和授权要求不足可能导致知识产权被盗，或将应用的大量技术细节暴露给攻击者。更严重的是，因为通常会信任镜像仓库，将其作为有效且获得批准的软件源，因此，镜像仓库遭到入侵有可能导致下游容器和主机遭到入侵。

3.3. 编排工具风险

3.3.1 管理访问权限不受限制

从历史上看，许多编排工具在设计时都假设，与其交互的所有用户都是管理员，并且管理员应具有对整个环境的控制能力。然而，在许多情况下，单个编排工具可以运行许多不同的应用，每一个由不同的团队管理，并具有不同的敏感性级别。如果提供给用户和不同分组的访问权限不局限于其特定需求，恶意或粗心

的用户可能会影响或破坏编排工具管理的其它容器的运行。

3.3.2 未经授权的访问

编排工具通常有自己的身份鉴别目录服务，这可能与组织机构内部已经使用的常见目录不同。这可能导致编排工具中帐户管理实践薄弱，并且存在“孤立”帐户，因为这些系统缺乏严格的管理。由于很多此类帐户在编排工具中享有很高特权，入侵这些账户可能会导致整个系统范围遭到入侵。

容器使用的数据存储卷通常是由编排工具管理的，而且不是主机特定的数据存储卷。由于容器可以在集群中任一给定节点上运行，而该容器内应用所需的数据必须是容器可用的数据，无论它运行哪台主机上。与此同时，许多组织机构管理的数据在静态下是必须加密的，以防止未经授权的访问。

3.3.3 容器间网络流量隔离效果差

在大多数容器化环境中，各个单独节点之间的流量是通过虚拟覆盖网络路由。该覆盖网络通常由编排工具管理，通常对现有的网络安全和管理工具是不透明的。例如，传统的网络过滤器看不到从 Web 服务器容器发送到另一台主机上的数据库容器的数据库查询，只能看到两个主机之间流动的加密数据包，看不到实际容器终端内部的活动，也看不到发送的流量。虽然加密的覆盖网

络提供了许多运行和安全效益，但也可能造成安全“盲点”，这种情况下，组织机构无法有效地监控自己网络内部的流量。

可能更严重的是，共享同一虚拟网络的不同应用存在流量风险。如果不同敏感级别的应用，诸如面向公众的网站和内部财务管理应用，使用同一虚拟网络，敏感的内部应用面临网络攻击的风险更大。例如，如果面向公众的网站被入侵，攻击者可能会利用共享网络来攻击财务应用。

3.3.4 混合不同敏感度级别的工作负载

编排工具通常重点关注工作负载规模和密度的管理。这意味着，在默认情况下，编排工具可以将不同敏感度级别的工作负载置于同一主机上。例如，在默认配置下，编排工具可能将一个运行面向公众的 Web 服务器的容器与运行处理敏感财务数据的容器置于同一台主机上，只是因为该主机在部署时恰巧有最多可用资源。在 Web 服务器有严重漏洞的情况下，这可能会提高处理敏感财务数据的容器面临的入侵风险。

3.3.5 编排工具节点受信问题

维护环境中各节点之间的信任时需要特别小心。编排工具是最基本的节点。编排工具配置不当可能让编排工具和所有其它容器技术组件面临更大风险。可能后果的示例包括：

- 未经授权的主机加入集群并运行容器

- 集群中单个主机被入侵意味着整个集群被入侵——例如，如果用于认证的密钥在所有节点中共享

- 编排工具和 DevOps 人员、管理员和主机之间的通信是未加密和未认证的

3.4. 容器风险

3.4.1 运行时软件中的漏洞

虽然运行时软件中的漏洞比较少见，但是，如果因为漏洞导致“容器逃逸”等情况的发生，恶意软件就可以攻击其它容器以及主机操作系统本身的资源，这就特别危险了。攻击者还可能能够利用漏洞入侵运行时软件本身，然后篡改该软件，从而让攻击者访问其它容器，监控容器与容器之间的通信等。

3.4.2 容器的网络访问不受限制

在大多数容器运行时的默认状态下，各容器都能够通过网络访问其它容器以及主机操作系统。如果容器被入侵并采取恶意行为，那么，允许存在这种网络流量可能会使环境中的其它资源面临危险。例如，遭到入侵的容器可以被用于扫描它所连接的网络，以便找到让攻击者可利用的其它弱点。这种风险与第 3.3.3 节

中讨论的虚拟网络隔离效果差有关，但也有所不同，因为它更侧重于从容器流出到其它目的地，而不是应用“相互通信”的情况。

由于容器之间大部分是虚拟化连接，因而，在容器化环境下管理对外网络访问更加复杂。因此，从一个容器到另一个容器的数据流在网络中可能只显示为封装的数据包，而没有直接表明其根本来源、目的地或有效载荷。不能感知容器的工具和操作流程无法检查这些该流量，也无法确定其是否存在威胁。

3.4.3 容器运行时配置不安全

容器运行时通常会给管理员提供多种配置选项。容器运行时配置不当会降低系统的相对安全性。例如，在 Linux 容器主机上，经允许的系统调用集通常默认仅限于容器安全运行所必需的调用。如果该列表被扩大，则被入侵的容器会让其它容器和主机操作系统面临更大风险。同样，如果容器在特权模式下运行，则可以访问主机上的所有设备，从而让其本质上成为主机操作系统的一部分，并影响在主机操作系统上运行的所有其它容器。

运行时配置不安全的另一个示例是允许容器在主机上装载敏感目录。容器通常很少会对主机操作系统的文件系统进行更改，而且几乎不应该更改控制主机操作系统基本功能的位置（例如，

Linux 容器的/boot 或/etc、Windows 容器的 C:\Windows）。如果允许遭到入侵的容器更改这些路径，那么，也可以被用来提权并攻击主机本身以及主机上运行的其它容器。

3.4.4 应用漏洞

即使组织机构采取本指南中建议的预防措施，但由于容器运行的应用存在缺陷，容器仍可能受到入侵。这不是容器本身的问题，而只是容器环境中典型软件缺陷的表现。例如，容器化的 Web 应用可能容易受到跨站脚本漏洞的攻击，数据库前端容器可能会受到 SQL 注入的影响。当容器遭到入侵时，容器可能会通过多种方式被滥用，例如允许非授权访问敏感信息，或实现对其它容器或主机操作系统的攻击。

3.4.5 流氓容器

流氓容器是环境中计划之外或未经批准的容器。这可能是一个常见现象，尤其是在开发环境中，应用开发人员可能会启动容器，以此来测试其代码。如果这些容器没有经过严格的漏洞扫描和适当配置，很有可能更容易被利用。因此，流氓容器给组织机构带来了额外的风险，尤其是流氓容器存在于环境中，而开发团队和安全管理员却没有意识到的时候。

