Docker 容器最佳安全实践

白皮书

（V1.0）

2018 年 5 月

※本文档是 Dosec 安全团队参考 CIS 的 docker 安全标准并结合实践经验整理，更多 docker 安全关注我们的公众号：DoSec 容器安全

目录

1.主机安全配置

1.1 为容器创建一个单独的分区

描述	所有 Docker 容器及数据和元数据都存储在/var/lib/docker 目录下。 默认情况下，/var/lib/docker 将根据可用性挂载在/ 或 /var 分区下。
安全出发点	Docker 依赖于/var/lib/docker 作为默认目录，其存储所有 Docker 相关文件，包括镜像文件。该目录可能会被恶意的写满，导致 Docker、甚至主机可能无法使用。因此，建议为存储 Docker 文件创建一个单独的分区（逻辑卷）。
审计方法	grep /var/lib/docker /etc/fstab
结果判定	应该返回/var/lib/docker 挂载点的分区详细信息。
修复措施	新安装 docker 时，为/var/lib/docker 挂载点创建一个单独的分区。对于先前安装的系统，请使用逻辑卷管理器（LVM）创建分区。
影响	None
默认值	默认情况下，/var/lib/docker 将根据可用性挂载在/ 或 /var 分区下。
参考文献	*http://www.projectatomic.io/docs/docker-storage-recommendation*

1.2 加固容器宿主机

描述	容器在 Linux 主机上运行。 容器主机可以运行一个或多个容器。 加强主机以缓解主机安全配置错误是非常重要的
安全出发点	应该遵循基础架构安全最佳实践并加强宿主机操作系统。 保证主机系统的安全可以确保主机漏洞得控制。 若不加固主机系统可能导致安全问题。
审计方法	确保遵守主机的安全规范。 询问系统管理员当前主机系统符合哪个安全标准。 确保主机系统实际符合主机制定的安全规范。
结果判定	查看主机加固记录
修复措施	使用主机安全标准加固主机
影响	None
默认值	默认情况下，主机只有默认设置，没有加固。
参考文献	*1.https://docs.docker.com/engine/security/security/* *2. https://learn.cisecurity.org/benchmarks* *3.https://docs.docker.com/engine/security/security/#other-kernel-security-features* *4. https://grsecurity.net/* *5. https://en.wikibooks.org/wiki/Grsecurity* *6. https://pax.grsecurity.net/* *7. http://en.wikipedia.org/wiki/PaX*

1.3 更新 docker 到最新版本

描述	Docker 软件频繁发布更新，旧版本可能存在安全漏洞。
安全出发点	通过及时了解 Docker 更新，Docker 软件中的漏洞可以得到修复。 攻击者可能会尝试获得访问权限或提升权限时利用已知的漏洞。 不安装常规的 Docker 更新可能会让现有的 Docker 软件受到攻击。 可能会导致提升权限，未经授权的访问或其他安全漏洞。 所以需要跟踪新版本并根据需要进行更新。
审计方法	执行以下命令并验证 Docker 版本是否为必要的最新版本。 Docker version
结果判定	和最新版本进行比对，查看是否为最新
修复措施	跟踪 Docker 发布并根据需要进行更新。
影响	对 Docker 版本更新进行风险评估，了解它们可能会如何影响 Docker 操作。 请注意，有些使用 Docker 的第三方产品可能需要支持较老版本的 Docker。
默认值	不适用
参考文献	*1. https://docs.docker.com/engine/installation/* *2. https://github.com/moby/moby/releases/latest* *3. https://github.com/docker/docker-ce/releases/latest*

1.4 只有受信任的用户才能控制 docker 守护进程

描述	Docker 守护进程需要'root'权限。 对于添加到 “docker” 组的用户为提供了完整的 “root” 访问权限。
安全出发点	Docker 允许在 Docker 主机和访客容器之间共享目录，而不会限制容器的访问权限。 这意味着可以启动容器并将主机上的 / 目录映射到容器。 容器将能够不受任何限制地更改您的主机文件系统。 简而言之，这意味着您只需作为 “docker” 组的成员即可获得较高的权限，然后在主机上启动具有映射 / 目录的容器。
审计方法	在 docker 主机上执行以下命令，并确保只有可信用户是 docker 组的成员。 getent group docker
结果判定	判断是否必须要加入 docker 组的用户
修复措施	从'docker'组中删除任何不受信任的用户。 另外，请勿在主机上创建敏感目录到容器卷的映射。
影响	作为普通用户构建和执行容器的权限将受到限制
默认值	不适用
参考文献	*1.https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface* *2.https://www.andreas-jung.com/contents/on-docker-security-docker-group-considered-harmful* *3.http://www.projectatomic.io/blog/2015/08/why-we-dont-let-non-root-users-run-docker-in-centos-fedora-or-rhel/*

1.5 审计 docker 守护进程

描述	审计所有活动的 Docker 守护进程
安全出发点	除了审核常规的 Linux 文件系统和系统调用外，还要审计 Docker 守护进程。 Docker 守护进程以'root'特权运行。 因此有必要审核其活动和使用情况。
审计方法	验证是否存在 Docker 守护程序的审核规则。 例如，执行以下命令： auditctl -l	grep / usr/bin/docker
结果判定	应该列出 Docker 守护程序的规则
修复措施	为 Docker 守护进程添加一条规则，例如， 在/etc/audit/audit.rules 文件中将该行添加到以下行中或者运行命令直接添加：auditctl -w / usr / bin / docker -k docker 然后，重新启动审计守护进程。 例如 server auditd restart
影响	审计生成相当大的日志文件。 确保定期归档它们。 另外，创建一个单独的审计分区以避免写满根文件系统。
默认值	默认，Docker 守护进程没有审计
参考文献	*1.https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html*

1.6 审计 docker 文件和目录-/var/lib/docker

描述	审计以下目录： /var/lib/docker.
安全出发点	除了审核常规的 Linux 文件系统和系统调用外，还要审核所有 Docker 相关的文件和目录。 Docker 守护进程以'root'特权运行。 它的运行取决于一些关键文件和目录。 /var/lib/docker 就是这样一个目录。 它包含有关 docker 的所有信息。
审计方法	验证是否存在与/var/lib/docker 目录相对应的审计规则。 例如，执行以下命令：auditctl -l | grep /var/lib/docker
结果判定	这应该列出/var/lib/docker 目录的规则。
修复措施	为/var/lib/docker 目录添加一条规则。 例如，将以下行添加到 /etc/audit/audit.rules 文件中： -w/var/lib/docker -k docker 然后，重新启动审计守护进程。 例如 service auditd restart
影响	审计生成相当大的日志文件。 确保定期归档它们。 另外，创建一个单独的审计分区以避写满根文件系统。
默认值	默认情况下，Docker 相关的文件和目录不被审计
参考文献	*1.https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html*

1.7 审计 docker 文件和目录-/etc/docker

描述	审计以下目录 /etc/docker
安全出发点	除了审核常规的 Linux 文件系统和系统调用外，还要审核所有 Docker 相关的文件和目录。 Docker 守护进程以'root'特权运行。 它的行为取决于一些关键文件和目录。 /etc/docker 就是这样一个目录。 它包含用于 Docker 守护进程和 Docker 客户端之间的 TLS 通信的各种证书和密钥。
审计方法	验证是否存在与/ etc / docker 目录相对应的审计规则。 例如，执行以下命令：auditctl -l | grep /etc/docker
结果判定	这应该列出/ etc / docker 目录的规则。
修复措施	为/etc/docker 目录添加一条规则。 例如， 在/etc/audit/audit.rules 文件中添加如下所示的行 -w / etc / docker -k docker 然后，重新启动审计守护进程。 例如 servise auditd restart
影响	审计生成相当大的日志文件。 确保定期归档它们。 另外，创建一个单独的审计分区以避免写满根文件系统
默认值	默认情况下，Docker 相关的文件和目录不被审计
参考文献	*1.https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html*

1.8 审计 docker 文件和目录-docker.service

描述	审核 docker.service（如果适用）。
安全出发点	除了正常的 Linux 文件系统和系统调用审核外，还可以审核所有与 Docker 相关的文件和目录。 Docker 守护程序以 “root” 权限运行。 它的行为取决于一些关键文件和目录。 docker.service 是一个这样的文件。 如果管理员更改了守护程序参数，则 docker.service 文件可能会出现问题。 它支持 Docker 守护进程的各种参数。
审计方法	步骤 1：找出文件位置：systemctl show -p FragmentPath docker.service 步骤 2：如果文件不存在，则此建议不适用。 如果该文件存在，请验证是否存在与该文件相对应的审核规则： 例如，执行以下命令：auditctl -l | grep docker.service
结果判定	应该根据其位置列出 docker.service 的规则。
修复措施	如果该文件存在，请为其添加规则。例如， 在/etc/audit/audit.rules 文件中添加以下行： -w /usr/lib/systemd/system/docker.service -k docker 然后，重新启动审计守护进程。 例如， servise auditd restart
影响	审核会生成相当大的日志文件。 确保定期归档。 另外，创建一个单独的审计分区，以避免填写根文件系统。
默认值	默认情况下，Docker 相关的文件和目录不会被审计。 文件 docker.service 可能在系统上不可用
参考文献	*1.https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html*

1.9 审计 docker 文件和目录-docker.socket

描述	审核 docker.socket（如果适用）。
安全出发点	除了正常的 Linux 文件系统和系统调用审核外，还可以审核所有与 Docker 相关的文件和目录。 Docker 守护程序以 “root” 权限运行。 它的行为取决于一些关键文件和目录。 docker.socket 就是一个这样的文件。 如果管理员更改了守护程序参数，则 docker.socket 文件可能会出现问题。 它支持 Docker 守护进程的各种参数。
审计方法	步骤 1：找出文件位置：systemctl show -p FragmentPath docker.socket 步骤 2：如果文件不存在，则此建议不适用。 如果该文件存在，请验证是否存在与该文件相对应的审核规则： 例如，执行以下命令：auditctl -l | grep docker.socket
结果判定	这应该根据其位置列出 docker.socket 的规则
修复措施	如果文件存在，为其添加审计策略：. 在/etc/audit/audit.rules 文件添加一行: -w /usr/lib/systemd/system/docker.socket -k docker 然后重启审计进程： service auditd restart
影响	审核会生成相当大的日志文件。 确保定期归档。 另外，创建一个单独的审计分区，以避免填写根文件系统。
默认值	默认情况下，Docker 相关的文件和目录不被审计。 文件 docker.socket 可能在系统上不可用，在这种情况下，此建议不适用。
参考文献	*1.https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html*

1.10 审计 docker 文件和目录-/etc/default/docker

描述	审核 /etc/default/docker,（如果适用）
安全出发点	除了正常的 Linux 文件系统和系统调用审核外，还可以审核所有与 Docker 相关的文件和目录。 Docker 守护程序以 “root” 权限运行。 它的行为取决于一些关键文件和目录。 / etc / default / docker 是一个这样的文件。 它支持 Docker 守护进程的各种参数。
审计方法	验证是否存在与/ etc / default / docker 文件相对应的审计规则。 例如，执行以下命令：auditctl -l	grep / etc / default / docker
结果判定	这应该列出/ etc / default / docker 文件的规则。
修复措施	如果文件存在，为其添加审计策略：. 在/etc/audit/audit.rules 文件添加一行: -w /etc/default/docker -k docker 然后重启审计进程： service auditd restart
影响	审核会生成相当大的日志文件。 确保定期归档。 另外，创建一个单独的审计分区，以避免填写根文件系统。
默认值	默认情况下，Docker 相关的文件和目录不被审计。 文件/ etc / default / docker 可能不可用。 在这种情况下，此建议不适用。
参考文献	*1.https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html*

1.11 审计 docker 文件和目录-/etc/docker/daemon.json

描述	审计 /etc/docker/daemon.json，如适用的话
安全出发点	除了正常的 Linux 文件系统和系统调用审核外，还可以审核所有与 Docker 相关的文件和目录。 Docker 守护程序以 “root” 权限运行。 它的行为取决于一些关键文件和目录。 /etc/docker/daemon.json 是一个这样的文件。 它支持 Docker 守护进程的各种参数。
审计方法	验证是否存在与/etc/docker/daemon.json 文件相对应的审计规则。 例如，执行以下命令：auditctl -l | grep /etc/docker/daemon.json
结果判定	这应该列出/etc/docker/daemon.json 文件的规则。
修复措施	添加 /etc/docker/daemon.json 文件的规则 例如， 在 /etc/audit/audit.rules 文件中添加如下行： -w /etc/docker/daemon.json -k docker 然后，重新启动审计守护程序。 例如 service auditd restart
影响	审核会生成相当大的日志文件。 确保定期归档。 另外，创建一个单独的审计分区，以避免填写根文件系统。
默认值	默认情况下，Docker 相关的文件和目录不被审计。. 文件 /etc/docker/daemon.json 可能在系统上可用。 在这种情况下，此建议不适用。
参考文献	*1.https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chapsystem_auditing.html* *2.https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file*

1.12 审计 docker 文件和目录-/usr/bin/docker-containerd

描述	审计 /usr/bin/docker-containerd，如果适用的话
安全出发点	除了正常的 Linux 文件系统和系统调用审核外，还可以审核所有与 Docker 相关的文件和目录。 Docker 守护程序以 “root” 权限运行。它的行为取决于一些关键文件和目录。 /usr/bin/docker-containerd 就是这样的文件. Docker 现在依赖于 containerd 和 runC 来生成容器。
审计方法	验证是否存在与/ usr / bin / docker-containerd 文件相对应的审计规则。 例如，执行以下命令：auditctl -l	grep / usr / bin / docker-containerd
结果判定	这应该列出/ usr / bin / docker-containerd 文件的规则。
修复措施	添加 /usr/bin/docker-containerd 文件的规则 例如， 在 /etc/audit/audit.rules 文件中添加如下行：: -w /usr/bin/docker-containerd -k docker 然后，重新启动审计守护程序。 例如 service auditd restart
影响	审核会生成相当大的日志文件。 确保定期归档。 另外，创建一个单独的审计分区，以避免填写根文件系统。
默认值	默认情况下，Docker 相关的文件和目录不被审计。. 文件 /usr/bin/dockercontainerd 可能在系统上可用。 在这种情况下，此建议不适用。
参考文献	*1.https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html* *https:/github.com/docker/docker/pull/20662* *https:/containerd.tools/*

1.13 审计 docker 文件和目录-/usr/bin/docker-runc

描述	审计 /usr/bin/docker-runc，如果适用的话.
安全出发点	除了正常的 Linux 文件系统和系统调用审核外，还可以审核所有与 Docker 相关的文件和目录。 Docker 守护程序以 “root” 权限运行。它的行为取决于一些关键文件和目录。 /usr/bin/docker-runc 是一个这样的文件。 Docker 现在依赖于 containerd 和 runC 来生成容器。
审计方法	验证是否存在与/usr/bin/docker-runc 文件相对应的审核规则。 例如，执行以下命令： auditctl -l | grep /usr/bin/docker-runc
结果判定	这应该列出/usr/bin/docker-runc 文件的规则。
修复措施	添加/usr/bin/docker-runc 文件的规则。例如， 在/etc/audit/audit.rules 文件中添加如下行： -w /usr/bin/docker-runc -k docker 然后，重新启动审计守护程序。 例如 service auditd restart
影响	审核会生成相当大的日志文件。 确保定期归档。 另外，创建一个单独的审计分区，以避免填写根文件系统。
默认值	默认情况下，Docker 相关的文件和目录不被审计。 文件/ usr / bin / dockerrunc 可能在系统上可用。 在这种情况下，此建议不适用。
参考文献	*1.https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html* *2.https://github.com/docker/docker/pull/20662* *3.https://containerd.tools/* *4.https://github.com/opencontainers/runc*

2.docker 守护进程配置

2.1 限制默认网桥上容器之间的网络流量

描述	默认情况下，默认网桥上同一主机上的容器之间允许所有网络通信。 如果不需要，限制所有的容器间通信。 将需要通信的特定容器链接在一起。 或者创建自定义网络，并只加入需要与该自定义网络通信的容器。
安全出发点	默认情况下，默认网桥上同一主机上的所有容器之间启用不受限制的网络通信。 因此，每个容器都有可能读取同一主机上容器网络上的所有数据包。 这可能会导致意外和不必要的信息泄露给其他容器。 因此，限制默认网桥上的容器间通信。
审计方法	运行以下命令并确认默认网桥已被配置为限制集装箱间通信。 docker network ls --quiet | xargs docker network inspect -- format'{{.Name}}：{{.Options}}'
结果判定	它应该返回默认网桥的 com.docker.network.bridge.enable_icc：false。
修复措施	在守护进程模式下运行 docker 并传递--icc = false 作为参数。 例如，dockerd --icc = false 或者可以遵循 Docker 文档并创建自定义网络，并只加入需要与该自定义网络通信的容器。 --icc 参数仅适用于默认泊坞桥，如果使用自定义网络，则应采用分段网络的方法。
影响	默认网桥上的容器间通信将被禁用。 如果需要在同一主机上的容器之间进行通信，则需要使用容器链接来明确定义它，或者必须定义自定义网络。
默认值	默认情况下，默认网桥上允许所有容器间通信。
参考文献	*1.https://docs.docker.com/engine/userguide/networking/* *2.https://docs.docker.com/engine/userguide/networking/default_network/container-communication/#communication-between-containers*

2.2 设置日志级别为 info

描述	将 Docker 守护程序日志级别设置为 “info” 。
安全出发点	设置适当的日志级别，配置 Docker 守护程序以记录需要查看的事件。 “info” 及以上的基准日志级别将捕获除调试日志之外的所有日志。 若无必须，不应该在'debug'日志级别运行 Docker 守护进程
审计方法	ps -ef | grep dockerd
结果判定	确保--log-level 参数不存在或存在，然后将其设置为 info。
修复措施	运行 Docker 守护进程如下： dockerd --log-level = “info”
影响	None.
默认值	默认情况下，Docker 守护程序设置为 “info” 的日志级别。
参考文献	*1.https://docs.docker.com/edge/engine/reference/commandline/dockerd/*

2.3 允许 docker 更改 IPtables

描述	Iptables 用于建立，维护和检查 Linux 内核中的 IP 包过滤规则表。 允许 Docker 守护程序更改 iptables
安全出发点	Docker 会根据用户为容器选择网络选项的方式自动对 iptables 进行必要的更改（如果允许的话）。 建议让 Docker 服务器自动更改 iptables，以避免可能妨碍容器与外界通信的网络配置错误。 另外，当每次选择运行容器或修改网络选项时，可以自动更新 iptables 的配置。
审计方法	ps -ef | grep dockerd
结果判定	确保'--iptables'参数不存在或不设置为'false'
修复措施	不要使用'--iptables = false'参数运行 Docker 守护程序。 例如，不要像下面那样启动 Docker 守护进程： dockerd --iptables = false
影响	Docker 守护进程服务需要在启动之前启用 iptables 规则。 在 Docker 守护进程操作期间任何重新启动 iptables 都可能导致丢失 docker 创建的规则。 使用 iptables-persistent 持久 iptables 规则可以帮助减轻这种影响。
默认值	默认情况下，'iptables'设置为'true'。
参考文献	*1.https://docs.docker.com/engine/userguide/networking/default_network/container-communication/* *2.https://fralef.me/docker-and-iptables.html*

2.4 不使用不安全的镜像仓库

描述	Docker 在默认情况下，私有仓库被认为是安全的
安全出发点	一个安全的镜像仓库建议使用 TLS。 在/etc/docker/certs.d/<registry-name>/目录下，将镜像仓库的 CA 证书副本放置在 Docker 主机上。 不安全的镜像仓库是没有有效的镜像仓库证书或不使用 TLS 的镜像仓库。不应该在生产环境中使用任何不安全的镜像仓库。 不安全的镜像仓库中的镜像可能会被篡改，从而导致生产系统可能受到损害。 此外，如果镜像仓库被标记为不安全，则“docker pull”， “docker push”和“docker search”命令并不能发现，那样用户可能无限期地使用不安全的镜像仓库而不会发现。
审计方法	执行以下命令，了解是否使用不安全的仓库： ps -ef | grep dockerd
结果判定	确保“--insecure-registry”参数不存在
修复措施	不要使用任何不安全的镜像仓库。 例如，不要像下面那样启动 Docker 守护进程： dockerd - insecure-registry 10.1.0.0/16
影响	None.
默认值	默认情况下，Docker 假定所有的本地镜像仓库都是安全的。
参考文献	*1.https://docs.docker.com/registry/insecure/*

2.5 不使用 aufs 存储驱动程序

描述	不要使用'aufs'作为 Docker 实例的存储驱动
安全出发点	‘aufs'存储驱动程序是较旧的存储驱动程序。 它基于 Linux 内核补丁集，不太可能合并到主要的 Linux 内核中。 'aufs'驱动会导致一些严重的内核崩溃。'aufs 在 Docker 中只是保留了历史遗留支持,现在主要使用 overlay2 和 devicemapper。 而且最重要的是，在许多使用最新 Linux 内核的发行版中，'aufs'不再被支持。
审计方法	执行以下命令并验证'aufs'不被用作存储驱动程序： docker info | grep -e "^Storage Driver:\s*aufs\s*$"
结果判定	上面的命令不应该返回任何东西
修复措施	不要刻意的使用'aufs'作为存储驱动。 例如，不要启动 Docker 守护程序，如下所示： dockerd --storage-driver aufs
影响	aufs'是允许容器共享可执行文件和共享库内存的存储驱动程序。 如果使用相同的程序或库运行数千个容器，可以选用。
默认值	默认情况下，Docker 在大多数平台上使用 “overlay2” 和 “devicemapper” 作为存储驱动程序。 默认存储驱动程序可能因操作系统而异。 应该首选操作系统最佳支持的存储驱动程序。
参考文献	*1.https://docs.docker.com/engine/userguide/storagedriver/selectadriver/#supported-backing-filesystems* *2.http://muehe.org/posts/switching-docker-from-aufs-to-devicemapper/* *3.http://jpetazzo.github.io/assets/2015-03-05-deep-dive-into-docker-storage-drivers.html#1* *4.https://docs.docker.com/engine/userguide/storagedriver/*

2.6 docker 守护进程配置 TLS 身份认证

描述	可以让 Docker 守护进程监听特定的 IP 和端口以及除默认 Unix 套接字以外的任何其他 Unix 套接字。 配置 TLS 身份验证以限制通过 IP 和端口访问 Docker 守护进程。
安全出发点	默认情况下，Docker 守护程序绑定到非联网的 Unix 套接字，并以“root”权限运行。 如果将默认的 docker 守护程序更改为绑定到 TCP 端口或任何其他 Unix 套接字，那么任何有权访问该端口或套接字的人都可以完全访问 Docker 守护程序，进而可以访问主机系统。 因此，不应该将 Docker 守护程序绑定到另一个 IP /端口或 Unix 套接字。 如果必须通过网络套接字暴露 Docker 守护程序，请为守护程序和 Docker Swarm API 配置 TLS 身份验证。
审计方法	ps -ef | grep dockerd
结果判定	确保存在以下参数： · '--tlsverify' · '--tlscacert' · '--tlscert' · '--tlskey'
修复措施	按照 Docker 文档或其他参考中提到的步骤进行操作
影响	您需要管理和保护 Docker 守护程序和 Docker 客户端的证书和密钥。
默认值	默认情况下，未配置 TLS 认证
参考文献	*1.https://docs.docker.com/engine/security/https/*

2.7 配置合适的 ulimit

描述	根据您的环境设置默认的 ulimit 选项
安全出发点	ulimit 提供对 shell 可用资源的控制。 设置系统资源控制可以防止资源耗尽带来的问题，如 fork 炸弹。 有时候合法的用户和进程也可能过度使用系统资源，导致系统资源耗尽。 为 Docker 守护程序设置默认 ulimit 将强制执行所有容器的 ulimit。 不需要单独为每个容器设置 ulimit。 但默认的 ulimit 可能在容器运行时被覆盖。因此，要控制系统资源，需要自定义默认的 ulimit。
审计方法	ps -ef | grep dockerd
结果判定	确保根据需要设置'--default-ulimit'参数
修复措施	在守护进程模式下运行 docker，并根据相应的 ulimits 传递'--default-ulimit'作为参数。例如： dockerd --default-ulimit nproc = 1024：2408 --default-ulimit nofile = 100：200
影响	如果 ulimits 未正确设置，则可能无法实现所需的资源控制，甚至可能导致系统无法使用
默认值	默认情况下，不设置 ulimit
参考文献	*1.https://docs.docker.com/edge/engine/reference/commandline/dockerd/#default-ulimits*

2.8 启用用户命名空间

描述	在 Docker 守护程序中启用用户命名空间支持，可对用户进行重新映射。该建议对镜像中没有指定用户是有帮助的。如果在容器镜像中已经定义了非 root 运行，可跳过此建议，因为该功能比较新，可能会给带来不可预测的问题。
安全出发点	Docker 守护程序中对 Linux 内核用户命名空间支持为 Docker 主机系统提供了额外的安全性。 它允许容器具有独特的用户和组 ID，这些用户和组 ID 在主机系统所使用的传统用户和组范围之外。 例如，root 用户希望有容器内的管理权限，可映射到主机系统上的非 root 的 UID 上。
审计方法	ps -p $(docker inspect --format='{{ .State.Pid }}' <CONTAINER ID>) -o pid,user 例如： ps -p $(docker inspect --format='{{ .State.Pid }}' a4723347b65b) -o pid,user
结果判定	上述命令将查找容器的 PID，然后列出与容器进程关联的主机用户。 如果容器进程以 root 身份运行，则不符合安全要求。 也运行 docker　info 以确保用户在安全选项下： docker info --format='{{.SecurityOptions}}'
修复措施	可参考 Docke 文档了解具体的配置方式。 操作可能因平台而异 例如，在 Red Hat 上，子 UID 和子 GID 映射创建不会自动工作。 必须手动创建映射。步骤如下： 第 1 步：确保文件/ etc / subuid 和/ etc / subgid 存在 touch / etc / subuid / etc / subgid 第 2 步：使用--userns-remap 标志启动 docker 守护进程 dockerd dockerd --userns-remap = default
影响	注意用户命名空间重新映射使得不少 Docker 功能不兼容，可查看 Docker 文档和参考链接以获取详细信息。
默认值	默认情况下，用户命名空间不会重新映射。
参考文献	*1.http://man7.org/linux/man-pages/man7/user_namespaces.7.html* *2.https://docs.docker.com/engine/reference/commandline/dockerd/#daem*

	on-user-namespace-options *3.http://events.linuxfoundation.org/sites/events/files/slides/User%20Namespaces%20-%20ContainerCon%202015%20-%2016-9-final_0.pdf*

2.9 使用默认 cgroup

描述	查看--cgroup-parent 选项允许设置用于所有容器的默认 cgroup parent。 如果没有特定用例,则该设置应保留默认值。
安全出发点	系统管理员可定义容器应运行的 cgroup。 若系统管理员没有明确定义 cgroup，容器也会在 docker cgroup 下运行。 应该监测和确认使用情况。 通过加到与默认不同的 cgroup，导致不合理地共享资源，从而可能会主机资源耗尽。
审计方法	ps -ef | grep dockerd
结果判定	确保'--cgroup-parent'参数未设置或设置为适当的非默认 cgroup。
修复措施	默认设置够用的话，可保留。 如果要特别设置非默认 cgroup，在启动时将-cgroup-parent 参数传递给 docker 守护程序。 例如， dockerd --cgroup-parent = / foobar
影响	None
默认值	默认情况下，docker 守护程序使用/ docker for fs cgroup driver 和 system.slice for systemd cgroup 驱动程序。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#default-cgroup-parent*

2.10 设置容器的默认空间大小

描述	在某些情况下，可能需要大于 10G 的容器空间。需要仔细选择空间的大小。
安全出发点	守护进程重启时可以增加容器空间的大小。 用户可以通过设置默认容器空间值来进行扩大，但不允许缩小。设立该值的时候需要谨慎，防止设置不当带来空间耗尽的情况。
审计方法	ps -ef | grep dockerd
结果判定	执行上述命令，它不应显示任何--storage-opt dm.basesize 参数
修复措施	如无需要，不要设置--storage-opt dm.basesize
影响	None
默认值	默认空间大小为 10G
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#storage-driver-options*

2.11 启用 docker 客户端命令的授权

描述	使用本机 Docker 授权插件或第三方授权机制与 Docker 守护程序来管理对 Docker 客户端命令的访问。
安全出发点	Docker 默认是没有对客户端命令进行授权管理的功能。 任何有权访问 Docker 守护程序的用户都可以运行任何 Docker 客户端命令。 对于使用 Docker 远程 API 来调用守护进程的调用者也是如此。 如果需要细粒度的访问控制，可以使用授权插件并将其添加到 Docker 守护程序配置中。使用授权插件，Docker 管理员可以配置更细粒度访问策略来管理对 Docker 守护进程的访问。Docker 的第三方集成可以实现他们自己的授权模型，以要求 Docker 的本地授权插件（即 Kubernetes，Cloud Foundry，Openshift）之外的 Docker 守护进程的授权。
审计方法	ps -ef | grep dockerd
结果判定	如果使用 docker 本地授权，可使用--authorization-plugin 参数加载授权插件。
修复措施	第 1 步：安装/创建授权插件。 第 2 步：根据需要配置授权策略。 第 3 步：重启 docker 守护进程，如下所示：dockerd --authorization-plugin = <PLUGIN_ID>
影响	使用授权插件可能会导致性能下降。
默认值	默认情况下，未设置授权插件。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#access-authorization* *2.https://docs.docker.com/engine/extend/plugins_authorization/* *3.https://github.com/twistlock/authz*

2.12 配置集中和远程日志记录

描述	Docker 现在支持各种日志驱动程序。 存储日志的最佳方式是支持集中式和远程日志记录。
安全出发点	集中和远程日志确保所有重要的日志记录都是安全的，以满足容灾的要求。 Docker 支持多种类型的日志驱动程序，可根据自身情况选取。
审计方法	运行 docker info 并确保日志记录驱动程序属性被设置为适当的。 docker info --format '{{ .LoggingDriver }}'
结果判定	可通过如下命令查看 --log-driver 的设置。 ps -ef | grep dockerd
修复措施	第 1 步：按照其文档设置所需的日志驱动程序。 第 2 步：使用该日志记录驱动程序启动 docker 守护进程。 例如，dockerd --log-driver = syslog --log-opt syslog-address = tcp：//192.xxx.xxx.xxx
影响	None.
默认值	默认情况下，容器日志为 json 文件格式
参考文献	*1.https://docs.docker.com/engine/admin/logging/overview/*

2.13 禁用旧仓库版本（v1）上的操作

描述	最新的 Docker 镜像仓库是 v2。 遗留镜像仓库版本（v1）上的所有操作都应受到限制
安全出发点	Docker 镜像仓库 v2 在 v1 中引入了许多性能和安全性改进。 它支持容器镜像来源验证和其他安全功能。 因此，对 Docker V1 仓库的操作应该受到限制
审计方法	ps -ef | grep dockerd
结果判定	上面的命令应该列出--disable-legacy-registry 作为传递给 docker 守护进程的选项。
修复措施	启动 docker 守护进程如下 dockerd --disable-legacy-registry
影响	旧版镜像仓库操作将受到限制
默认值	默认情况下，允许旧版镜像仓库操作
参考文献	*1.https://docs.docker.com/edge/engine/reference/commandline/dockerd/#legacy-registries* *2.https://docs.docker.com/registry/spec/api/* *3.https://the.binbashtheory.com/creating-private-docker-registry-2-0-with-token-authentication-service/* *4.https://blog.docker.com/2015/07/new-tool-v1-registry-docker-trusted-registry-v2-open-source/* *5.http://www.slideshare.net/Docker/docker-registry-v2*