3.5. 主机操作系统风险

3.5.1 攻击面大

每个主机操作系统都有一个攻击面，这是攻击者可以尝试访问和利用主机操作系统各种漏洞的所有方法的集合。例如，任何可访问网络的服务都为攻击者提供了潜在的入口点，增大了攻击面。攻击面越大，攻击者就越有可能找到并利用漏洞，从而导致主机操作系统以及在主机操作系统上运行的容器遭到入侵。

3.5.2 共享内核

与通用操作系统相比，容器专用操作系统的攻击面要小得多。例如，容器专用操作系统不包含使通用操作系统用来运行数据库和 Web 服务器应用的库和软件包管理器。不过，虽然容器提供了功能强大的软件级别的资源隔离功能，但使用共享内核无疑会导致相对于虚拟机管理器甚至容器专用操作系统而言更大的攻击面。换言之，容器运行时提供的隔离级别不像虚拟机管理器所提供的那样高。

3.5.3 主机操作系统组件漏洞

所有主机操作系统，甚至是容器专用的操作系统，都提供基本的系统组件，例如用于鉴别远程连接的加密库和用于通用过程

调用和管理的内核原语。与其它任何软件一样，这些组件也会有漏洞，而且由于这些组件存在于容器技术架构的底层，所以会影响运行在这些主机上的所有容器和应用。

3.5.4 用户访问权限不当

由于交互式用户登录应该很少，因此容器专用操作系统通常没有进行优化，以支持多用户场景。当用户不通过编排层，直接登录到主机来管理容器时，组织机构就会面临风险。直接管理可能造成系统及系统上所有容器产生广泛更改，并有可能让只需要管理特定容器的用户能够影响到许多其它人。

3.5.5 篡改主机操作系统文件系统

容器配置不安全可能会让主机中文件被篡改的风险增大。例如，如果允许容器在主机操作系统上装载敏感目录，则该容器就可以更改这些目录中的文件。这些更改可能会影响主机及主机上运行的所有其它容器的稳定性和安全性。

4. 主要风险应对措施

本节针对第 3 节中确定的主要风险提出了对策和建议。

4.1. 镜像应对措施

4.1.1 镜像漏洞

需要使用专用的容器漏洞管理工具和流程。传统的漏洞管理工具在主机持久性和应用更新机制和频率方面做出很多假设，但这与容器化模型完全不相符。这些工具往往无法检测容器内部漏洞，从而产生虚假的安全感。

组织机构使用的工具应采用基于管道的构建方法，并将容器和镜像的不变性融入其设计中，从而提供更可行、更可靠的结果。有效工具和流程的关键内容包括：

1. 与镜像的整个生命周期集成，从构建过程初期开始，到组织机构使用的镜像仓库，到运行时。

2. 对镜像所有层的漏洞均实现可视化，而不只是根镜像层，还包括组织机构正在使用的应用框架和自定义软件。应在整个组织机构范围内统一提供这种可视化功能，并提供符合组织机构业务流程的灵活报告和监控视图。

3. 策略驱动的执行：组织应能够在构建和部署过程的每个阶段创建“质量门户”，确保只允许继续执行符合组织机构漏洞和配置策略的镜像。例如，组织机构应能够在构建过程中配置规则，

防止继续执行那些超过安全漏洞评分系统（CVSS）评级既定阈值的漏洞。

4.1.2 镜像配置缺陷

组织机构根据安全配置最佳实践要求采用对应工具和流程来验证并强制执行合规要求。例如，应将镜像配置为以非特权用户的身份运行。应采用的工具和流程包括：

1. 验证镜像配置设置，包括供应商建议和第三方的最佳实践。

2. 对镜像合规状态持续不断的更新、集中报告和监控，确定组织机构级别存在的薄弱环节和风险。

3. 通过选择防止不合规镜像运行，以强制执行合规性要求。

4. 只使用来自可信来源的根镜像层、常用更新版本、以及从极简技术（如 Alpine Linux 和 Windows Nano Server）中选择根镜像层，减小攻击面的大小。

对镜像配置的最后建议是，在容器内决不能启用 SSH 和其它旨在向主机提供远程 shell 的远程管理工具。容器应以不可变的方式运行，以便从其使用中获得最大的安全效益。通过这些工具启用远程访问容器意味着一定程度的更改，这违反了该原则，并使

容器面临更大的网络攻击风险。相反，所有容器的远程管理都应该通过容器运行时的 API 来完成，这可以通过编排工具，或者通过创建远程 shell 会话连接到运行容器的主机来进行访问。

4.1.3 嵌入式恶意软件

组织机构应持续监控所有镜像的嵌入式恶意软件。监控过程应包括使用恶意软件签名库，以及基于大量实际的非现场攻击的行为检测。

4.1.4 嵌入式明文秘钥

秘钥应存储在镜像之外，并在运行时根据需要动态提供。大多数编排工具，如 Docker Swarm 和 Kubernetes，包含原生的秘钥管理方法。这些编排工具不仅安全存储秘钥并向容器“及时”注入，而且还使构建和部署过程中集成秘钥管理变得更加简单。例如，组织机构可以使用这些工具将数据库连接字符串安全地提供给 Web 容器。编排工具可以确保只有 Web 容器能够访问该秘钥，并且不会将其持久化到磁盘，而是在部署 Web 应用时提供该秘钥。

组织机构还可以将其容器部署与现有的在非容器环境中企业机密管理系统集成在一起。这些工具通常提供用于在部署容器时安全地取回机密的 API，从而消除了将它们保留在镜像中的必要

性。

无论选择什么工具，组织机构都应确保基于预先定义和管理员控制的设置，只将秘钥提供给需要它们的具体容器，并且秘钥在静止状态下总是加密的，在传输中使用联邦信息处理标准（FIPS）140 批准的密码算法[5]。

4.1.5 使用不可信镜像

组织机构应维护一组可信镜像和镜像仓库，并确保仅允许该组镜像在其环境中运行，从而降低部署不可信或恶意组件的风险。

为了降低这些风险，组织机构应采取多层次的方法，包括：

- 能够集中控制在其环境中需要信任哪些具体的镜像和镜像仓库；
- 使用 NIST 认可的措施，通过加密签名对每个镜像进行单独标识[6]；
- 强制执行，以确保环境中的所有主机仅运行已批准列表中的镜像；
- 在镜像执行前验证镜像签名，以确保镜像来自可信来源且

[5] 欲了解 NIST 认可的加密实现的更多信息，请参见加密模块验证项目（CMVP）网页：https://csrc.nist.gov/groups/STM/cmvp/。

[6] 欲了解 NIST 认可的加密实现的更多信息，请参见加密模块验证项目（CMVP）网页：https:/csrc.nist.gov/projects/cryptographic-module-validation-program。

未被篡改；

● 对这些镜像仓库进行持续监控和维护，确保随着漏洞和配置要求的变化维护和更新存储库中的镜像。

4.2. 镜像仓库应对措施

4.2.1 与镜像仓库的连接不安全

组织机构应将其开发工具、编排工具和容器运行时配置为只通过加密信道连接到仓库系统。不同工具具体的步骤不同，但关键目标是确保，只通过可信端点向镜像仓库中推送并从中获取所有数据，并在传输过程中使用加密技术。

4.2.2 镜像仓库中的镜像过时

通过两种主要方法可以降低使用过时镜像的风险。首先，组织机构可以删除镜像仓库中不应再使用的不安全、易受攻击的镜像。该过程可以基于时间触发器以及与镜像相关的标签自动进行。

其次，运行实践应强调使用不可变名称来访问镜像，这些名称指定了要使用镜像的版本。例如，配置部署工作时使用特定版本的镜像，例如，“my-app：2.3”和"my-app：2.4”，而不是使用名称为“my-app”的镜像，以确保在每个作业中部署镜像特定的、已知的完好实例。

另一个选项是对镜像上使用“最新”标记，并在自动化部署时引用该标记。然而，因为这个标签只是附加到镜像的一个标签，并非对最新版本的保证，因此，组织机构应该谨慎使用，不应过分信任。不管组织机构是采用具体定义名称还是“最新”标记，关键是所采用的流程要确保自动化工具使用最新版本的唯一名称，或者确保标记为“最新”的镜像确实是最新版本。

4.2.3 认证和授权限制不足

对包含专有或敏感镜像的镜像仓库，所有访问都应要求进行身份验证。对镜像仓库的可写入访问都应要求进行身份验证，确保只能向镜像仓库添加来自可信实体的镜像。例如，只允许开发人员将镜像推送到其负责的特定仓库，而不是能够更新任何仓库。

组织机构应考虑联合现有帐户（如其自身或云提供商的目录服务），充分利用已为这些帐户建立的安全控制措施。应对镜像仓库的所有写入的访问权限进行审计，并且对敏感镜像的任何读取操作也应进行类似的日志记录。

镜像仓库还提供了机会，将情景感知的授权控制措施应用于操作。例如，组织机构可以配置其持续集成过程，允许授权人员对镜像签名，并仅在通过漏洞扫描和合规性评估后才将其推送到

镜像仓库。组织机构应将这些自动扫描集成到其流程中，防止升级和部署有漏洞或配置错误的镜像。

4.3. 编排工具应对措施

4.3.1 管理访问权限不受限制

特别是由于编排工具广泛的控制范围，编排工具应运用最小授权访问模式，在这种模式下，只授权用户其工作职责所要求的在特定主机、容器和镜像上执行特定操作的权力。例如，应只允许测试团队成员访问在测试中使用的镜像以及用于运行这些镜像的主机，并且只能够操纵他们所创建的容器。测试团队成员对于生产中使用的容器只有有限的访问权限，或没有访问权限。

4.3.2 未经授权的访问

应严格控制对集群范围内的管理帐户的访问权限，因为这些账户能够影响环境中所有资源。组织机构应采用强有力的身份验证方法，如要求多因素身份验证，而不仅仅是单一口令。

组织机构应尽可能对现有的目录系统实行单点登录。单点登录简化了编排工具的验证过程，更便于用户使用强大的身份验证凭证，并集中对访问进行审计，提高异常检测的效率。

对静止数据加密的传统方法通常涉及到使用基于主机的功

能，这可能与容器不兼容。因此，组织机构应使用专用于容器数据的加密工具，从而让无论运行于哪个节点上的容器都能够适当地访问数据。这种加密工具应使用 NIST SP800-111 中定义的加密方法，对于非授权访问和篡改提供相同的壁垒。

4.3.3 容器间网络流量隔离效果差

应配置好编排工具，根据敏感度级别将网络流量分配到不同的虚拟网络中。虽然按照应用进行细分也是可以的，但对大多数组织机构和用例来说，简单地依据敏感度级别定义网络，也能充分降低风险，将复杂度控制在可管理的范围内。例如，面向公众的应用可以共享一个虚拟网络，内部应用可以使用另一种，并且两者之间的通信应通过几个明确确定的接口来实现。

4.3.4 混合不同敏感度级别的工作负载

应配置好编排工具，按照敏感度级别将对特定主机集合上的部署隔离开来。用于实现这个要求的具体方法因所使用的编排工具不同而异，但通用模型定义的规则可防止将高敏感度工作载荷放在运行较低敏感度载荷的同一主机上。这可以通过在编排工具中使用主机“锁定”来实现，或者简单地对每个敏感级别建立单独管理的集群来实现。

虽然大多数容器运行时环境在将容器与容器、容器与主机操

作系统隔离方面卓有成效，但在有些情况下，在同一主机操作系统上运行不同敏感度级别的应用会产生不必要的风险。根据目的、敏感度和威胁态势将容器隔离开来增强了纵深防御能力。在规划应用部署时应考虑诸如应用分层以及网络和主机分隔等概念。例如，假设主机正在运行的容器既有财务数据库又有面向公众的博客。虽然容器运行时一般会有效地将这些环境彼此隔离，但应用的 DevOps 团队要共同负责其安全运行以及消除不必要的风险。如果博客应用被攻击者入侵，而两个应用运行在同一主机上，那就没有什么防御层来保护数据库了。

因此，最佳惯例是依据相对敏感性将容器分组，并确保给定主机内核只运行同一敏感度级别的容器。这种分隔可以通过使用多台物理服务器来实现，但现代的虚拟机管理器也可以提供功能强大的隔离，有效缓解这些风险。就前面的例子来说，这可能意味着组织机构的容器有两种敏感度级别。一个是财务应用，数据库包含在该组容器内。另一个是 Web 应用，博客包含在该组容器内。组织机构就会有两个 VM 池，每种敏感度级别的容器运行在一个 VM 池中。例如，称为 vm-financial 的主机可以托管运行财务数据库以及纳税申报软件的容器，而名为 vm-web 的主机则可以托管博客和公共网站。