2.14 启用实时恢复

描述	-live-restore 参数可以支持无守护程序的容器运行。 它确保 Docker daemon 在关闭或恢复时不会停止容器，并在重新启动后重新连接到容器。
安全出发点	可用性作为安全一个重要的属性。 在 Docker 守护进程中设置'--live-restore'标志可确保当 docker 守护进程不可用时容器执行不会中断。 这也意味着当更新和修复 docker 守护进程而不会导致容器停止工作。
审计方法	运行 docker info 并确保 Live Restore Enabled 属性设置为 true。 docker info --format '{{ .LiveRestoreEnabled }}'
结果判定	或者，运行以下命令并确保使用--live-restore。 ps -ef | grep dockerd
修复措施	在守护进程模式下运行 docker 并传递'--live-restore'作为参数。 例如，dockerd --live-restore
影响	None
默认值	默认情况下，--live-restore 不启用
参考文献	1. *https://docs.docker.com/engine/admin/live-restore/*

2.15 禁用 userland 代理

描述	当容器端口需要被映射时，docker 守护进程都会启动用于端口转发的 userland-proxy 方式。 如果使用了 DNAT 方式，该功能可以被禁用。
安全出发点	Docker 引擎提供了两种机制将主机端口转发到容器,DNAT 和 userland-proxy。 在大多数情况下，DNAT 模式是首选，因为它提高了性能，并使用本地 Linux iptables 功能而需要附加组件。 如果 DNAT 可用，则应在启动时禁用 userland-proxy 以减少安全风险。
审计方法	ps -ef | grep dockerd
结果判定	确保--userland-proxy 参数设置为 false。
修复措施	运行 Docker 守护进程如下：dockerd --userland-proxy = false
影响	某些旧版 Linux 内核的系统可能无法支持 DNAT，因此需要 userland-prox 服务。 此外，某些网络设置可能会因删除 userland-prox 而受到影响。
默认值	默认情况下，userland-prox 已启用。
参考文献	*1.http://windsock.io/the-docker-proxy/* *2.https://github.com/docker/docker/issues/14856* *3 https://github.com/docker/docker/issues/22741* *4.https://docs.docker.com/engine/userguide/networking/default_network/binding/*

2.16 应用守护进程范围的自定义 seccomp 配置文件

描述	如果需要，您可以选择在守护进程级别自定义 seccomp 配置文件，并覆盖 Docker 的默认 seccomp 配置文件。
安全出发点	大量系统调用暴露于每个用户级进程，其中许多系统调用在整个生命周期中都未被使用。 大多数应用程序不需要所有的系统调用，因此可以通过减少可用的系统调用来增加安全性。 可自定义 seccomp 配置文件，而不是使用 Docker 的默认 seccomp 配置文件。 如果 Docker 的默认配置文件够用的话，则可以选择忽略此建议。
审计方法	运行以下命令并查看 “SecurityOptions” 部分中列出的 seccomp 配置文件。 如果它是默认，那就意味着，应用了 Docker 的默认 seccomp 配置文件。 docker info –format='{{.SecurityOptions}}'
结果判定	查看是否有非默认的 seccomp 配置文件
修复措施	默认情况下，Docker 使用默认 seccomp 配置文件。 如果这对当前环境有益，则不需要采取任何行动。 当然，也可以选择应用自己的 seccomp 配置文件，需在守护进程启动时使用--seccomp-profile 标志，或将其放入守护程序运行时参数文件中。 dockerd --seccomp-profile </ path / to / seccomp / profile>
影响	错误配置的 seccomp 配置文件可能会中断的容器运行。 Docker 默认的策略兼容性很好，可以解决一些基本的安全问题。 所以，在重写默认值时，你应该非常小心。
默认值	默认情况下，Docker 应用 seccomp 配置文件。
参考文献	*1. https://docs.docker.com/engine/security/seccomp/* *2. https://github.com/docker/docker/pull/26276*

2.17 生产环境中避免实验性功能

描述	避免生产环境中的实验性功-Experimental。
安全出发点	docker 实验功能现在是一个运行时 docker 守护进程标志，其作为运行时标志传递给 docker 守护进程，激活实验性功能。 实验性功能现在虽然比较稳定，但是一些功能可能没有大规模经使用，并不能保证 API 的稳定性，所以不建议在生产环境中使用。
审计方法	运行下面的命令，并确保在 Server 部分将 Experimental 属性设置为 false。 docker version –format='{{.Server.Experimental}}'
结果判定	应该返回 false
修复措施	不要将--experimental 作为运行时参数传递给 docker 守护进程。
影响	None
默认值	默认情况下，docker 守护进程不会激活实验功能。
参考文献	*1.https://docs.docker.com/edge/engine/reference/commandline/dockerd/#options*

2.18 限制容器获取新的权限

描述	默认情况下，限制容器通过 suid 或 sgid 位获取附加权限。
安全出发点	一个进程可以在内核中设置 no_new_priv。 它支持 fork，clone 和 execve。 no_new_priv 确保进程或其子进程不会通过 suid 或 sgid 位获得任何其他特权。 这样，很多危险的操作就降低安全风险。在守护程序级别进行设置可确保默认情况下，所有新容器不能获取新的权限。
审计方法	ps -ef | grep dockerd
结果判定	确保--no-new-privileges 参数存在且未设置为 false。
修复措施	运行 Docker 守护进程如下：dockerd --no-new-privileges
影响	no_new_priv 会阻止像 SELinux 这样的 LSM 访问当前进程的进程标签。
默认值	默认情况下，容器不会获得新的权限。
参考文献	*1. https://github.com/moby/moby/pull/29984* *2. https://github.com/moby/moby/pull/20727*

3.docker 守护程序文件配置

3.1 设置 docker.service 文件的所有权为 root:root

描述	验证'docker.service'文件所有权和组所有权是否正确设置为 “root” 。
安全出发点	docker.service'文件包含可能会改变 Docker 守护进程行为的敏感参数。因此，它应该由 “root” 拥有和归属，以保持文件的完整性。
审计方法	第 1 步：找出文件位置：systemctl show -p FragmentPath docker.service 步骤 2：如果文件不存在，则此建议不适用。 如果该文件存在，请使用正确的文件路径执行以下命令，以验证文件是由 root 拥有还是属于群组。 例如，stat -c %U:%G /lib/systemd/system/docker.service | grep -v root:root
结果判定	上述命令不应该返回任何内容。
修复措施	步骤 1：找出文件位置：systemctl show -p FragmentPath docker.service 步骤 2：如果该文件不存在，则此建议不适用。 如果文件存在，请使用正确的文件路径执行以下命令，将文件的所有权和组所有权设置为 “root” 。 例如，chown root:root /usr/lib/systemd/system/docker.service
影响	None.
默认值	该文件可能不存在于系统上。 在这种情况下，此建议不适用。 默认情况下，如果文件存在，则该文件的所有权和组所有权正确设置为 “root” 。
参考文献	*1. https://docs.docker.com/engine/admin/systemd/*

3.2 设置 docker.service 文件权限为 644 或更多限制性

描述	验证'docker.service'文件权限是否正确设置为'644'或更多限制。
安全出发点	docker.service'文件包含可能会改变 Docker 守护程序行为的敏感参数。因此，除 “root” 之外的任何其他用户不应该保留文件的完整性。
审计方法	第 1 步：找出文件位置：systemctl show -p FragmentPath docker.service 步骤 2：如果文件不存在，则此建议不适用。 如果文件存在，请使用正确的文件路径执行以下命令，以验证文件权限是否设置为 644 或更多限制。 例如， stat -c %a /etc/systemd/system/docker.service
结果判定	返回是否为 644
修复措施	步骤 1：找出文件位置：systemctl show -p FragmentPath docker.service 步骤 2：如果该文件不存在，则此建议不适用。 如果文件存在，请使用正确的文件路径执行以下命令，将文件权限设置为'644'。 例如，chmod 644 /usr/lib/systemd/system/docker.service
影响	None.
默认值	该文件可能不存在于系统上。 在这种情况下，此建议不适用。 默认情况下，如果文件存在，则文件权限正确设置为'644'。
参考文献	*1. https://docs.docker.com/articles/systemd/*

3.3 设置 docker.socket 文件所有权为 root:root

描述	验证'docker.socket'文件所有权和组所有权是否正确设置为 “root” 。
安全出发点	docker.socket 文件包含可能会改变 Docker 远程 API 行为的敏感参数。 因此，它应该拥有 “root” 权限，以保持文件的完整性。
审计方法	步骤 1：查找文件位置：systemctl show -p FragmentPath docker.socket 步骤 2：如果文件不存在，则此建议不适用。 如果文件存在，请使用正确的文件路径执行以下命令，以验证该文件是由 “root” 拥有和归属的。 例如，stat -c %U:%G /lib/systemd/system/docker.service | grep -v root:root
结果判定	上述命令不应该返回任何内容。
修复措施	步骤 1：找出文件位置：systemctl show -p FragmentPath docker.socket 步骤 2：如果该文件不存在，则此建议不适用。 如果文件存在，请使用正确的文件路径执行以下命令，将文件的所有权和组所有权设置为 “root” 。 例如，chown root：root /usr/lib/systemd/system/docker.socket
影响	None.
默认值	该文件可能不存在于系统上。 在这种情况下，此建议不适用。 默认情况下，如果文件存在，则该文件的所有权和组所有权正确设置为 “root” 。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-socket-option* *2.https://github.com/docker/docker-ce/blob/master/components/packaging/deb/systemd/docker.socket*

3.4 设置 docker.socket 文件权限为 644 或更多限制性

描述	验证'docker.socket'文件权限是否正确设置为'644'或更多限制。
安全出发点	docker.socket'文件包含可能会改变 Docker 远程 API 行为的敏感参数。因此，它应该只能通过'root'来保持文件的完整性。
审计方法	步骤 1：找出文件位置：systemctl show -p FragmentPath docker.socket 步骤 2：如果该文件不存在，则此建议不适用。 如果文件存在，请使用正确的文件路径执行以下命令，以验证文件权限是否设置为 “644” 或更多限制。 例如，stat –c%a /usr/lib/systemd/system/docker.socket
结果判定	应该返回 644
修复措施	步骤 1：找出文件位置：systemctl show -p FragmentPath docker.socket 步骤 2：如果该文件不存在，则此建议不适用。 如果文件存在，请使用正确的文件路径执行以下命令，将文件权限设置为'644'。 例如，chmod 644 /usr/lib/systemd/system/docker.socket
影响	None.
默认值	该文件可能不存在于系统上。 在这种情况下，此建议不适用。 默认情况下，如果文件存在，则该文件的文件权限正确设置为'644'。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#bind-docker-to-another-hostport-or-a-unix-socket* *2.https://github.com/YungSang/fedora-atomic-packer/blob/master/oem/docker.socket* *3.http://daviddaeschler.com/2014/12/14/centos-7rhel-7-and-docker-containers-on-boot/*

3.5 设置/etc/docker 目录所有权为 root:root

描述	验证/ etc / docker 目录所有权和组所有权是否正确设置为 “root” 。
安全出发点	除了各种敏感文件之外，'/ etc / docker'目录还包含证书和密钥。 因此，它应该由 “root” 拥有和归组来维护目录的完整性。
审计方法	执行以下命令以验证该目录是由 “root” 拥有和归属的： stat -c %U:%G /etc/docker | grep -v root:root
结果判定	以上命令不应该返回任何东西
修复措施	chown root：root / etc / docker 这将将目录的所有权和组所有权设置为 “root” 。
影响	None.
默认值	默认情况下，此目录的所有权和组所有权正确设置为 “root” 。
参考文献	*1. https://docs.docker.com/engine/security/https/*

3.6 设置/etc/docker 目录权限为 755 或更多限制性

描述	验证/ etc / docker 目录权限是否正确设置为'755'或更多限制。
安全出发点	/ etc / docker'目录除了各种敏感文件外还包含证书和密钥。 因此，它只能由'root'写入以维护目录的完整性。
审计方法	执行以下命令以验证目录具有 “755” 或更多限制的权限： stat -c%a /etc/docker | grep -v root:root
结果判定	应该显示为 755
修复措施	chmod 755 / etc / docker 这将把目录的权限设置为'755'。
影响	None.
默认值	默认情况下，此目录的权限正确设置为'755'。
参考文献	*1. https://docs.docker.com/engine/security/https/*

3.7 设置仓库证书文件所有权为 root：root

描述	验证所有仓库证书文件（通常位于/etc/docker/certs.d/ <registry-name> 目录下）均由 “root” 拥有并归组所有。
安全出发点	/etc/docker/certs.d/ <registry-name>目录包含 Docker 镜像仓库证书。这些证书文件必须由 “root” 和其组拥有，以维护证书的完整性
审计方法	执行以下命令以验证镜像仓库证书文件是由 “root” 拥有并由 Group-owned 拥有的： stat -c %U:%G /etc/docker/certs.d/* | grep -v root:root
结果判定	上面的命令不应该返回任何东西。
修复措施	chown root：root /etc/docker/certs.d/<registry-name>/* 这将将镜像仓库证书文件的所有权和组所有权设置为 “root” 。
影响	None.
默认值	默认情况下，镜像仓库证书文件的所有权和组所有权正确设置为 “root” 。
参考文献	*1. https://docs.docker.com/registry/insecure/*

3.8 设置仓库证书文件权限为 444 或更多限制性

描述	验证所有镜像仓库证书文件（通常位于/etc/docker/certs.d/ <registry-name>目录下）具有 “444” 或更多限制的权限。
安全出发点	/etc/docker/certs.d/ <registry-name>目录包含 Docker 镜像仓库证书。这些证书文件必须具有 “444” 权限，以维护证书的完整性。
审计方法	执行以下命令以验证镜像仓库证书文件是否具有 “444” 或更多限制的权限：stat –c%a /etc/docker/certs.d/<registry-name>/*
结果判定	将返回 444
修复措施	“chmod 444 /etc/docker/certs.d/<registry-name>/* 这会将镜像仓库证书文件的权限设置为'444'。
影响	None.
默认值	默认情况下，镜像仓库证书文件的权限可能不是 “444” 。 默认文件权限由系统或用户特定的 umask 值控制。
参考文献	*1. https://docs.docker.com/registry/insecure/*

3.9 设置 TLS CA 证书文件所有权为 root：root

描述	验证 TLS CA 证书文件（与--tlscacert'参数一起传递的文件）是由“root”拥有和分组拥有的。
安全出发点	TLS CA 证书文件应受到保护，不受任何篡改。 它用于指定的 CA 证书验证。 因此，它必须由“root”拥有，以维护 CA 证书的完整性。
审计方法	执行以下命令以验证 TLS CA 证书文件是否由“root”和其组拥有： stat –c %U:%G <路径到 TLS CA 证书文件> | grep -v root:root
结果判定	上面的命令不应该返回任何东西。
修复措施	chown root：root <路径到 TLS CA 证书文件> 这将 TLS CA 证书文件的所有权和组所有权设置为“root”。
影响	None.
默认值	默认情况下，TLS CA 证书文件的所有权和组属性正确设置为“root”。
参考文献	*1. https://docs.docker.com/registry/insecure/* *2. https://docs.docker.com/engine/security/https/*