通过这种方式对容器进行隔离，入侵某一组容器的攻击者就

很难扩展到攻击其它分组。入侵单个服务器的攻击者只对该主机又访问权限，对其它敏感度级别的容器的侦察和攻击能力就会受到限制。这方法还可以确保任何残留数据，如缓存或加载临时文件的本地卷，保留在数据安全区域中。就前面的例子来说，这种分区将确保在容器终止后，缓存在本地和残留的任何财务数据都不会在运行较低敏感度级别应用的主机上存在。

在数以百计的主机和数以千计的容器这样大规模的环境中，自动化分组才具有现实可操作性。幸运的是，容器技术通常包含某种机制，能够将应用组合在一起，而且容器安全工具可以使用像容器名称和标签等属性来对容器执行安全策略。在这些环境中，除了简单的主机隔离之外，纵深防御的其它层也可能利用这种隔离。例如，组织机构可以部署单独的主机区域或网络，不仅能够隔离虚拟机管理器内的这些容器，而且还能单独隔离其网络流量，如此一来，一种敏感级别的应用流量就会与其它敏感度级别的流量分隔开来。

4.3.5 编排工具节点受信问题

应配置好编排平台，提供创建安全环境所需的功能，以便运行应用。编排工具应确保将节点安全地引入集群，在整个生命周期内保持身份标识不变，并且还可以提供节点及其连接状态的完

整清单。组织机构应确保编排平台的设计具有足够的弹性，让单个节点的入侵不会损害集群的整体安全。被入侵的节点必须能够从集群中隔离并移除，而不会破坏或降低整个集群的运行情况。最后，组织机构应选择能够让集群成员之间相互验证网络连接，并对集群内流量进行端到端加密的编排工具。由于容器的可移植性，许多时候会对组织机构不直接控制的网络进行跨网络部署，因此默认安全方式对于该场景尤为重要。

4.4. 容器应对措施

4.4.1 运行时软件中的漏洞

必须仔细监控容器运行时的漏洞，当检测到问题时必须快速修补。运行时有漏洞会让运行时所支持的所有容器以及主机本身面临潜在的重大风险。组织机构应使用工具从运行时容器中查找常见漏洞和 CVE 中披露的漏洞，升级任何有风险的实例，并确保编排工具只允许部署妥善维护的运行时。

4.4.2 容器的网络访问不受限制

组织机构应控制由容器发送的对外网络流量。至少，这些控制措施应位于网络边界，确保容器无法跨不同敏感度级别的网络（例如从托管安全数据的环境发送到互联网）发送流量，这与传

统架构所使用的模式类似。但是，容器间通信的虚拟化网络模式也带来了额外的挑战。

由于跨多个主机部署的容器通常通过虚拟、加密网络通信，传统的网络设备往往无法查看这些流量。另外，容器通常在编排工具部署时自动分配动态 IP 地址，并且随着应用的扩展变化和负载均衡，这些地址也会不断变化。因此，理想情况下，组织机构应综合使用现有网络级设备和更多能够感知应用的网络过滤设备。能够感知应用的工具不仅应该能够看到容器间的数据流，还可以动态地基于容器中运行的应用的具体特点，创建规则，过滤流量。

由于容器化应用的伸缩和变更频率，以及其多变的网络拓扑，这种动态规则管理非常关键。

具体而言，可感知应用的工具应提供以下功能：

- 自动确定正确的容器网络面，包括入栈端口和进程-端口绑定；
- 检测容器之间以及与其它网络实体之间的通信流，无论是“在线”流量还是封装流量；
- 检测网络异常，如组织机构网络中的异常通信流、端口扫描或对外访问潜在危险地址。

4.4.3 容器运行时配置不安全

组织机构应自动遵守容器运行时的配置标准。文档化的技术实施指南，例如互联网安全中心 Docker 安全基准，详细介绍了有关选项和建议设置，但该指南的实施有赖于自动化工具。组织机构可以使用各类工具“扫描”和评估在某个时间点上组织机构的合规性，但这种方法不具有扩展性。相反，组织机构应该使用工具或进程，对整个环境中的配置设置进行持续评估，并主动执行这些设置。

此外，如 SELinux 和 AppArmor 等强制访问控制（MAC）技术为在 Linux 操作系统上运行的容器提供强化控制和隔离。例如，强制访问控制技术可以用来提供额外的隔离和保障，保障容器只能访问特定的文件路径、进程和网络套接字，进一步限制了遭到入侵的容器影响主机或其他容器的能力。强制访问控制技术提供了主机操作系统层保护，确保容器化应用只能访问特定的文件、路径和进程。鼓励组织机构在部署容器时，使用主机操作系统提供的强制访问控制技术。

安全计算（seccomp）[7]配置文件是另一种可以用于限制容器运行时所分配的系统级能力的机制。像 Docker 这样的常见容器运

[7] 欲了解 seccomp 的更多信息，请参见 https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt。

行时包含默认的 seccomp 配置文件，它会丢弃不安全、容器运行通常不需要的系统调用。此外，可以创建用户定义的配置文件并将其传递给容器运行时，以进一步限制其功能。至少，组织应确保容器运行时使用其运行时提供的默认配置文件，并应考虑为高风险应用使用额外的配置文件。

4.4.4 应用漏洞

鉴于之前讨论的技术架构和操作实践的差异，现有基于主机的入侵检测过程和工具往往无法检测和防止容器内的攻击。

组织机构应实施其它工具，这些工具应具有容器感知能力，并设计用于按照在容器中运行常见的扩展率和变更频率运行。这些工具应该能够使用行为学习和安全构建配置文件，自动为容器化应用提供配置文件，最大限度地减少人机交互。这些配置文件应该能够防止和检测运行时异常，包括以下事件：

- 无效或意外进程执行
- 系统调用错误或意外调用系统
- 更改受保护的配置文件和二进制文件
- 写入异常位置和文件类型
- 创建异常的网络侦听器
- 流量发送到异常网络目的地

- 恶意软件的存储或执行

容器也应该以只读模式运行其根文件系统。这样就将写操作隔离到特定目录，从而可以更容易地用上述工具进行监控。此外，使用只读文件系统让容器在遭到入侵时更具弹性，因为任何篡改都会被隔离到这些特定的位置，并且可以很容易地与应用的其余部分隔离开来。

4.4.5 流氓容器

组织机构应为开发、测试、生产和其它场景搭建不同环境，每个场景都采用具体的控制措施，从而为容器的部署和管理活动提供基于角色的访问控制。所有容器的创建都应与具体用户身份标识相关联，并进行日志记录，提供活动的明确审计跟踪。此外，鼓励组织机构在允许运行镜像之前，使用安全工具强制执行漏洞管理以及合规的基线要求。

4.5. 主机操作系统应对措施

4.5.1 攻击面大

对于使用容器专用操作系统的组织机构来说，开始时威胁通常会比较小，因为操作系统是专门为托管容器而设计的，并且其它服务和功能都禁用了。此外，由于这些优化操作系统是专门用

于托管容器的，因此，这些系统通常使用只读文件系统，并在默认情况下还采用了其它加固方法。在可能的情况下，组织机构应使用这些极简版操作系统来减少攻击面，缓解与通用操作系统相关的典型风险，并减少相关的加固措施。

无法使用容器专用操作系统的组织机构应遵循 NIST SP800-123《通用服务器安全指南》中的指导，尽可能减少其主机的攻击面。例如，运行容器的主机应该只运行容器，而不运行容器以外的其它应用，如 Web 服务器或数据库等。主机操作系统不应运行不必要的系统服务（如打印后台处理服务），这会增加其攻击面增大，补丁需求增多。最后，应连续扫描主机漏洞并快速实施更新，不仅对容器运行时而言，而且对底层组件也应如此，如容器安全和隔离操作所依赖的内核。

4.5.2 共享内核

除了将容器工作负载按敏感度级别分组到主机上之外，组织机构也不应在同一主机实例上混合使用容器化和非容器化工作负载。例如，如果主机运行的是 Web 服务器容器，则不应直接在主机操作系统中运行 Web 服务器（或任何其它应用），将其作为正常安装的组件。将容器化工作负载隔离到容器专用主机上，可以更简单、更安全地应用针对容器保护优化的防范措施和防御策

略。

4.5.3 主机操作系统组件漏洞

组织机构应实施管理实践和工具，来验证为操作系统提供基本管理和功能所用的组件版本。尽管容器专用操作系统的组件数量比通用操作系统要小得多，但仍然会存在漏洞，仍然需要修补。组织机构应使用操作系统供应商或其它可信组织提供的工具，对操作系统中使用的所有软件组件进行定期检查并实施更新。操作系统不仅要更新到最新的安全版本，而且还要将供应商建议的最新组件更新到最新版本。这对于内核和容器运行时组件尤为重要，因为较新版本的组件通常会增加额外的安全保护和功能，而不仅仅是纠正漏洞。一些组织机构可能选择简单地重新部署含有必要更新的新操作系统实例，而不是更新现有系统。虽然这通常需要更复杂的运维操作，但这种做法也是有效的。

主机操作系统应以不可变的方式运行，主机上不会持久地存储任何独特数据或状态，并且主机不会提供应用级别的依赖项。相反，所有应用组件和依赖项都应打包并部署在容器中。这就确保了主机将以几乎无状态的方式运行，并大大减少了攻击面。此外，这还提供了一种更可信的方式来识别异常情况和配置漂移。

4.5.4 用户访问权限不当

尽管大多数容器部署依赖于编排工具在主机之间分发作业，但组织机构仍应确保，对操作系统的所有身份验证进行审核，对登录异常进行监视，并记录任何通过提权来执行特权操作的情况。这样就可以识别异常的访问模式，例如个人直接登录到主机上并运行特权命令来操控容器。

4.5.5 篡改主机文件系统

确保使用所需的最小文件系统权限来运行容器。容器很少会在主机上装载本地文件系统。相反，容器需要保存到磁盘的任何文件更改都应通过专门为此目的分配的存储卷进行。在任何情况下，容器都不能在主机文件系统上装载敏感目录，尤其是那些包含操作系统配置设置的文件夹。

组织机构应使用工具监视容器所装载的目录，并防止部署违反这些策略的容器。

4.6. 硬件应对措施

正如 NIST SP800-164 所述，基于软件的安全通常被攻破。NIST 在 SP800-147、800-155 和 800-164 中确定了可信计算的要求。对 NIST 来说，“可信”意味着平台的行为符合预期：软件

清单准确，配置设置和安全控制落实到位，并按规定方式运行等等。“可信”还意味着已知晓，没有未经授权的人员篡改主机上的软件或其配置。硬件的可信根不是容器独有的概念，但容器管理和安全工具可验证容器技术架构的其它内容，确保容器在安全环境中运行。

当前，提供可信计算的可用方法是：

1. 在使用度量根（RTM）之前，先检测固件、软件和配置数据。

2. 将这些度量结果存储在硬件信任根中，如可信平台模块（TPM）。

3. 验证当前度量值与预期度量值相匹配。如果匹配，可以证明平台的表现符合预期，可以得到信任。

启用 TPM 的设备可以在启动过程中检查计算机的完整性，从而在引导和安全启动过程中能够在硬件中执行保护和检测机制。同样的信任和完整性保障可以扩展到操作系统和引导加载程序之外，延伸至容器运行时和应用。请注意，虽然正在制定标准，实现云服务用户对硬件信任的验证，但并非所有的云都向其客户开放此功能。在未提供技术验证的情况下，组织机构应将硬件信任要求纳入到与云提供商的服务协议中来。

系统的复杂性日益增加，加之当今威胁的根深蒂固，这意味着安全应该从硬件和固件开始，扩展到所有容器技术组件。这会形成分布式可信计算模型，提供构建、运行、编排和管理容器的最可信和最安全的方法。

可信计算模型应该从度量、安全启动开始，提供了一个经过验证的系统平台，并构建了一个植根于硬件并扩展到引导加载程序、操作系统内核和操作系统组件的信任链，实现对启动机制、系统镜像、容器运行时和容器镜像的加密验证。对于容器技术，这些技术目前适用于硬件、虚拟机管理器和主机操作系统层，前期开展的工作时将这些方法运用到容器专用组件中来。

在编写本文档时，NIST 正在与业界伙伴协作，基于商业成品构建一个参考架构，展现容器环境的可信计算模型。[8]

5. 容器威胁场景案例

为了说明第 4 节中建议的应对措施的有效性，请考虑以下容器威胁场景案例。

[8] 欲了解此前 NIST 在该领域中工作内容的相关信息，请参阅 NIST IR7904《云中的可信定位：概念验证实现》。

5.1. 利用镜像中的漏洞

容器化环境中最常见的一个威胁是容器内的软件中存在应用级漏洞。例如，组织机构可以基于通用的 Web 应用创建镜像。如果该应用有漏洞，漏洞会被用于破坏容器内的应用。一旦攻破，攻击者可能能够映射到环境中的其它系统，试图在被入侵的容器内提权，或滥用容器攻击其它系统（如充当文件下载器或指挥和控制终端）。