3.10 设置 TLS CA 证书文件权限为 444 或更多限制性

描述	验证 TLS CA 证书文件（与--tlscacert'参数一起传递的文件）具有“444”或更多限制的权限。
安全出发点	TLS CA 证书文件应受到保护，不受任何篡改。因此，它必须具有“444”的权限以维护 CA 证书的完整性。
审计方法	执行以下命令以验证 TLS CA 证书文件是否具有“444”或更多限制的权限：stat –c%a <路径到 TLS CA 证书文件>
结果判定	应返回 444
修复措施	chmod 444 <路径到 TLS CA 证书文件> 这将把 TLS CA 文件的文件权限设置为'444'。
影响	None.
默认值	默认情况下，TLS CA 证书文件的权限可能不是“444”。 默认文件权限由系统或用户特定的 umask 值控制。
参考文献	*1. https://docs.docker.com/registry/insecure/* *2. https://docs.docker.com/engine/security/https/*

3.11 设置 docker 服务器证书文件所有权为 root：root

描述	验证 Docker 服务器证书文件（与--tlscert'参数一起传递的文件）是否由“root” 和其组拥有。
安全出发点	Docker 服务器证书文件应受到保护，不受任何篡改。 它用于验证 Docker 服务器。 因此，它必须由“root” 拥有以维护证书的完整性。
审计方法	执行以下命令以验证 Docker 服务器证书文件是否由“root” 拥有和分组拥有： stat –c %U:%G <路径到 Docker 服务器证书文件> | grep -v root:root
结果判定	上面的命令不应该返回任何东西。
修复措施	“chown root:root <路径到 Docker 服务器证书文件> 这将 Docker 服务器证书文件的所有权和组所有权设置为“root” 。
影响	None.
默认值	默认情况下，Docker 服务器证书文件的所有权和组所有权正确设置为“root” 。
参考文献	*1.https:/docs.docker.com/registry/insecure/* *2. https://docs.docker.com/engine/security/https/*

3.12 设置 docker 服务器证书文件权限为 444 或更多限制

描述	验证 Docker 服务器证书文件（与--tlscert'参数一起传递的文件）具有“444”或更多限制的权限。
安全出发点	Docker 服务器证书文件应受到保护，不受任何篡改。 因此，它必须具有“444”的权限以维护证书的完整性。
审计方法	执行以下命令以验证 Docker 服务器证书文件是否具有“444”或更多限制的权限： stat –c %a <Docker 服务器证书文件的路径>
结果判定	应返回 444
修复措施	chmod 444 <路径到 Docker 服务器证书文件> 这将把 Docker 服务器文件的文件权限设置为'444'。
影响	None.
默认值	默认情况下，Docker 服务器证书文件的权限可能不是“444”。 默认文件权限由系统或用户特定的 umask 值控制。
参考文献	*1. https://docs.docker.com/registry/insecure/* *2. https://docs.docker.com/engine/security/https/*

3.13 设置 docker 服务器证书密钥文件所有权为 root：root

描述	验证 Docker 服务器证书密钥文件（与--tlskey'参数一起传递的文件）是由由“root”拥有。
安全出发点	Docker 服务器证书密钥文件应受到保护，不受任何篡改或不必要的读取。 它保存 Docker 服务器证书的私钥。 因此，它必须由“root”拥有拥有，以维护 Docker 服务器证书的完整性。
审计方法	执行以下命令验证 Docker 服务器证书密钥文件是否由“root”拥有： stat –c %U:%G <路径到 Docker 服务器证书密钥文件> | grep -v root:root
结果判定	上面的命令不应该返回任何东西。
修复措施	chown root:root <路径到 Docker 服务器证书密钥文件> 这将将 Docker 服务器证书密钥文件的所有权和组所有权设置为“root”。
影响	None.
默认值	默认情况下，Docker 服务器证书密钥文件的所有权和组所有权正确设置为“root”。
参考文献	*1. https://docs.docker.com/registry/insecure/* *2. https://docs.docker.com/engine/security/https/*

3.14 设置 docker 服务器证书密钥文件权限为 400

描述	验证 Docker 服务器证书密钥文件（与--tlskey'参数一起传递的文件）的权限为 “400” 。
安全出发点	Docker 服务器证书密钥文件应受到保护，不受任何篡改或不必要的读取。 它保存 Docker 服务器证书的私钥。 因此，它必须具有 “400” 的权限，以维护 Docker 服务器证书的完整性。
审计方法	执行以下命令以验证 Docker 服务器证书密钥文件是否具有 “400” 的权限： stat –c%a <Docker 服务器证书密钥文件的路径>
结果判定	应返回 400
修复措施	chmod 400 <路径到 Docker 服务器证书密钥文件> 这将 Docker 服务器证书密钥文件权限设置为 “400” 。
影响	None.
默认值	默认情况下，Docker 服务器证书密钥文件的权限可能不是 “400” 。 默认文件权限由系统或用户特定的 umask 值控制。
参考文献	*1. https://docs.docker.com/registry/insecure/* *2. https://docs.docker.com/engine/security/https/*

3.15 设置 docker.sock 文件所有权为 root：docker

描述	验证 Docker sock 文件由 “root” 拥有，而用户组为 “docker” 。
安全出发点	Docker 守护进程以 root 用户身份运行。 因此，默认的 Unix 套接字必须由 root 拥有。 如果任何其他用户或进程拥有此套接字，那么该非特权用户或进程可能与 Docker 守护进程交互。 另外，这样的非特权用户或进程可能与容器交互。 这样非常不安全。 另外，Docker 安装程序会创建一个名为 docker 的用户组。 可以将用户添加到该组，然后这些用户将能够读写默认的 Docker Unix 套接字。 docker 组成员由系统管理员严格控制。 如果任何其他组拥有此套接字，那么该组的成员可能会与 Docker 守护进程交互。。 因此，默认的 Docker Unix 套接字文件必须由 docker 组拥有权限，以维护套接字文件的完整性。
审计方法	执行以下命令以验证 Docker 套接字文件是由 “root” 拥有，由 “docker” 拥有的组： stat –c %U:%G /var/run/docker.sock | grep -v root:docker
结果判定	上面的命令不应该返回任何东西。
修复措施	chown root：docker /var/run/docker.sock 这将将所有权设置为 “root” 和组所有权到 “docker” 作为默认 Docker 套接字文件。
影响	None.
默认值	默认情况下，Docker 套接字文件的所有权和组所有权正确设置为 “root：docker” 。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-socket-option* *2.https://docs.docker.com/engine/reference/commandline/dockerd/#bind-docker-to-another-hostport-or-a-unix-socket*

3.16 设置 docker.sock 文件权限为 660 或更多限制性

描述	验证 Docker 套接字文件是否具有 “660” 或更多限制的权限
安全出发点	只有'root'和'docker'组的成员允许读取和写入默认的 Docker Unix 套接字。 因此，Docket 套接字文件必须具有 “660” 或更多限制的权限。
审计方法	执行以下命令以验证 Docker 套接字文件是否具有 “660” 或更多限制的权限：stat –c%a /var/run/docker.sock
结果判定	应该返回 660
修复措施	chmod 660 /var/run/docker.sock 这会将 Docker 套接字文件的文件权限设置为'660'。
影响	None.
默认值	默认情况下，Docker 套接字文件的权限正确设置为'660'
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-socket-option* *2.https://docs.docker.com/engine/reference/commandline/dockerd/#bind-docker-to-another-hostport-or-a-unix-socket*

3.17 设置 daemon.json 文件所有权为 root：root

描述	验证'daemon.json'文件所有权和组所有权是否正确设置为 “root” 。
安全出发点	daemon.json'文件包含可能会改变 docker 守护程序行为的敏感参数。 因此，它应该由 “root” 拥有，以维护文件的完整性。
审计方法	执行以下命令以验证该文件是由 “root” 拥有和分组拥有的： stat –c %U:%G /etc/docker/daemon.json | grep -v root:root
结果判定	上面的命令不应该返回任何东西。
修复措施	chown root：root /etc/docker/daemon.json 这将将文件的所有权和组所有权设置为 “root” 。
影响	None.
默认值	该文件可能不存在于系统上。 在这种情况下，此建议不适用。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file*

3.18 设置 daemon.json 文件权限为 644 或更多限制性

描述	验证'daemon.json'文件权限是否正确设置为'644'或更多限制。
安全出发点	daemon.json'文件包含可能会改变 docker 守护程序行为的敏感参数。 因此，只能通过'root'来保存文件的完整性。
审计方法	执行以下命令以验证文件权限是否正确设置为 “644” 或更多限制： stat –c%a /etc/docker/daemon.json
结果判定	应该返回 644
修复措施	chmod 644 /etc/docker/daemon.json 这将把该文件的文件权限设置为'644'。
影响	None.
默认值	该文件可能不存在于系统上。 在这种情况下，此建议不适用。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file*

3.19 设置/etc/default/docker 文件所有权为 root：root

描述	验证 “/ etc / default / docker” 文件所有权和组所有权是否正确设置为 “root” 。
安全出发点	/etc/default/docker 文件包含可能会改变 docker 守护进程行为的敏感参数。 因此，它应该由 “root” 拥有，以维护文件的完整性。
审计方法	执行以下命令以验证该文件是由 “root” 拥有和分组拥有的： stat -c%U:%G /etc/default/ docker | grep -v root:root
结果判定	上面的命令不应该返回任何东西。
修复措施	chown root:root /etc/default/docker 这将文件的所有权和组所有权设置为 “root” 。
影响	None.
默认值	该文件可能不存在于系统上。 在这种情况下，此建议不适用。
参考文献	*1. https://docs.docker.com/engine/admin/configuring/*

3.20 设置/etc/default/docker 文件权限为 644 或更多限制性

描述	验证/etc/default/docker'文件权限是否正确设置为'644'或更多限制。
安全出发点	/etc/default/docker 文件包含可能会改变 docker 守护进程行为的敏感参数。 因此，只能通过'root'来保存文件的完整性。
审计方法	执行以下命令以验证文件权限是否正确设置为 “644” 或更多限制： stat –c%a /etc/default/docker
结果判定	应当返回 644
修复措施	chmod 644 / etc / default / docker 这将把该文件的文件权限设置为'644'。
影响	None.
默认值	该文件可能不存在于系统上。 在这种情况下，此建议不适用。
参考文献	*1. https://docs.docker.com/engine/admin/configuring/*

4 容器镜像和构建文件

4.1 创建容器的用户

描述	为容器镜像的 Dockerfile 中的容器创建非 root 用户。
安全出发点	如果可能，指定非 root 用户身份运行容器是一个很好的做法。 虽然用户命名空间映射可用，但是如果用户在容器镜像中指定了用户，则默认情况下容器将作为该用户运行，并且不需要特定的用户命名空间重新映射。
审计方法	docker ps --quiet | xargs docker inspect --format '{{ .Id }}: User={{.Config.User}}'
结果判定	上述命令应该返回容器用户名或用户 ID。 如果为空，则表示容器以 root 身份运行。
修复措施	确保容器镜像的 Dockerfile 包含以下指令：USER <用户名或 ID> 其中用户名或 ID 是指可以在容器基础镜像中找到的用户。 如果在容器基础镜像中没有创建特定用户，则在 USER 指令之前添加 useradd 命令以添加特定用户。 例如，在 Dockerfile 中创建用户： RUN useradd -d /home/username -m -s / bin / bash username USER username 注意：如果镜像中有容器不需要的用户，请考虑删除它们。 删除这些用户后，提交镜像，然后生成新的容器实例以供使用。
影响	None.
默认值	默认情况下，容器以 root 权限运行，并以容器中的用户 root 身份运行。
参考文献	*1. https://github.com/docker/docker/issues/2918* *2. https://github.com/docker/docker/pull/4572* *3. https://github.com/docker/docker/issues/7906*

4.2 容器使用可信的基础镜像

描述	确保容器镜像是从头开始编写的，或者是基于通过安全仓库下载的另一个已建立且可信的基本镜像。
安全出发点	官方存储库是由 Docker 社区或供应商优化的 Docker 镜像。 可能还存在其他不安全的公共存储库。 在从 Docker 和第三方获取容器镜像时，需谨慎使用。
审计方法	第 1 步 - 检查 Docker 主机以查看执行以下命令使用的 Docker 镜像：docker images 这将列出当前可用于 Docker 主机的所有容器镜像。 访谈系统管理员并获取证据，证明镜像列表是通过安全的镜像仓库获到的，也可简单的从镜像的 TAG 名称来判断是否为可信镜像。 步骤 2 - 对于在 Docker 主机上找到的每个 Docker 镜像，检查镜像的构建方式，以验证是否来自可信来源：docker history <imageName>
结果判定	判断镜像来源的合法性
修复措施	配置和使用 Docker 内容信任。 检查 Docker 镜像历史记录以评估其在网络上运行的风险。 扫描 Docker 镜像以查找其依赖关系中的漏洞。
影响	None.
默认值	无
参考文献	*1. https://titanous.com/posts/docker-insecurity* *2. https://registry.hub.docker.com/* *3.http://blog.docker.com/2014/10/docker-1-3-signed-images-process-injection-security-options-mac-shared-directories/* *4. https://github.com/docker/docker/issues/8093* *5. https://docs.docker.com/engine/reference/commandline/pull/* *6. https://github.com/docker/docker/pull/11109* *7. https://blog.docker.com/2015/11/docker-trusted-registry-1-4/*

4.3 容器中不安装没有必要的软件包

描述	容器往往是操作系统的最简的版本。 不要安装任何不需要的软件。

安全出发点	安装不必要的软件可能会增加容器的攻击风险。因此，除了容器的真正需要的软件之外，不要安装其他多余的软件。
审计方法	步骤 1：通过执行以下命令列出所有运行的容器实例：docker ps --quiet 步骤 2：对于每个容器实例，执行以下或等效的命令：docker exec $ INSTANCE_ID rpm –qa 或者 docker exec $ INSTANCE_ID dpkg –l 要取决于你的镜像是基于哪一种软件包管理方式。
结果判定	上述命令将列出安装在容器上的包。 查看列表并确保它是合法的。
修复措施	如果可能的话，考虑使用最小基本镜像而不是标准的 Redhat / Centos / Debian 镜像。 可以选择 BusyBox 和 Alpine。 这不仅可以将您的镜像大小从> 150Mb 修剪至 20 Mb 左右，还可以使用更少的工具和路径来提升权限。
影响	None.
默认值	不适用。
参考文献	*1. https://docs.docker.com/userguide/dockerimages/* *2.http://www.livewyer.com/blog/2015/02/24/slimming-down-your-docker-containers-alpine-linux* *3. https://github.com/progrium/busybox*

4.4 扫描镜像漏洞并且构建包含安全补丁的镜像

描述	应该经常扫描镜像以查找漏洞。 重建镜像安装最新的补丁。
安全出发点	安全补丁可以解决软件的安全问题。 可以使用镜像漏洞扫描工具来查找镜像中的任何类型的漏洞，然后检查可用的补丁以减轻这些漏洞。修补程序将系统更新到最新的代码库。 此外，如果镜像漏洞扫描工具可以执行二进制级别分析，而不仅仅是版本字符串匹配，则会更好。
审计方法	步骤 1：通过执行以下命令列出所有运行的容器实例：docker ps --quiet 步骤 2：对于每个容器实例，执行下面的或等效的命令来查找容器中安装的包的列表。 确保安装各种受影响软件包的安全更新。 docker exec $ INSTANCE_ID rpm –qa 或者，可以运行镜像漏洞扫描工具，它可以扫描生态系统中的所有镜像，然后修复。另外，可使用 DoSec 团队针对镜像的全面扫描工具。
结果判定	
修复措施	按照以下步骤重新构建带有安全补丁的镜像： 第 1 步：取出所有基本镜像（即给定一组 Dockerfiles，提取在 FROM 指令中声明的所有镜像，并重新提取它们以检查更新/修补版本） 第 2 步：强制重建每个镜像：docker build --no-cache 第 3 步：使用更新的镜像重新启动所有容器。 还可以在 Dockerfile 中使用 ONBUILD 指令来触发经常用作基本镜像的特定更新指令。
影响	None.
默认值	默认情况下，容器和镜像不会自动更新。
参考文献	*1. https://docs.docker.com/userguide/dockerimages/* *2. https://docs.docker.com/docker-cloud/builds/image-scan/* *3. https://blog.docker.com/2016/05/docker-security-scanning/* *4. https://docs.docker.com/engine/reference/builder/#/onbuild*

4.5 启用 docker 内容信任

描述	默认情况下禁用内容信任，为了安全起见，可以启用。
安全出发点	内容信任为向远程 Docker 镜像仓库发送和接收的数据提供了使用数字签名的能力。 这些签名允许客户端验证特定镜像标签的完整性和发布者。这确保了容器镜像的来源的合法性。
审计方法	echo $DOCKER_CONTENT_TRUST
结果判定	应该返回 1。
修复措施	要在 bash shell 中启用内容信任，请输入以下命令： export DOCKER_CONTENT_TRUST = 1 或者，在配置文件中设置此环境变量，以便每次登录时启用内容信任。
影响	在设置了 DOCKER_CONTENT_TRUST 的环境中，需要在处理镜像时遵循信任过程 - 构建，创建，拉取，推送和运行。 可以使用--disable-content-trust 标志按照需要在标记镜像上运行单独的操作，一般用于测试目的，生成环境中应尽不要使用。
默认值	默认情况下，内容信任被禁用。
参考文献	*1. https://docs.docker.com/engine/security/trust/content_trust/* *2. https://docs.docker.com/engine/reference/commandline/cli/#notary* *3.https://docs.docker.com/engine/reference/commandline/cli/#environment-variables*

4.6 将 HEALTHCHECK 说明添加到容器镜像

描述	在 Docker 容器镜像中添加 HEALTHCHECK 指令以对正在运行的容器执行运行状况检查。
安全出发点	安全性最重要的一个特性就是可用性。 将 HEALTHCHECK 指令添加到容器镜像可确保 docker 引擎定期检查运行的容器实例是否符合该指令，以确保实例仍在运行。根据报告的健康状况，docker 引擎可以退出非工作容器并实例化新容器。
审计方法	运行以下命令，并确保 docker 镜像对 HEALTHCHECK 指令设置。 docker inspect --format ='{{.Config.Healthcheck}}'<镜像 ID>
结果判定	应当返回设置值
修复措施	按照 Docker 文档，并使用 HEALTHCHECK 指令重建容器镜像。
影响	None.
默认值	默认情况下，HEALTHCHECK 未设置。
参考文献	*1. https://docs.docker.com/engine/reference/builder/#healthcheck*

4.7 不在 dockerfile 中单独使用更新命令

描述	不要单独使用 apt-get update 等更新指令，也不要在 Dockerfile 中使用更新指令。
安全出发点	在 Dockerfile 添加更新指令将缓存更新的层。稍后使用相同的指令构建任何镜像时，将使用先前缓存的更新图层。 这可能会拒绝任何新版本进入到以后的版本。
审计方法	第 1 步：运行以下命令以获取镜像列表：docker images 第 2 步：对上面列表中的每个镜像运行以下命令，并查找任何更新指令：docker history <Image_ID> 或者，如果有权访问镜像的 Dockerfile，请确认没有上述更新指示。
结果判定	无更新指令
修复措施	在安装软件包时，请使用最新的固定版本软件包。 或可以在 docker 构建过程中使用--no-cache 标志，以避免使用缓存的层。
影响	None.
默认值	默认情况下，docker 对更新无限制。
参考文献	*1.https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#run* *2. https://github.com/docker/docker/issues/3313*

4.8 镜像中删除 setuid 和 setgid 权限

描述	删除镜像中的 setuid 和 setgid 权限防止容器中的提权攻击。
安全出发点	setuid 和 setgid 可用于提升权限。 虽然这些权限有时必须，应考虑为镜像中不需要的软件包删除这些权限。
审计方法	在镜像上运行以下命令以列出具有 setuid 和 setgid 权限的可执行文件：docker run <Image_ID> find / -perm +6000-type f -exec ls -ld {} \; 2> / dev / null
结果判定	仔细检查列表，确保它是合法的。
修复措施	只在需要可执行的文件上允许 setuid 和 setgid 权限。 可在构建时通过在 Dockerfile 中添加以下命令来删除这些权限，最好添加在 Dockerfile 的末尾：RUN find / -perm +6000-type f -exec chmod a-s {} \;|| true
影响	以上命令会导致依赖 setuid 或 setgid 权限（包括合法权限）的可执行文件无法执行。，需要小心处理。
默认值	Not Applicable
参考文献	*1. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf* *2.http://container-solutions.com/content/uploads/2015/06/15.06.15_DockerCheatSheet_A2.pdf* *3. http://man7.org/linux/man-pages/man2/setuid.2.html* *4. http://man7.org/linux/man-pages/man2/setgid.2.html*

4.9 在 dockerfile 中使用 copy 而不是 add

描述	在 Dockerfile 中使用 COPY 指令而不是 ADD 指令。
安全出发点	COPY 指令只是将文件从本地主机复制到容器文件系统。 ADD 指令可能会从远程 URL 下载文件并执行诸如解包等操作。 因此，ADD 指令增加了从 URL 添加恶意文件的风险。
审计方法	步骤 1：运行以下命令获取镜像列表：docker images 步骤 2：对上述列表中的每个镜像执行以下命令，并查找任何 ADD 指令：docker history <Image_ID> 或如果可以访问镜像的 Dockerfile，需验证是否有 ADD 指令
结果判定	不允许存在 ADD 指令
修复措施	在 Dockerfiles 中使用 COPY 指令。
影响	可能需 ADD 指令提供的功能，例如从远程 URL 获取文件。
默认值	Not Applicable
参考文献	*1.https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#/add-or-copy*

4.10 涉密信息不存储在 dockerfile

描述	不要在 Dockerfiles 中存储任何涉密信息。
安全出发点	通过使用 Docker 历史命令，可以查看各种工具和实用程序。 通常情况，镜像发布者提供 Dockerfiles 来构建镜像。 所以，Dockerfiles 中的涉密信息可能会被暴露并被恶意利用。
审计方法	第 1 步：运行以下命令以获取镜像列表：docker images 第 2 步：对上面列表中的每个镜像运行以下命令，并查找是否有涉密信息： docker history<Image_ID> 如果有权访问镜像的 Dockerfile，请确认没有涉密信息。
结果判定	不应该有涉密的信息，如用户账号，私钥证书等。
修复措施	不要在 Dockerfiles 中存储任何类型的涉密信息。
影响	若必须使用，需要制定相应的措施
默认值	默认情况下，在 Dockerfiles 中存储配置密码没有限制。
参考文献	*1. https://github.com/docker/docker/issues/13490* *2. http://12factor.net/config* *3. https://avicoder.me/2016/07/22/Twitter-Vine-Source-code-dump/*

4.11 仅安装已经验证的软件包

描述	在将软件包安装到镜像中之前，验证软件包可靠性。
安全出发点	验证软件包的可靠性对于构建安全的容器镜像至关重要。不合法的软件包可能具有恶意或者存在一些可能被利用的已知漏洞。
审计方法	第 1 步：运行以下命令以获取镜像列表：docker images 第 2 步：对上面列表中的每个镜像运行以下命令，并查看软件包的合法性 docker history<Image_ID> 若可以访问镜像的 Dockerfile，请验证是否检查了软件包的合法性。
结果判定	
修复措施	使用 GPG 密钥下载和验证您所选择的软件包或任何其他安全软件包分发机制。
影响	None
默认值	不适用
参考文献	*1. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf* *2.https://github.com/docker-library/httpd/blob/12bf8c8883340c98b3988a7bade8ef2d0d6dcf8a/2.4/Dockerfile* *3.https://github.com/docker-library/php/blob/d8a4ccf4d620ec866d5b42335b699742df08c5f0/7.0/alpine/Dockerfile* *4. https://access.redhat.com/security/team/key*

5 容器运行时保护

5.1 启用 AppArmor 配置文件

描述	AppArmor 是一个高效且易于使用的 Linux 应用程序安全系统。 它可以在很多 Linux 发行版上使用，例如 Debian 和 Ubuntu。
安全出发点	AppArmor 通过执行安全策略（也称为 AppArmor 配置文件）来保护 Linux 操作系统和应用程序免受各种威胁。 您可以为容器创建自己的 AppArmor 配置文件或使用 Docker 的默认 AppArmor 配置文件。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: AppArmorProfile={{ .AppArmorProfile }}'
结果判定	上述命令应该为每个容器实例返回一个有效的 AppArmor 配置文件。
修复措施	如果 AppArmor 适用于你的 Linux 操作系统，可能需要遵循以下步骤： 1.验证是否安装了 AppArmor。 如果没有，请安装。 2.为 Docker 容器创建或导入 AppArmor 配置文件。 3.将此配置文件置于强制模式。 4.使用自定义的 AppArmor 配置文件启动 Docker 容器。 例如，docker run --interactive --tty --security-opt = “apparmor：PROFILENAME” centos / bin / bash 或者，可以保留 Docker 的默认 apparmor 配置文件
影响	AppArmor 配置文件中定义的一组操作限制。 如果配置错误，容器可能无法完成工作。
默认值	默认情况下，docker-default AppArmor 配置文件应用于运行容器，该配置文件可以在/etc/apparmor.d/docker 找到。
参考文献	*1. https://docs.docker.com/engine/security/apparmor/* *2.https://docs.docker.com/engine/reference/run/#security-configuration* *3.https://docs.docker.com/engine/security/security/#other-kernel-security-features*

5.2 设置 SElinux 安全选项

描述	SELinux 是一个有效且易于使用的 Linux 访问控制机制。 默认情况下，它可以在很多 Linux 发行版上使用，如 Red Hat 和 Fedora。
安全出发点	SELinux 提供强制访问控制（MAC）系统，大大增强了默认的自由访问控制（DAC）模型。 因此，可以通过在 Linux 主机上启用 SELinux（如果适用）来增加额外的安全防护。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: SecurityOpt={{ .HostConfig.SecurityOpt }}'
结果判定	上述命令应返回当前为容器配置的所有安全选项。
修复措施	如果 SELinux 适用于你的 Linux 操作系统，请使用它。 可能需要遵循以下步骤： 1.设置 SELinux 状态。 2.设置 SELinux 策略。 3.为 Docker 容器创建或导入 SELinux 策略模板。 4.启用 SELinux 的守护程序模式下启动 Docker。 例如，docker daemon --selinux-enabled 5.使用安全选项启动 Docker 容器。 例如，docker run--interactive --tty --security-opt label = level：TopSecret centos / bin / bash
影响	selinux 配置文件中定义的一组操作限制。 如果配置错误，容器可能无法完成工作。
默认值	默认情况下，在容器上不应用 SELinux 安全选项。
参考文献	*1.https://docs.docker.com/engine/security/security/#other-kernel-security-features* *2. https://docs.docker.com/engine/reference/run/#security-configuration* *3. http://docs.fedoraproject.org/en-US/Fedora/13/html/Security-Enhanced_Linux/* *4.https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/docker_selinux_security_policy*

5.3 linux 内核功能在容器内受限

描述	默认情况下，Docker 使用一组受限制的 Linux 内核功能启动容器。 这意味着可以将任何进程授予所需的功能，而不是 root 访问。 使用 Linux 内核功能，这些进程不必以 root 用户身份运行。
安全出发点	Docker 支持添加和删除功能，允许使用非默认配置文件。 这可能会使 Docker 通过移除功能更加安全，或者通过增加功能来减少安全性。 因此，建议除去容器进程明确要求的所有功能。 例如，容器进程通常不需要如下所示的功能：NET_ADMIN SYS_ADMIN SYS_MODULE
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: CapAdd={{ .HostConfig.CapAdd }} CapDrop={{ .HostConfig.CapDrop }}'
结果判定	验证添加和删除的 Linux 内核功能是否符合每个容器实例的容器进程所需的功能。
修复措施	执行以下命令添加所需的功能： $> docker run --cap-add = { “Capability 1”， “Capability 2” } 例如，docker run --interactive --tty --cap-add = { “NET_ADMIN”， “SYS_ADMIN” } centos：latest / bin / bash 执行以下命令删除不需要的功能： $> docker run --cap-drop = { “能力 1”， “能力 2” } 例如，docker run --interactive --tty --cap-drop = { “SETUID”， “SETGID” } centos：latest / bin / bash 或者， 您可以选择删除所有功能并只添加所需功能： $> docker run --cap-drop = all --cap-add = { “Capability 1”， “Capability 2” } 例如，docker run --interactive --tty --cap-drop = all --cap-add = { “NET_ADMIN”， “SYS_ADMIN” } centos：latest / bin / bash
影响	基于添加或删除的 Linux 内核功能，容器中功能会受到限制。
默认值	默认情况下，以下功能可用于容器：AUDIT_WRITE CHOWN DAC_OVERRIDE FOWNER FSETID KILL MKNOD NET_BIND_SERVICE NET_RAW SETFCAP SETGID SETPCAP SETUID SYS_CHROOT

参考文献	1.https://docs.docker.com/engine/security/security/#linux-kernel-capabilities 2. http://man7.org/linux/man-pages/man7/capabilities.7.html 3. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf

5.4 不使用特权容器

描述	使用--privileged 标志将所有 Linux 内核功能提供给容器，从而覆盖--cap-add 和-cap-drop 标志。 若无必须请不要使用它。
安全出发点	--privileged 标志给容器提供所有功能,并且还提升了 cgroup 控制器执行的所有限制。 换句话说,容器可以做几乎主机可以做的一切。 这个标志存在允许特殊用例,就像在 Docker 中运行 Docker 一样。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Privileged={{ .HostConfig.Privileged }}'
结果判定	上述命令应为每个容器实例返回 Privileged = false。
修复措施	不要运行带有--privileged 标志的容器。 例如，不要启动如下容器：docker run --interactive --tty --privileged centos / bin / bash
影响	除默认值之外的 Linux 内核功能将无法在容器内使用。
默认值	False.
参考文献	*1. https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities*

5.5 敏感的主机系统目录未挂载在容器上

描述	不应允许将敏感的主机系统目录（如下所示）作为容器卷进行挂载，特别是在读写模式下。 / / boot / dev / etc / lib / proc / sys / usr
安全出发点	如果敏感目录以读写方式挂载，则可以对这些敏感目录中的文件进行更改。 这些更改可能会降低安全性，且直接影响 Docker 宿主机。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Volumes={{ .Mounts }}'
结果判定	上述命令将返回当前映射目录的列表，以及是否以每个容器实例的读写模式进行挂载。
修复措施	不要将主机敏感目录挂载在容器上，尤其是在读写模式下。
影响	None.
默认值	Docker 默认为读写卷，也可以以只读方式挂载一个目录。 默认情况下，不会在容器上挂载敏感的主机目录。
参考文献	*1. https://docs.docker.com/engine/tutorials/dockervolumes/*

5.6 SSH 不在容器中运行

描述	SSH 服务不应该在容器内运行。
安全出发点	在容器内运行 SSH 可以增加安全管理的复杂性 难以管理 SSH 服务器的访问策略和安全合规性 难以管理各种容器的密钥和密码 难以管理 SSH 服务器的安全升级 可以在不使用 SSH 的情况下对容器进行 shell 访问，避免不必要地增加安全管理的复杂性。
审计方法	步骤 1：通过执行以下命令列出所有运行的容器实例： docker ps --quiet 步骤 2：对于每个容器实例，执行以下命令： docker exec $ INSTANCE_ID ps -el
结果判定	确保没有 SSH 服务
修复措施	从容器中卸载 SSH 服务器，并使用 nsenter 或其他任何命令（如 docker exec 或 docker attach）与容器实例进行交互。 docker exec --interactive --tty $ INSTANCE_ID sh OR docker attach $INSTANCE_ID
影响	None.
默认值	默认情况下，SSH 服务不在容器内运行。 每个容器只允许一个进程。
参考文献	*1. http://blog.docker.com/2014/06/why-you-dont-need-to-run-sshd-in-docker/*