采用下述推荐应对措施的组织机构会具备多层纵深防御措施来应对这种威胁：

1. 在部署过程中尽早检测到含有漏洞的镜像，并采取适当的控制措施，防止部署有漏洞的镜像，可以防止将该漏洞引入生产环境中。

2. 容器感知网络监控和过滤功能，可以在容器试图映射到其它系统的过程中，检测到与其它容器的异常连接。

3. 容器感知进程的监控和恶意软件检测，会检测到错误或异常的恶意进程及其引入环境中的数据。

5.2. 利用容器运行时

虽然并不常发生，但如果容器运行时受到攻击，则攻击者可以利用对此容器的访问权限攻击主机上的所有容器，甚至攻击主机本身。

针对此威胁情景的相关缓解措施包括：

1. 使用强制访问控制功能可以提供额外的壁垒，确保进程和文件系统活动仍然隔离在既定边界范围之内。

2. 工作负载的分隔确保了入侵范围将仅限于共享主机的、相同敏感度级别的应用。例如，对于仅运行 Web 应用的主机，其运行时被入侵不会影响其它运行财务容器的主机上的运行时。

3. 能够报告运行时漏洞状态、防止将镜像部署到有漏洞的运行时安全工具中，可以防止工作负载在容器中运行。

5.3. 运行有毒镜像

由于镜像很容易从公共位置获取，通常是未知来源，攻击者可能会在已知目标会使用的镜像中嵌入恶意软件。

例如，如攻击者确定，某个目标在讨论区积极讨论一个特定项目，并且会使用该项目网站提供的镜像，则攻击者可能会尽力制作恶意版本的镜像，以便用于攻击。

相关的缓解措施包括：

1. 确保只允许将经过审核、测试、验证和数字签名的镜像上传到组织机构的镜像仓库。

2. 确保只允许运行可信镜像，防止使用外部、未审核来源的

镜像。

3. 自动扫描镜像是否存在漏洞和恶意软件，检测嵌入镜像的恶意代码，例如 rootkit。

4. 部署运行时控制措施，限制容器滥用资源、提权和运行可执行文件的能力。

5. 使用容器级网络分段来限制中毒镜像的“爆破半径”。

6. 确认容器运行时按照最小授权和最小访问原则运行。

7. 构建容器运行时的威胁特征。这包括但不限于进程、网络调用和文件系统变更。

8. 在运行前通过哈希值和数字签名验证镜像的完整性。

9. 根据可接受的漏洞严重级别的确定规则，限制镜像的运行。例如，防止带有中等或更高 CVSS 级别漏洞的镜像运行。

6. 容器技术生命周期安全考虑因素

安装、配置和部署容器技术之前，仔细规划是极为重要的。这有助于确保容器环境尽可能安全，并符合所有相关的组织策略、外部法规和其它要求。

容器技术的规划和实施建议与虚拟化解决方案非常相似。NIST SP800-125 第 5 节中包含了一整套针对虚拟化解决方案的建议。本节不再重复所有建议，而是为读者指出该文档，并声明，除了下面列出的例外情况外，组织机构应在容器技术场景中应用 NIST SP800-125 第 5 节的所有建议。例如，制定容器技术安全策

略，而不是制定虚拟化安全策略。

本文档的本节内容列出了 NIST SP800-125 第 5 节建议中的例外情况和新增内容，并按规划和实施生命周期中的相应阶段进行了分组。

6.1. 起始阶段

组织机构应考虑其它安全策略可能对容器造成的影响，并根据需要，调整策略，将容器纳入考虑范围。例如，考虑到容器的特殊需求，事件响应（尤其是取证）和漏洞管理策略可能需要调整。

采用容器技术可能会扰乱组织机构中现有的文化和软件开发方法。为了充分利用容器所能带来的效益，应调整组织机构的流程，支持以新方式开发、运行和支持应用。传统的开发方法、补丁技术、以及系统升级过程可能无法直接适用于容器化环境，组织机构中的员工愿意适应一种新模式，这一点也很重要。新的流程可以考虑并解决因技术转变而产生的任何潜在文化冲击。可以向参与软件开发生命周期的任何人提供教育和培训，让他们能够适应构建、传输和运行应用的新方法。

6.2. 规划设计阶段

在规划和设计阶段，容器特定的主要考虑因素是取证。由于容器主要基于操作系统中已存在的组件构建，因此，在容器化环境中进行取证的工具和技术主要是由现有做法演变而来。并且容器和镜像的不可变性实际上可以提高取证能力，因为在事件发生过程中，镜像应该做什么和实际发生的事情之间的界限更加一目了然。例如，如果启动运行 Web 服务器的容器突然启动了邮件转发，则很明显，新进程并不是原来创建容器的镜像中的一部分。在传统的平台上，操作系统和应用之间隔离效果较差，进行这种区分可能更困难。

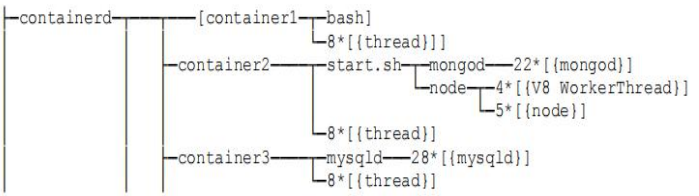
熟悉进程、内存和磁盘事件响应活动的组织机构在处理容器时会发现它们大体相似。不过，还是有些区别要注意。

容器通常使用在主机 OS 中虚拟化的分层文件系统。直接检查主机上的路径通常仅会显示这些层的外边界，而不显示其中的文件和数据。因此，在对容器化环境中的事件作出响应时，用户应该确定使用的特定存储提供商，并了解如何正确地离线检查其内容。

容器通常使用虚拟化覆盖网络相互连接。这些覆盖网络经常使用封装和加密，让流量可以安全地通过现有网络进行路由通信。但是，这意味着在调查容器网络上的事件时，尤其是在进行

实时数据包分析时，所使用的工具必须了解这些虚拟化网络，并了解如何从其中提取嵌入式 IP 帧，以便使用现有工具进行分析。

容器内的进程和内存活动很大程度上类似于传统应用中所观察到的活动，只是父进程不同。例如，容器运行时可能会以嵌套的方式在容器内生成所有进程，在该进程中，运行时是顶级进程，每个容器包含一级子进程，每个进程包含二级进程。例如：



6.3. 实施阶段

在设计完容器技术之后，下一步是对设计原型进行测试，然后再将解决方案投入生产。请注意，容器技术没有 VM 技术所具有的内省能力。

NIST SP800-125 列举了在生产部署之前应评估的虚拟化技术的几方面内容，包括身份验证、连接和组网、应用功能、管理、性能和技术本身的安全性。此外，评估容器技术的隔离能力也很重要。确保容器中的进程可以访问允许其访问的所有资源，但无