5.7 特权端口禁止映射到容器内

描述	低于 1024 的 TCP / IP 端口号被认为是特权端口。 由于各种安全原因，普通用户和进程不允许使用它们。
安全出发点	默认情况下，如果用户没有明确声明容器端口进行主机端口映射，Docker 会自动地将容器端口映射到主机上的 49153-65535 中。 但是，如果用户明确声明它，Docker 可以将容器端口映射到主机上的特权端口。 这是因为容器使用不限制特权端口映射的 NET_BIND_SERVICE Linux 内核功能来执行。 特权端口接收和发送各种敏感和特权的数据。允许 docker 使用它们可能会带来严重的影响。
审计方法	通过执行以下命令列出容器的所有运行实例及其端口映射： docker ps --quiet | xargs docker inspect --format'{{.Id}}：Ports = {{.NetworkSettings.Ports}}'
结果判定	查看列表，并确保容器端口未映射到低于 1024 的主机端口号。
修复措施	启动容器时，不要将容器端口映射到特权主机端口。 另外，确保没有容器在 Docker 文件中特权端口映射声明。
影响	None.
默认值	默认情况下，允许将容器端口映射到主机上的特权端口。 注意：有些端口是必须使用的 例如：HTTP 和 HTTPS 必须绑定 80 / tcp 和 443 / tcp。
参考文献	*1. https://docs.docker.com/engine/userguide/networking/*

5.8 只映射必要的端口

描述	容器镜像的 Dockerfile 定义了在容器实例上默认要打开的端口。 端口列表可能与在容器内运行的应用程序相关。
安全出发点	一个容器可以运行在 Docker 文件中为其镜像定义的端口，也可以任意传递运行时参数以打开一个端口列表。 此外，Docker 文件可能会进行各种更改，暴露的端口列表可能与在容器内运行的应用程序不相关。推荐做法是不要打开不需要的端口。
审计方法	通过执行以下命令列出容器的所有运行实例及其端口映射：docker ps --quiet | xargs docker inspect --format'{{.Id}}：Ports = {{.NetworkSettings.Ports}}'
结果判定	查看列表，并确保映射的端口是容器真正需要的端口。
修复措施	修复容器镜像的 Dockerfile，以便仅通过容器化应用程序公开所需的端口。 也可以通过在启动容器时不使用-P（UPPERCASE）或--publish-all 标志来完全忽略 Dockerfile 中定义的端口列表。 使用-p（小写）或--publish 标志来明确定义特定容器实例所需的端口。 例如，docker run --interactive --tty --publish 5000 - publish 5001 - publish 5002 centos / bin / bash
影响	None.
默认值	默认情况下，当使用'-P'或'--publish-all'标志运行容器时，打开在 EXPOSE 指令下的 Dockerfile 中列出的所有端口。
参考文献	*1. https://docs.docker.com/engine/userguide/networking/*

5.9 不共享主机的网络命名空间

描述	当设置为 “--net = host” 时，容器上的网络模式将容器放置在单独的网络堆栈中。这个选择告诉 Docker 不使用 docker 内部网络，那就意味着容器在可以完全访问主机的网络接口。
安全出发点	这有一定的安全风险， 允许容器进程像任何其他 root 进程一样打开低端端口。 还允许容器访问 Docker 主机上的 D-bus 等网络服务。 因此，容器进程可以潜在地执行恶意的事，例如关闭 Docker 主机。 若无需要不要使用 host 模式。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: NetworkMode={{ .HostConfig.NetworkMode }}'
结果判定	上述命令返回 NetworkMode = host，则意味着在启动容器时传递了--net = host 选项。
修复措施	启动容器时不要通过'--net = host'选项。
影响	None.
默认值	默认情况下，容器连接到 Docker 网桥。
参考文献	*1. https://docs.docker.com/engine/userguide/networking/* *2. https://docs.docker.com/engine/reference/run/#network-settings*

5.10 确保容器的内存使用合理

描述	默认情况下，Docker 主机上的所有容器均等共享资源。 通过使用 Docker 主机的资源管理功能，例如内存限制，您可以控制容器可能消耗的内存量。
安全出发点	默认情况下，容器可以使用主机上的所有内存。 您可以使用内存限制机制来防止由于一个容器消耗了所有主机资源而导致拒绝服务，以致同一主机上的其他容器无法执行预期功能。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Memory={{ .HostConfig.Memory }}'
结果判定	如果上述命令返回 0，则表示内存无限制。 如果上述命令返回非零值，则表示已有内存限制策略。
修复措施	建议使用--memory 参数运行容器。 例如，可以运行一个容器如下： docker run --interactive --tty --memory 256m centos / bin / bash 在上面的示例中，容器启动时的内存限制为 256 MB。 值得注意的是，如果存在内存限制，下面命令的输出将以科学计数形式返回值。 docker inspect --format ='{{。Config.Memory}}'7c5a2d4c7fe0 例如，如果上述容器实例的内存限制设置为 256 MB，则上述命令的输出将为 2.68435456e + 08 而不是 256m。 应该使用科学计算器或编程方法来转换此值。
影响	如果您有设置适当的限制，容器可能将无法使用。
默认值	默认情况下，Docker 主机上的所有容器均可共享资源。 没有执行内存限制。
参考文献	*1.https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/* *2.https://docs.docker.com/engine/reference/commandline/run/#options* *3. https://docs.docker.com/engine/admin/runmetrics/*

5.11 正确设置容器上的 CPU 优先级

描述	默认情况下，Docker 主机上的所有容器均可共享资源。 通过使用 Docker 主机的资源管理功能（如 CPU 共享），可以控制容器可能占用的主机 CPU 资源。
安全出发点	默认情况下，CPU 时间在容器间平均分配。 如果需要，为了控制容器实例之间的 CPU 时间，可以使用 CPU 共享功能。 CPU 共享允许将一个容器优先于另一个容器，并禁止较低优先级的容器更频繁占用 CPU 资源。 可确保高优先级的容器更好地运行。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: CpuShares={{ .HostConfig.CpuShares }}'
结果判定	如果上述命令返回 0 或 1024，则表示 CPU 无限制。 如果上述命令返回非 1024 值以外的非零值，则表示 CPU 已经限制。
修复措施	管理容器之间的 CPU 份额。为此，请使用--cpu-shares 参数启动容器。 例如，可以运行一个容器，如下所示： docker run --interactive --tty --cpu-shares 512 centos / bin / bash 在上面的示例中，容器以其他容器使用的 50%的 CPU 份额启动。因此，如果另一个容器的 CPU 份额为 80%，则此容器的 CPU 份额为 40%。 注意：默认情况下，每个新容器将拥有 1024 个 CPU 份额。但是，如果运行审计部分中提到的命令，则此值显示为 0。 或者使用如下方法： 1.进入/sys/fs/cgroup/cpu/system.slice/目录。 2.使用 docker ps 检查容器实例 ID。 3.在上面的目录中（在步骤 1 中），将有一个名称为 docker- <Instance ID> .scope 的目录。例如，docker-4acae729e8659c6be696ee35b2237cc1fe4edd2672e9186434c5116e1a6fbed6.scope。进入此目录。 4.可以找到一个名为 cpu.shares 的文件。执行 cat cpu.shares。可以看到 CPU 的份额值。因此，在 docker run 命令中没有使用-c 或--cpu-shares 参数配置 CPU 共享，该文件的值为 1024。

	如果我们将一个容器的 CPU 份额设置为 512，则与其他容器相比，它将获得一半的 CPU 时间。因此，以 1024 作为 100%，然后快速计算数据以得出您应为各个 CPU 份额设置的数字。例如，如果您想设置 50%，则使用 512;如果您想设置 25%，则使用 256。
影响	如果没有设置适当的 CPU 共享，容器进程可能会不能执行。 但主机上的 CPU 资源是空闲的，CPU 共享不会对容器可能使用的 CPU 造成任何限制。
默认值	默认情况下，Docker 主机上的所有容器均可共享资源。 没有执行 CPU 份额。
参考文献	*1. https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/* *2. https://docs.docker.com/engine/reference/commandline/run/#options* *3. https://docs.docker.com/engine/admin/runmetrics/*

5.12 设置容器的根文件系统为只读

描述	通过使用 Docker 运行的只读选项，容器的根文件系统应被视为“黄金镜像”。 这样可以防止在容器运行时写入容器的根文件系统。
安全出发点	启用此选项会迫使运行时的容器明确定义其数据写入策略，可减少安全风险，因为容器实例的文件系统不能被篡改或写入，除非它对文件系统文件夹和目录具有明确的读写权限。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: ReadonlyRootfs={{ .HostConfig.ReadonlyRootfs }}'
结果判定	如果上述命令返回 true，则表示容器的根文件系统是只读的。 如果上述命令返回 false，则意味着容器的根文件系统是可写的。
修复措施	在容器的运行时添加一个只读标志以强制容器的根文件系统以只读方式装入。 docker run <Run arguments> - read-only <Container Image Name or ID> <Command> 在容器的运行时启用只读选项，包括但不限于如下： 1.使用--tmpfs 选项为非持久数据写入临时文件系统。 docker run --interactive --tty --read-only --tmpfs “/ run” --tmpfs “/ tmp” centos / bin / bash 2.启用 Docker rw 在容器的运行时载入，以便将容器数据直接保存在 Docker 主机文件系统上。 docker run --interactive --tty --read-only -v / opt / app / data：/ run / app / data：rw centos / bin / bash 3.使用 Docker 共享存储卷插件来存储 Docker 数据卷以保留容器数据。 docker volume create -d convoy --opt o = size = 20GB my-named-volume docker run --interactive --tty --read-only -v my-named-volume：/ run / app / data centos / bin /bash 4.在容器运行期间，将容器数据传输到容器外部，以便保持容器数据。包括托管数据库，网络文件共享和 API。
影响	如果未定义数据写入策略，则在容器运行时启用--read-only 可能会破坏某些容器软件包。 定义容器的数据应该在运行时保持不变，以确定要使用哪个建议过程。

	例： ·启用--tmpfs 将临时文件写入到/ tmp ·使用 Docker 共享数据卷进行持久数据写入
默认值	默认情况下，容器的根文件系统是可写的，允许所有容器进程写入容器运行时产生的文件。
参考文献	*1. http://docs.docker.com/reference/commandline/cli/#run* *2. https://docs.docker.com/engine/tutorials/dockervolumes/* *3.http://www.projectatomic.io/blog/2015/12/making-docker-images-write-only-in-production/* *4.https://docs.docker.com/engine/reference/commandline/run/#mount-tmpfs-tmpfs* *5.https://docs.docker.com/engine/tutorials/dockervolumes/#creating-and-mounting-a-data-volume-container*

5.13 确保进入容器的流量绑定到特定的主机接口

描述	默认情况下，Docker 容器可以连接到外部，但外部无法连接到容器。每个传出连接都源自主机自己的 IP 地址。 所以只允许通过主机上的特定外部接口访问容器服务。
安全出发点	如果主机上有多个网络接口，则容器可以接受任何网络接口上公开端口的连接。 这可能不安全。 很多时候，特定的端口暴露在外部，并且在这些端口上运行诸如入侵检测，入侵防护，防火墙，负载均衡等服务以筛选传入的公共流量。 因此，只允许来自特定外部接口的传入连接。
审计方法	通过执行以下命令列出容器的所有运行实例及其端口映射：docker ps --quiet | xargs docker inspect --format'{{.Id}}：Ports = {{.NetworkSettings.Ports}}'
结果判定	查看列表并确保公开的容器端口与特定接口绑定，而不是通配符 IP 地址 - 0.0.0.0。 例如，如果上述命令如下返回，那么这是不安全的，并且容器可以接受指定端口 49153 上的任何主机接口上的连接。 Ports = map [443 / tcp：<nil> 80 / tcp：[map [HostPort：49153 HostIp：0.0.0.0]]] 如果将暴露端口连接到主机上的特定接口（如下所示），建议将根据需要进行配置。 Ports = map [443 / tcp：<nil> 80 / tcp：[map [HostIp：10.2.3.4 HostPort：49153]]]
修复措施	将容器端口绑定到所需主机端口上的特定主机接口。 例如，docker run--detach --publish 10.2.3.4:49153:80 nginx 在上面的示例中，容器端口 80 绑定到 49153 上的主机端口，并且仅接受来自 10.2.3.4 外部接口的传入连接。
影响	None.
默认值	默认情况下，Docker 将容器端口公开在 0.0.0.0，可接受主机上任何可能的传入网络端口。
参考文献	*1. https://docs.docker.com/engine/userguide/networking/*

5.14 容器重启策略 on-failure 设置为 5

描述	在 docker run 命令中使用--restart 标志，可以指定重启策略，以便在退出时确定是否重启容器。 基于安全考虑，应该设置重启尝试次数限制为 5 次。
安全出发点	如果无限期地尝试启动容器，可能会导致主机上的拒绝服务。 这可能是一种简单的方法来执行分布式拒绝服务攻击，特别是在同一主机上有多个容器时。 此外，忽略容器的退出状态并始终尝试重新启动容器导致未调查容器终止的根本原因。 如果一个容器被终止，应该做的是去调查它重启的原因，而不是试图无限期地重启它。 因此，建议使用故障重启策略并将其限制为最多 5 次重启尝试。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: RestartPolicyName={{ .HostConfig.RestartPolicy.Name }} MaximumRetryCount={{ .HostConfig.RestartPolicy.MaximumRetryCount }}'
结果判定	如果上述命令返回 RestartPolicyName = always，那么系统没有按需要进行配置。 如果上述命令返回 RestartPolicyName = no 或仅 RestartPolicyName =，则重新启动策略未被使用，容器不会重新启动。 如果上述命令返回 RestartPolicyName = on-failure，则通过查看 MaximumRetryCount 验证重新启动尝试的次数是否设置为 5 或更少。
修复措施	如果一个容器需要自己重新启动，可以如下设置： docker run --detach --restart = on-failure：5 nginx
影响	容器只会尝试重新启动 5 次。
默认值	默认情况下，容器未配置重新启动策略。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/run/#restart-policies-restart*

5.15 确保主机的进程命名空间不共享

描述	进程 ID（PID）命名空间隔离进程 ID 空间，这意味着不同 PID 命名空间中的进程可以具有相同的 PID。 这就是容器和主机之间的进程级隔离。
安全出发点	PID 名称空间提供了进程的隔离。 PID 命名空间删除了系统进程的视图，并允许重用包括 PID 的进程 ID。如果主机的 PID 名称空间与容器共享，它基本上允许容器内的进程查看主机上的所有进程。 这就打破了主机和容器之间进程级别隔离的优点。 若访问容器最终可以知道主机系统上运行的所有进程，甚至可以从容器内杀死主机系统进程。 这可能是灾难性的。 因此，不要将容器与主机的进程名称空间共享。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: PidMode={{ .HostConfig.PidMode }}'
结果判定	如果上述命令返回 host，则表示主机 PID 名称空间与容器共享，存在安全风险。
修复措施	不要使用'--pid = host'参数启动容器。 例如，不要启动一个容器，如下所示： docker run --interactive --tty --pid = host centos / bin / bash
影响	容器进程无法看到主机系统上的进程。 在某些情况下，可能需要容器共享主机的进程命名空间。 例如，可以使用像 strace 或 gdb 这样的调试工具构建容器，在调试容器中的进程时要使用这些工具。 如必要，最好只能使用 “-p” 参数共享必须的主机进程。 例如，docker run --pid = host rhel7 strace -p 1234
默认值	默认情况下，所有容器都启用了 PID 命名空间，并且主机的进程命名空间不与容器共享。
参考文献	*1. https://docs.docker.com/engine/reference/run/#pid-settings-pid* *2. http://man7.org/linux/man-pages/man7/pid_namespaces.7.html*

5.16 主机的 IPC 命令空间不共享

描述	IPC（POSIX / SysV IPC）命名空间提供命名共享内存段，信号量和消息队列的分离。 因此主机上的 IPC 命名空间不应该与容器共享，并且应该保持独立。
安全出发点	IPC 命名空间提供主机和容器之间的 IPC 分离。 如果主机的 IPC 名称空间与容器共享，它允许容器内的进程查看主机系统上的所有 IPC。 这打破了主机和容器之间 IPC 级别隔离的好处。 可通过访问容器操纵主机 IPC。 这可能是灾难性的。 因此，不要将主机的 IPC 名称空间与容器共享。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: IpcMode={{ .HostConfig.IpcMode }}'
结果判定	如果上述命令返回 host，则意味着主机 IPC 命名空间与容器共享。 如果上述命令不返回任何内容，则主机的 IPC 命名空间不会共享。
修复措施	不要使用'--ipc = host'参数启动容器。 例如，不要启动如下容器： docker run --interactive --tty --ipc = host centos / bin / bash
影响	共享内存段用于加速进程间通信。 它通常被高性能应用程序使用。 如果这些应用程序被容器化为多个容器，则可能需要共享容器的 IPC 名称空间以实现高性能。 在这种情况下，您仍然应该共享容器特定的 IPC 命名空间而不是整个主机 IPC 命名空间。 可以将容器的 IPC 名称空间与另一个容器共享，如下所示： 例如，docker run --interactive --tty --ipc = container：e3a7a1a97c58 centos / bin / bash
默认值	默认情况下，所有容器都启用 IPC 命名空间，并且主机 IPC 命名空间不与任何容器共享。
参考文献	*1. https://docs.docker.com/engine/reference/run/#ipc-settings-ipc* *2. http://man7.org/linux/man-pages/man7/namespaces.7.html*

5.17 主机设备不直接共享给容器

描述	主机设备可以在运行时直接共享给容器。 不要将主机设备直接共享给容器，特别是对不受信任的容器。
安全出发点	选项--device 将主机设备共享给容器，因此容器可以直接访问这些主机设备。 不允许容器以特权模式运行以访问和操作主机设备。 默认情况下，容器将能够读取，写入和 mknod 这些设备。 此外，容器可能会从主机中删除设备。 因此，不要直接将主机设备共享给容器。 如果必须的将主机设备共享给容器，请适当地使用共享权限： r - read w - write m - mknod allowed
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Devices={{ .HostConfig.Devices }}'
结果判定	以上命令将列出每个设备的以下信息： ·CgroupPermissions - 例如，rwm ·PathInContainer - 容器内的设备路径 ·PathOnHost - 主机上的设备路径 验证是否需要从容器中访问主机设备，并且正确设置所需的权限。 如果上述命令返回[ ]，则容器无权访问主机设备。
修复措施	不要将主机设备直接共享于容器。 如果必须将主机设备共享给容器，请使用正确的一组权限： 例如，不要启动一个容器，如下所示： docker run --interactive --tty --device = / dev / tty0：/ dev / tty0：rwm --device = / dev / temp_sda：/ dev / temp_sda：rwm centos bash 例如，以正确的权限共享主机设备： docker run --interactive --tty --device = / dev / tty0：/ dev / tty0：rw --device = / dev / temp_sda：/ dev / temp_sda：r centos bash
影响	将无法直接在容器内使用主机设备。
默认值	默认情况下，主机设备不共享于容器。 如果您不提供共享权限并选择将主机设备展示给容器，则主机设备将具有读取，写入和 mknod 权限。

参考文献	1.https://docs.docker.com/engine/reference/commandline/run/#options

5.18 设置默认的 ulimit 配置（在需要时）

描述	默认的 ulimit 是在 Docker 守护程序级别设置的。如果需要，您可以在容器运行时重写默认的 ulimit 设置。
安全出发点	ulimit 提供对 shell 可用资源的控制。 设置系统资源控制可以防止资源耗尽带来的问题，如 fork 炸弹。 有时候合法的用户和进程也可能过度使用系统资源，导致系统资源耗尽 应该遵守在 Docker 守护程序级别设置的默认 ulimit。 如果默认的 ulimit 设置不适合特定的容器实例，则可以将它们覆盖为例外。 但是尽量不要这样做。 如果有太多例外的话，可以直接修改默认的 ulimit 设置。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Ulimits={{ .HostConfig.Ulimits }}'
结果判定	对于每个容器实例，上述命令应该返回 Ulimits = <no value>，除非出现异常并且需要覆盖默认的 ulimit 设置。
修复措施	如果需要，覆盖默认的 ulimit 设置。 例如，要覆盖默认的 ulimit 设置，请启动一个容器，如下所示： docker run --ulimit nofile = 1024：1024 --interactive --tty centos / bin / bash
影响	如果 ulimits 未正确设置，则可能无法实现所需的资源控制，甚至导致系统无法使用。
默认值	容器实例继承在 Docker 守护程序级别设置的默认 ulimit 设置。
参考文献	*1. https://docs.docker.com/engine/reference/commandline/run/#set-ulimits-in-container-ulimit* *2. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf*

5.19 设置装载传播模式不共享

描述	装载传播模式允许在容器上以 shared、slave 和 private 模式挂载数据卷。 只有必要的时候才使用共享模式。
安全出发点	共享模式下挂载卷不会限制任何其他容器的安装并对该卷进行更改。如果使用的数据卷对变化比较敏感，则这可能是灾难性的。 最好不要将安装传播模式设置为共享。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Propagation={{range $mnt := .Mounts}} {{json $mnt.Propagation}} {{end}}'
结果判定	上述命令将返回已安装卷的传播模式。 除非需要，不应将传播模式设置为共享。。
修复措施	不建议以共享模式传播中安装卷。 例如，不要启动容器，如下所示：docker run <Run arguments> --volume = / hostPath：/ containerPath：shared <Container Image Name or ID> <Command>
影响	None.
默认值	默认情况下，容器装载是私有的。
参考文献	*1. https://github.com/docker/docker/pull/17034* *2.https://docs.docker.com/engine/reference/run/#volume-shared-filesystems* *3.https://www.kernel.org/doc/Documentation/filesystems/sharedsubtree.txt*

5.20 设置主机的 UTS 命令空间不共享

描述	UTS 命名空间提供两个系统标识符的隔离：主机名和 NIS 域名。 它用于设置在该名称空间中运行进程可见的主机名和域名。 在容器中运行的进程通常不需要知道主机名和域名。 因此，名称空间不应与主机共享。
安全出发点	与主机共享 UTS 命名空间提供了容器可更改主机的主机名。 这是不安全的。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: UTSMode={{ .HostConfig.UTSMode }}'
结果判定	如果上述命令返回 host，则意味着主机 UTS 名称空间与容器共享，是不符合要求的。 如果上述命令不返回任何内容，则主机的 UTS 名称空间不共享。
修复措施	不要使用'--uts = host'参数启动容器。 例如，不要启动如下容器：docker run --rm --interactive --tty --uts = host rhel7.2
影响	None.
默认值	默认情况下，所有容器都启用了 UTS 命名空间，并且主机 UTS 命名空间不与任何容器共享。
参考文献	*1. https://docs.docker.com/engine/reference/run/#uts-settings-uts* *2. http://man7.org/linux/man-pages/man7/namespaces.7.html*

5.21 默认的 seccomp 配置文件未禁用

描述	Seccomp 为容器运行时的系统调用提供了额外的保护。 默认的 Docker seccomp 配置文件以白名单为基础，允许了系统调用。，不应该禁用。
安全出发点	大量系统调用暴露于每个用户级进程，其中许多系统调用在整个生命周期中都未被使用。 大多数应用程序不需要所有的系统调用，因此可以通过减少可用的系统调用来增加安全性。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: SecurityOpt={{ .HostConfig.SecurityOpt }}'
结果判定	上面的命令应该返回<no value>或修改的 seccomp 配置文件。 如果它返回[seccomp：unconfined]，这意味着是不安全的，表明容器运行时没有任何 seccomp 配置文件。
修复措施	默认情况下，docker 启用 seccomp 配置文件。 除非要修改和使用修改的 seccomp 配置文件，否则不需要执行任何操作。
影响	使用 Docker 1.10 或更高版本，默认的 seccomp 配置文件会阻止部分系统调用，而不管传递给容器的--cap-add。 在这种情况下，应创建自定义 seccomp 配置文件。 可以通过在 docker run 上传递--security-opt = seccomp：unconfined 来禁用默认的 seccomp 配置文件。
默认值	运行容器时，它将使用默认配置文件，除非您使用--security-opt 选项覆盖它。
参考文献	*1. http://blog.scalock.com/new-docker-security-features-and-what-they-mean-seccomp-profiles* *2. https://docs.docker.com/engine/reference/run/#security-configuration* *3.https://github.com/docker/docker/blob/master/profiles/seccomp/default.json* *4. https://docs.docker.com/engine/security/seccomp/* *5. https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt* *6. https://github.com/docker/docker/issues/22870*

5.22docker exec 命令不能使用特权选项

描述	不要使用--privileged 选项来执行 docker exec

安全出发点	在 docker exec 中使用--privileged 选项可为命令提供扩展的 Linux 功能。 这可能会造成不安全的情况。
审计方法	如果按照第 1 部分的规定启用了审计，则可以使用以下命令过滤掉使用--privileged 选项的 docker exec 命令。 ausearch -k docker | grep exec | grep 特权
结果判定	无--privileged 选项
修复措施	在 docker exec 命令中不要使用--privileged 选项。
影响	如果您需要容器内的增强功能，则只运行具有所需功能的容器。
默认值	默认情况下，docker exec 命令不使用--privileged 选项运行。
参考文献	*1. https://docs.docker.com/engine/reference/commandline/exec/*

5.23docker exec 命令不能与 user 选项一起使用

描述	不要使用--user 选项执行 docker exec。
安全出发点	在 docker exec 中使用--user 选项以该用户身份在容器内执行该命令。这可能会造成不安全的情况。 例如，假设你的容器是以 tomcat 用户（或任何其他非 root 用户）身份运行的，那么可以使用--user = root 选项以 root 用户身份运行命令。这是非常危险的。
审计方法	如果您按照第 1 部分的规定启用了审计，则可以使用以下命令过滤掉使用了--user 选项的 docker exec 命令。 ausearch -k docker | grep exec | grep user
结果判定	无--user 选项
修复措施	在 docker exec 命令中不要使用--user 选项。
影响	None.
默认值	默认情况下，docker exec 命令不使用--user 选项运行。
参考文献	*1. https://docs.docker.com/engine/reference/commandline/exec/*

5.24 确保 cgroug 安全使用

描述	可以在容器运行时附加到特定的 cgroup。 确认 cgroup 的使用将确保容器在定义的 cgroup 下运行。
安全出发点	系统管理员通常定义容器应该在哪个 cgroup 下运行。 即使 cgroups 没有被系统管理员明确定义，容器默认情况下仍会在 docker cgroup 下运行。在运行时，可以连接到除默认使用的 cgroup 以外的其他 cgroup。通过附加到与默认不同的 cgroup，可能会向容器授予额外的权限和资源，可能导致安全问题
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: CgroupParent={{ .HostConfig.CgroupParent }}'
结果判定	上述命令将返回运行容器的 cgroup。 如果为空，则表示容器在默认的 docker cgroup 下运行。可以认为是安全的。 如果发现容器在 cgroup 之外运行，则可能存在安全问题。
修复措施	如无必须，不要使用 --cgroup-parent 选项在 docker 运行
影响	None.
默认值	默认情况下，容器在 docker cgroup 下运行。
参考文献	*1.https://docs.docker.com/engine/reference/run/#specify-custom-cgroups* *2.https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html*

5.25 限制容器获得额外的权限

描述	限制容器通过 suid 或 sgid 位获取额外的权限。
安全出发点	进程可以在内核中设置 no_new_priv 位。 它可以 fork、clone 和 execute。 no_new_priv 位确保进程或其子进程不通过 suid 或 sgid 位获取任何其他权限。 这样一来，很多危险的操作变得不那么危险。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: SecurityOpt={{ .HostConfig.SecurityOpt }}'
结果判定	上述命令应返回当前为容器配置的所有安全选项。
修复措施	例如，应该按如下启动容器：docker run --rm -it --security-opt = no-new-privileges ubuntu bash
影响	no_new_priv 会阻止像 SELinux 这样的 LSM 变为不允许访问当前进程的进程标签。
默认值	默认情况下，新的权限不受限制。
参考文献	*1. https://github.com/projectatomic/atomic-site/issues/269* *2. https://github.com/docker/docker/pull/20727* *3. https://www.kernel.org/doc/Documentation/prctl/no_new_privs.txt* *4. https://lwn.net/Articles/475678/* *5. https://lwn.net/Articles/475362/*

5.26 检查容器运行时状态

描述	如果容器镜像没有定义 HEALTHCHECK 指令，请在容器运行时使用--health-cmd 参数来检查容器运行状况。
安全出发点	可用性是安全一个重要特性。 如果您用的容器镜像没有预定义的 HEALTHCHECK 指令，请使用--health-cmd 参数在运行时检查容器运行状况。根据报告的健康状况，可以采取必要的措施。
审计方法	运行以下命令并确保所有容器都报告运行状况：docker ps --quiet | xargs docker inspect --format'{{.Id}}：Health = {{.State.Health.Status}}'
结果判定	确保使用使用--health-cmd 参数
修复措施	使用--health-cmd 和其他参数运行容器。 例如，docker run -d --health-cmd ='stat / etc / passwd || exit1'nginx
影响	None.
默认值	默认情况下，运行状况检查不会在容器运行时完成。
参考文献	*1. https://docs.docker.com/engine/reference/run/#healthcheck*

5.27 确保 docker 命令始终获取最新版本的镜像

描述	始终确保您在存储库中使用最新版本的镜像，而不是缓存的旧版本。
安全出发点	已知多个 docker 命令（如 docker pull，docker run 等）有一个问题，即默认情况下，它们会提取镜像的本地副本（如果存在），即使存在具有“相同标记”镜像的更新版本在仓库中。
审计方法	第 1 步：打开镜像库并列出正在检查的镜像的镜像版本历史记录。 步骤 2：观察 docker pull 命令触发时的状态。 如果状态显示为镜像为最新，则表示您正在获取镜像的缓存版本。 步骤 3：将正在运行的镜像版本与存储库中报告的最新版本进行匹配，以判断运行缓存版本还是最新版本。
结果判定	
修复措施	使用正确的版本锁定机制（默认情况下分配的最新标签仍然不完善），避免提取缓存的旧版本。 版本固定机制也应用于基本镜像，软件包和整个镜像。 可以根据要求定制版本固定规则。
影响	None
默认值	默认情况下，docker 命令提取本地副本。
参考文献	*1. https://github.com/docker/docker/pull/16609*