法查看或访问任何其它资源。

实施阶段可能需要新的安全工具，专用于容器和云原生应用，并且对其运行情况具有可视性，这是传统工具所缺乏的。最后，部署还可能包括更改现有安全控制和技术的配置，例如安全事件日志记录、网络管理、代码存储库和身份验证服务器，以便直接使用容器，并集成容器新安全工具。

当原型评估完成，且容器技术准备好投入生产使用时，最初，容器应该用于少量应用。出现的问题可能会影响多个应用，因此，及早发现这些问题很有用，这样就可以先解决这些问题，然后再进行进一步部署。分阶段部署还为开发人员和 IT 人员（如系统管理员、服务台）接受有关其使用方法和支持的培训提供了时间。

6.4. 运维阶段

运维流程对于维护容器技术的安全性尤为重要，因此，应定期进行运维，包括更新所有镜像、并将更新的镜像分发给容器，代替原有镜像。其它安全最佳实践，如进行漏洞管理、更新其它支持层（如主机和编排工具），也是持续运维的关键任务。容器安全和监控工具也同样应该与现有的 SIEM 等工具相结合，以确保容器相关事件采用为确保环境中其它内容的安全而提供的相同

工具和流程。

在容器化环境中发生安全事件时，组织机构应准备好用针对容器的独特特性而优化的流程和工具作出响应。NIST SP800-61《计算机安全事件处理指南》中概述的核心指南也非常适用于容器环境。然而，采用容器的组织机构应确保它们加强了针对容器安全的一些独特方面的响应措施。

- 因为容器化的应用可能不是由传统运行团队负责，而是由其它团队负责，所以，组织机构应确保，无论是哪种团队负责容器运行，都要纳入事件响应计划，并了解其在响应计划中的作用。

- 正如本文档中所讨论的，容器管理的短暂性和自动化性质可能与组织机构传统上使用的资产管理策略和工具不相符。事件响应团队必须能够熟知容器的作用、所有者和敏感度级别，并能够将这些数据集成到他们的响应过程中来。

- 应确定对容器相关的事件作出响应的明确流程。例如，如果某个特定镜像正在被利用，但有数百个容器在使用该镜像，则响应团队可能需要关闭所有这些容器，才能停止遭受攻击。尽管长期以来，单个漏洞会导致许多系统产生问题，但对于容器来说，响应可能需要广泛地重建和重新部署新镜像，而不是对现有

系统安装补丁。在响应方面的这种变化可能涉及不同的团队，并需要获得审批，应该提前熟悉和演练。

● 如前所述，日志记录和其它取证数据在容器化环境中可能以不同的方式存储。事件响应团队应该熟悉收集数据所需的不同工具和技术，并有专门针对这些环境的记录过程。

6.5. 处置阶段

根据应用的需求，能够自动部署和销毁容器，能够提高系统效率，但也会带来记录保留、取证和事件数据需求方面的一些挑战。组织机构应确保建立适当的机制来满足其数据保留策略。应该解决的问题示例包括如何销毁容器和镜像、在处置之前应从容器中提取哪些数据、以及如何进行数据提取、如何撤销或删除容器所使用的加密密钥等。

组织机构开发的任何处置计划中应涵盖支持容器化环境的数据存储库以及介质。

7. 青藤基于宿主机 Agent 的全生命周期容器解决方案

针对各类容器安全问题，目前，国际市场上涌现了一批容器

安全产品安全厂商，如 Neuvector、Twistlock、StackRox、Aqua 等等，国内自研容器安全产品的厂商则有青藤云安全。从容器安全产品的技术方案上来看，目前大部分的容器安全厂商均使用了平行容器的方式对宿主机上的容器进行安全防护，而青藤云安全则采取了基于容器全生命周期的解决方案进行安全防护。这两种技术方式有何不同，会产生怎样不同的安全防护效果呢？

平行容器技术方案：利用容器的隔离性和良好的资源控制能力，在容器的宿主机中启动一个容器，该容器通过挂载宿主机的所有文件系统，而后在容器内部对这些文件系统进行实时监控和处理响应，以实现对容器进行防护的作用。

基于宿主机 Agent 解决方案：即基于全生命周期的主机防护能力，监控宿主机上容器相关的文件、进程、系统调用等信息，增加对于容器的清点、监控、防护能力，从而通过一种安全解决方案，即可实现宿主机安全、容器安全两种防护效果。国内新一代主机安全厂商青藤云安全是此类方案的践行者。

以上两种方案的示意图如下：

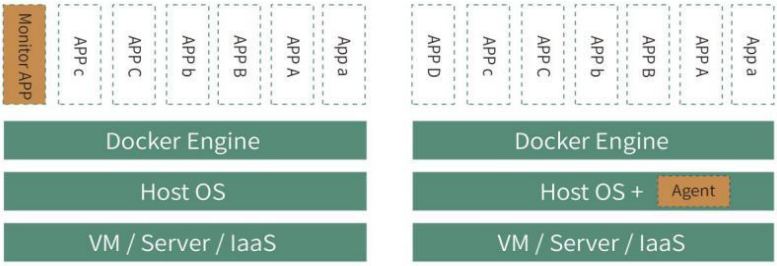


图 4：平行容器技术方案与基于全生命周期的容器安全解决方案

根据青藤容器安全解决方案，容器的安全能力应覆盖容器的整个生命周期，即构建、分发、运行三个阶段。下图展示了容器环境的安全工作流程。对于每个阶段，青藤容器安全解决方案都提出了详细的解决方案。

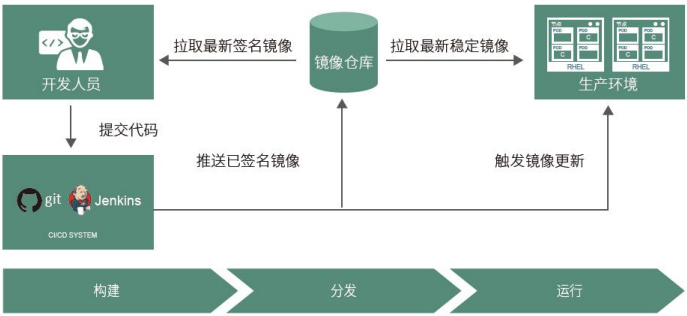


图 5：容器环境安全工作流程

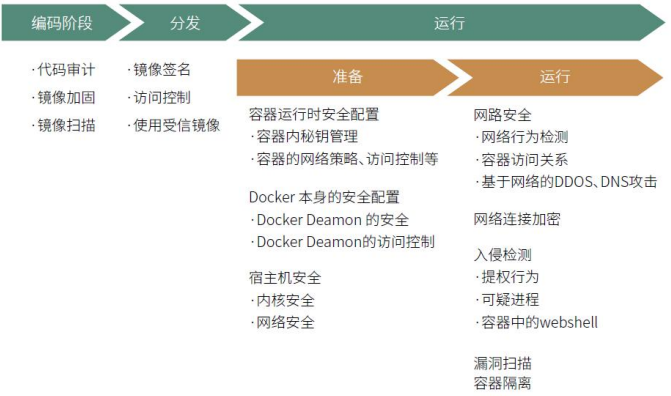


图 6：青藤全生命周期容器安全解决方案

7.1. 构建阶段

持续性容器安全的第一步，是构建安全的应用。

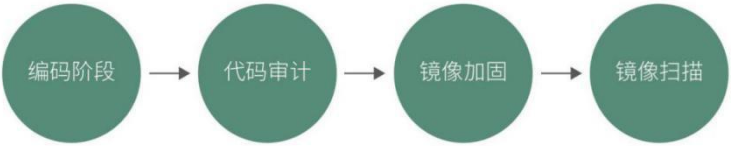


图 7：构建阶段的容器安全

首先，在编码阶段，要求工程师具备一定的安全知识，从代码源头上减少可被攻击的风险；

其次，进行代码集成和测试之前，利用代码审计工具发现代码中潜在的漏洞；

再次，构建镜像时，通过删除不必要的库和安装包，对镜像进行精简、加固，尽量减少攻击面；

最后，在镜像投入使用之前，对镜像进行漏洞扫描可及时发现潜在的风险；同时，应该对镜像仓库中的镜像进行周期性扫描，以便能够扫描出镜像中近期新爆发的漏洞。

7.2. 分发阶段

在镜像分发阶段，进行适当的访问控制和镜像校验是很有必要的。

第一，要防止镜像在传输过程中被篡改。青藤容器安全解决方案采用 Docker 的 Content Trust 功能，对镜像添加多重签名，在拉取镜像时进行校验，确保镜像没有被篡改过。

第二，要对镜像进行访问控制。对镜像仓库、编排工具、其他开发的访问控制应集成到如 LDAP 这类统一认证平台。

第三，使用受信任的镜像。不要随意使用未进行漏洞扫描、未加固的镜像，这些很可能是恶意镜像或存在漏洞的镜像。

7.3. 运行阶段

7.3.1 运行准备阶段

容器在生产环境运行之前，应确保容器的运行时环境是安全的。

● 要确保容器运行时安全配置。例如，对容器内秘钥进行管理；使用编排工具配置好容器的网络策略、控制访问，确保容器处于运行状态。

● Docker 本身的安全配置，包括 Docker Deamon 的安全和 Docker Deamon 的访问控制。

● 宿主机安全，包括内核安全和网络安全。

7.3.2 生产环境阶段

由于容器带来了东西向流程的安全问题、新的入侵方式，因此，对生产环境进行持续性的安全防护和检测必不可少。对此，青藤容器安全解决方案主要是从以下几个方面着手：

网络安全——对网络行为进行检测，实现容器连接关系的可视化，轻松识别基于网络的攻击，如 DDOS 攻击、DNS 攻击等。

网络连接加密——对网络连接进行加密，可以防止在网络传

输中窃取机密信息或敏感数据。

入侵检测——对主机、容器的提权行为进行检测、对可疑进程进行检测、并检测容器中是否存在 webshell

运行时容器漏洞扫描——确保运行时的容器中不存在漏洞，如有使用了新的容器，也应该在第一时间进行漏洞漏扫

容器隔离——对恶意容器进行隔离

8. 总结

容器是应用构建和运行方式的重大变革。不需要因此而大幅度地创新安全最佳实践；相反，只需对已经有效建立起来的技术和原则进行细化，就能解决容器安全的最重要的问题。考虑到容器技术的特殊风险，本文档更新并扩展了通用安全建议。

本文档还讨论了确保容器安全和确保 VM 中相同应用的安全之间的一些差异。围绕这些论点对本文档中的指导意见进行汇总大有裨益。

在容器环境中，实体的数量更多，因此安全流程和工具必须能够相应地扩展。扩展不仅是指数据库中支持的对象总数，而且还意味着如何有效且自主地管理策略。许多组织机构都在为成百上千台 VM 的安全管理而不堪重负。随着以容器为中心的体系结

构成为规范，并且这些组织机构负责成千上万个容器，其安全性实践应强调实现自动化、提高效率，才能跟上步伐。

使用容器后，变化率提高，应用的更新频率从一年更新几次发展到每周、甚至每天更新几次。过去曾经可以接受的手动操作现在已无用武之地。自动化的重要性不仅体现在要处理的净实体数量方面，还体现在这些实体的变更频率方面。能够集中表述策略，并使用软件来管理整个环境中策略的执行，这一点至关重要。采用容器的组织机构应准备好管理这种频繁的变更。这可能需要采用全新的运行方法，变革组织结构。

使用容器会将安全的大部分职责转移给开发人员，因此，组织机构应该确保他们的开发人员拥有做出正确决策所需的所有信息、技能和工具。此外，应确保安全小组在整个开发周期中积极执行质量要求。组织机构若能成功度过这一过渡阶段，便能够提高漏洞响应速度，减轻以前面临的运营负担，从而尽享安全带来的效益。

安全性必须与容器本身一样可移植，因此，组织机构所采用的技术和工具应该是开放的，并在所有平台和环境中能够发挥作用。许多组织机构都是开发人员在一个环境中进行构建，在另一种环境中进行测试，然后在第三种环境中进行部署，因此，确保

评估和执行的一致性是关键所在。可移植性不仅对于环境，而且对于时间也是如此。持续集成和部署的做法打破了开发和部署周期各个阶段之间的传统壁垒，因此，组织机构需要确保在镜像仓库中创建、存储镜像，以及在容器中运行镜像的过程中采用一致的自动化安全方法。

组织机构对这些变化轻车熟路后就可以开始利用容器来实际提高其整体安全性。容器的不变性和声明性让组织机构能够开始实现更自动化、以应用为核心的安全愿景，这需要最小化人工干预，并且随着应用的变化对自身进行调整。容器让组织机构能够从被动、手动、高成本的安全模式，转变为更具伸缩性和更高效的模式，从而降低风险。

青藤官方微信

青藤云安全

客服热线：400-188-9287

地址：北京市海淀区创业路8号群英科技园1号楼5层

网址：https://qingteng.cn

电话：010-53520955