5.28 限制使用 PID cgroup

描述	在容器运行时使用--pids-limit 标志。
安全出发点	攻击者可以在容器内发射 fork 炸弹。 这个 fork 炸弹可能会使整个系统崩溃，并需要重新启动主机以使系统重新运行。 PIDs cgroup --pids-limit 将通过限制在给定时间内可能发生在容器内的 fork 数来防止这种攻击。
审计方法	运行以下命令并确保 PidsLimit 未设置为 0 或-1。 PidsLimit 为 0 或-1 意味着任何数量的进程可以同时在容器内分叉。 docker ps --quiet | xargs docker inspect --format'{{.Id}}：PidsLimit = {{.HostConfig.PidsLimit}}'
结果判定	PidsLimit 未设置为 0 或-1
修复措施	在启动容器时，使用--pids-limit 标志。例如，docker run -it -pids-limit 100 <Image_ID> 在上述示例中，允许在任何给定时间运行的进程数设置为 100.在达到 100 个并发运行进程的限制之后，docker 将限制任何新的进程创建。
影响	根据需要设置 PID 限制值。 不正确的值可能会使容器不能使用。
默认值	通常 pids-limit 的默认值为 0,这意味着对 fork 数量没有限制。 另外请注意,pids cgroup 限制仅适用于内核版本 4.3+。
参考文献	*1. https://github.com/docker/docker/pull/18697* *2.https://docs.docker.com/engine/reference/commandline/run/#options*

5.29 不要使用 docker 的默认网桥 docker0

描述	不要使用 Docker 的默认 bridge docker0。 使用 docker 的用户定义的网络进行容器联网。
安全出发点	Docker 将以桥模式创建的虚拟接口连接到名为 docker0 的公共桥。 这种默认网络模型易受 ARP 欺骗和 MAC 洪泛攻击的攻击，因为没有应用过滤。
审计方法	运行以下命令，并验证容器是否在用户定义的网络上，而不是默认的 docker0 网桥。 Docker network ls --quiet | xargs xargs docker network inspect -format'{{.Name}}：{{.Options}}'
结果判定	使用 bridge docker0
修复措施	遵循 Docker 文档并设置用户定义的网络。 运行定义的网络中的所有容器。
影响	必须管理用户定义的网络。
默认值	默认情况下，docker 在其 docker0 桥上运行容器。
参考文献	*1. https://github.com/nyantec/narwhal* *2. https://arxiv.org/pdf/1501.02967* *3. https://docs.docker.com/engine/userguide/networking/*

5.30 不共享主机的用户命名空间

描述	不要与容器共享主机的用户名空间。
安全出发点	用户名空间确保容器内的根进程将映射到容器外部的非根进程。 因此，与容器共享主机的用户名空间不会使主机上的用户与容器上的用户隔离。
审计方法	运行以下命令，并确保它不返回 UsernsMode 的任何值。 如果返回主机的值，则表示主机用户命名空间与容器共享。 docker ps --quiet | xargs docker inspect --format'{{.Id}}：UsernsMode = {{.HostConfig.UsernsMode}}'
结果判定	UsernsMode 无返回值
修复措施	不要在主机和容器之间共享用户名称空间。 例如，不要像下面那样运行容器： docker run --rm -it --userns = host ubuntu bash
影响	None
默认值	默认情况下，主机用户命名空间与容器共享，直到启用用户命名空间支持。
参考文献	*1. https://docs.docker.com/engine/security/userns-remap/* *2.https://docs.docker.com/engine/reference/commandline/run/#options* *3. https://github.com/docker/docker/pull/12648* *4.https://events.linuxfoundation.org/sites/events/files/slides/User%20Namespaces%20-%20ContainerCon%202015%20-%2016-9-final_0.pdf*

5.31 任何容器内不能安装 docker 套接字

描述	docker socket（docker.sock）不应该安装在容器内。
安全出发点	如果 Docker 套接字安装在容器内，它将允许在容器内运行的进程执行 docker 命令，这有效地允许完全控制主机。
审计方法	docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Volumes={{ .Mounts }}' | grep docker.sock
结果判定	上述命令将返回 docker.sock 作为卷映射到容器的任何实例。
修复措施	确保没有容器将 docker.sock 作为卷。
影响	None
默认值	默认情况下，docker.sock 未装入容器内。
参考文献	*1.https://raesene.github.io/blog/2016/03/06/The-Dangers-Of-Docker.sock/* *2.https://forums.docker.com/t/docker-in-docker-vs-mounting-var-run-docker-sock/9450/2* *3. https://github.com/docker/docker/issues/21109*

6 docker 安全操作

6.1 避免镜像泛滥

描述	不要在同一主机上保留大量容器镜像。 根据需要仅使用标记的镜像。
安全出发点	标记镜像有助于从 “最新” 退回到生产中镜像的特定版本。 如果实例化了未使用或旧标签的镜像，则可能包含可能被利用的漏洞。 此外，如果您无法从系统中删除未使用的镜像，并且存在各种此类冗余和未使用的镜像，主机文件空间可能会变满，从而导致拒绝服务。
审计方法	步骤 1 通过执行以下命令列出当前实例化的所有镜像 ID： docker images --quiet | xargs docker inspect --format'{{.Id}}：Image = {{.Config.Image}}' 步骤 2：通过执行以下命令列出系统中存在的所有镜像：docker images 步骤 3：比较步骤 1 和步骤 2 中的镜像 ID 列表，找出当前未实例化的镜像。 如果发现未使用或旧镜像，请与系统管理员讨论是否需要在系统上保留这些镜像。
结果判定	主机不保留无用镜像
修复措施	保留您实际需要的一组镜像，并建立工作流程以从主机中删除陈旧的镜像。 此外，使用诸如按摘要的功能从镜像仓库中获取特定镜像。 可以按照以下步骤找出系统上未使用的镜像并删除它们。 步骤 1 通过执行以下命令列出当前实例化的所有镜像 ID： docker images --quiet | xargs docker inspect --format'{{.Id}}：Image = {{.Config.Image}}' 步骤 2：通过执行以下命令列出系统中存在的所有镜像：docker images 步骤 3：比较步骤 1 和步骤 2 中填充的镜像 ID 列表，找出当前未实例化的镜像。 第 4 步：决定是否要保留当前未使用的镜像。 如果不通过执行以下命令删除它们：docker rmi $ IMAGE_ID
影响	None.
默认值	镜像和分层文件系统在主机上保持可访问状态，直到管理员删除这些镜像或图层的所有标签。

参考文献	1.http://craiccomputing.blogspot.in/2014/09/clean-up-unused-docker-containers-and.html 2.https://forums.docker.com/t/command-to-remove-all-unused-images/20/8 3. https://github.com/docker/docker/issues/9054 4. https://docs.docker.com/engine/reference/commandline/rmi/ 5. https://docs.docker.com/engine/reference/commandline/pull/ 6. https://github.com/docker/docker/pull/11109

6.2 避免容器泛滥

描述	不要在同一主机上保留大量无用容器。
安全出发点	容器的灵活性使得运行多个应用程序实例变得很容易，并间接导致存在于不同安全补丁级别的 Docker 镜像。因此，避免容器泛滥，并将主机上的容器数量保持在可管理的总量上。
审计方法	步骤 1 - 查找主机上的容器总数：docker info --format'{{.Containers}}' 步骤 2 - 执行以下命令以查找主机上实际正在运行或处于停止状态的容器总数。 docker info - format'{{.ContainersStopped}}'docker info - format'{{.ContainersRunning}}'
结果判定	如果主机上保留的容器数量与主机上实际运行的容器数量之间的差异很大（比如说 25 或更多），那么请清理。
修复措施	定期检查每个主机的容器清单，并使用以下命令清理已停止的容器： docker container prune
影响	如果你每个主机的容器数量太少，那么你可能没有充分利用你的主机资源。
默认值	默认情况下，Docker 不会限制主机上可能拥有的容器数量。
参考文献	*1.https://zeltser.com/security-risks-and-benefits-of-docker-application/* *2.http://searchsdn.techtarget.com/feature/Docker-networking-How-Linux-containers-will-change-your-network*

7 docker 集群配置

7.1 不启用群集模式

描述	除非需要，不要在 Docker 引擎实例上启用群集模式。
安全出发点	默认情况下，Docker 引擎实例不会监听任何网络端口，所有与客户端的通信都将通过 Unix 套接字进行传输。 在 Docker 引擎实例上启用 Docker 群集模式时，系统上会打开多个网络端口，并使其可用于网络中的其他系统，以实现群集管理和节点通信。 打开系统上的网络端口会增加攻击风险，除非需要，不要开启群集模式。
审计方法	查看 docker info 命令的输出。 如果输出包括 Swarm：active，则表明群集模式已在 Docker 引擎上激活。 确认是否实际需要 docker 引擎实例上的 swarm 模式。
结果判定	Swarm 应该处于禁用状态
修复措施	如果在系统出错时启用了群模式，请停用
影响	None.
默认值	默认情况下，Docker 群集模式未启用。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/swarm_init/*

7.2 在群集中最小数量创建管理器节点

描述	确保在群中创建所需管理器节点的最小数量。
安全出发点	群集中的管理节点控制群集并更改其修改安全参数的配置。 拥有过多的管理节点可能会使群更容易受到损害。 如果管理节点中不需要容错功能，则应选择一个节点作为管理员。 如果需要容错，则应配置最小的实际奇数以达到适当的容错水平。
审计方法	运行 docker info 并验证管理数。 docke info - format'{{.Swarm.Managers}}' 或者运行下面的命令。 Docker node ls | grep'leader'
结果判定	管理节点需要合理
修复措施	如果配置的管理节点数量过多，则可以使用以下命令将超出部分作为节点降级： docker node demote <ID>
影响	无
默认值	一个管理节点拥有管理集群一切功能。
参考文献	*1. https://docs.docker.com/engine/swarm/manage-nodes/* *2.https://docs.docker.com/engine/swarm/admin_guide/#/add-manager-nodes-for-fault-tolerance*

7.3 群集服务绑定到特定的主机接口

描述	默认情况下，docker swarm 服务将侦听主机上的所有接口，这可能不是必需的。
安全出发点	当群集初始化时，--listen-addr 参数的默认值为 0.0.0.0:2377，这意味着群集服务将监听主机上的所有接口。 如果主机有多个网络接口，这可能不安全，因为它可能会将 docker 群集服务暴露给不参与群集操作的网络。 通过将一个特定的 IP 地址传递给--listen-addr，可以指定一个特定的网络接口来限制这种风险。
审计方法	在端口 2377 / TCP 上列出网络监听器（docker swarm 的默认设置），并确认它只监听特定的接口。 例如，使用 ubuntu 可以使用以下命令完成：netstat -lt | grep -i 2377
结果判定	只监听特定端口
修复措施	对此操作需要重新初始化群集，以指定--listen-addr 参数的特定接口。
影响	None
默认值	默认情况下，docker swarm 服务监听所有可用的主机接口。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/swarm_init/#--listen-addr* *2.https://docs.docker.com/engine/swarm/admin_guide/#recover-from-disaster*

7.4 数据在的不同节点上进行加密

描述	加密覆盖网络上不同节点上的容器之间交换的数据。
安全出发点	默认情况下，覆盖网络上不同节点上的容器之间交换的数据未加密。这可能会暴露容器节点之间的流量。
审计方法	运行以下命令并确保每个覆盖网络已被加密。 docker network ls --filter driver = overlay --quiet | xargs docker network inspect - format'{{.Name}} {{.Options}}'
结果判定	启用—opt
修复措施	使用--opt 加密标志创建网络。
影响	None
默认值	默认情况下，网络上不同节点上的容器之间交换的数据在 Docker 群集模式下未加密。
参考文献	*1.https://docs.docker.com/engine/userguide/networking/overlay-security-model/* *2. https://github.com/docker/docker/issues/24253*

7.5 管理 Swarm 集群中的涉密信息

描述	使用 Docker 的内置秘密管理命令。
安全出发点	Docker 拥有各种用于管理 Swarm 集群中秘密的命令。 这是 Docker 未来秘密支持的基础，可能会进一步改进，如对 Windows 支持等。
审计方法	在 swarm manager 节点上，运行以下命令并确保在您的环境中使用 docker 秘密管理（如果适用）。 Docker secret ls
结果判定	有秘密管理措施
修复措施	按照 docker 秘密管理方法，并用它来有效管理秘密。
影响	无
默认值	无
参考文献	*1. https://docs.docker.com/engine/reference/commandline/secret/*

7.6 swarm manager 在自动锁定模式下运行

描述	在自动锁定模式下运行 Docker 群管理器。
安全出发点	当 Docker 重新启动时，用于加密群集节点间通信的 TLS 密钥以及用于加密和解密磁盘上的 Raft 日志的密钥都会加载到每个管理器节点的内存中。 应该保护相互的 TLS 加密密钥以及用于加密和解密时的 Raft 日志的密钥。 这种保护可以通过使用--autolock 标志初始化 swarm 来启用。 使用--autolockenabled，当 Docker 重新启动时，您必须先使用 Docker 在群集初始化时生成的密钥加密密钥解锁群集。
审计方法	运行以下命令。 如果它输出密钥，则意味着 swarm 使用--autolock 标志初始化。 如果输出未设置解锁密钥，则意味着群集未使用--autolock。 docker swarm unlock-key
结果判定	是否输出秘钥
修复措施	如果正在初始化 swarm，使用下面的命令。 docker swarm init --autolock 如果想在现有的 swarm manager 节点上设置--autolock，请使用以下命令。 Docker swarm update --autolock
影响	在自动锁定模式下的群体不会从重新启动恢复，需用户手动干预以输入解锁密钥。 在某些部署中，这可能不是很方便。
默认值	默认情况下，swarm manager 不会以自动锁定模式运行。
参考文献	*1. https://docs.docker.com/engine/swarm/swarm_manager_locking/*

7.7 swarm manager 自动锁定秘钥周期性轮换

描述	定期轮换 swarm manager 自动锁定密钥。
安全出发点	Swarm 管理器自动锁定键不会自动更换。 应该定期轮换它们。
审计方法	目前，没有机制发现密钥在 swarm manager 节点上最后一次轮换的时间。 如果存在更换记录并且按预定义的频率进行了更换，应该向系统管理员确认。
结果判定	是否进行轮换自动锁定秘钥
修复措施	运行以下命令来更换。 Docker swarm unlock-key --rotate 此外，为了便于审计，维护密钥轮换记录并确保为密钥轮换符合规定周期。
影响	None
默认值	默认情况下，秘钥不会自动更换。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/swarm_unlock-key/*

7.8 节点证书适当轮换

描述	定期更换 swarm 节点证书。
安全出发点	Docker Swarm 使用相互 TLS 进行节点间的集群操作。 证书轮换可确保在受损节点密钥可能泄露及时更新。 默认情况下，节点证书每 90 天轮换一次。为了安全起见，可更频繁地或在适当的时候更换它。
审计方法	运行以下命令并确保节点证书 Expiry Duration 设置为适当。 Docker info | grep “Expiry Duration”
结果判定	证书应该按照确定的频率轮换
修复措施	运行以下命令来设置所需的到期时间。 例如，docker swarm update--cert-expiry 48h
影响	None
默认值	默认情况下，节点证书每 90 天自动轮换一次。
参考文献	*1.https://docs.docker.com/engine/reference/commandline/swarm_update/#examples*

7.9CA 根证书根据需要进行轮换

描述	根据需要轮换 CA 根证书。
安全出发点	Docker Swarm 使用相互 TLS 进行节点间的集群操作。 证书轮换可确保在受损节点密钥可能泄露及时更新。 节点证书取决于根 CA 证书。 为了操作安全性，经常轮换这些操作非常重要。 目前，根 CA 证书不会自动更换。 因此，应该建立一个周期定期更换。
审计方法	根据安装路径，检查根 CA 证书文件上的时间戳。 例如，ls -l /var/lib/docker/swarm/certificates/swarm-root-ca.crt
结果判定	证书应该按照规定的频率轮换。
修复措施	运行以下命令来更换证书。 Docker swarm ca - rotate
影响	None
默认值	默认情况下，根 CA 证书不会更换。
参考文献	*1.https://docs.docker.com/engine/swarm/how-swarm-mode-works/pki/#rotating-the-ca-certificate*