云计算开源产业联盟技术文件

|  |
| --- |
| Q/KXY CS002—2024 |

云原生安全配置基线规范V2.0

Cloud-native Security Configuration Baseline Specification V2.0

|  |
| --- |
|  |
|  |

**2024-08-09 发布 2024-08-28 实施**

云计算开源产业联盟

**Q/KXY**

目次

[目次 I](#_Toc175049617)

[前言 III](#_Toc175049618)

[1 范围 1](#_Toc175049619)

[2 规范性引用文件 1](#_Toc175049620)

[3 术语和定义 1](#_Toc175049621)

[3.1 基线控制 baseline controls 1](#_Toc175049622)

[3.2 配置管理 configuration management;CM 1](#_Toc175049623)

[4 框架概述 1](#_Toc175049624)

[5 集群编排基线要求 3](#_Toc175049625)

[5.1 API Server安全配置要求 3](#_Toc175049626)

[5.2 控制管理器安全配置要求 6](#_Toc175049633)

[5.3 调度器安全配置要求 7](#_Toc175049639)

[5.4 etcd安全配置要求 8](#_Toc175049642)

[5.5 kube-proxy安全配置要求 9](#_Toc175049648)

[5.6 Kubelet安全配置要求 9](#_Toc175049651)

[5.7 CNI插件和网络策略安全配置要求 11](#_Toc175049657)

[6 容器运行时基线要求 12](#_Toc175049660)

[6.1 docker运行时安全要求 12](#_Toc175049661)

[6.2 containerd运行时安全要求 13](#_Toc175049665)

[6.3 cri-o运行时安全要求 14](#_Toc175049670)

[7 工作负载安全配置 15](#_Toc175049674)

[7.1 启动安全 15](#_Toc175049675)

[7.2身份认证和访问控制 16](#_Toc175049676)

[7.3防止信息泄露 17](#_Toc175049677)

[7.4资源配额 17](#_Toc175049678)

[7.5其他 17](#_Toc175049679)

[8镜像安全配置 18](#_Toc175049680)

[8.1 镜像创建配置 18](#_Toc175049681)

[8.2镜像仓库安全配置 19](#_Toc175049682)

[8.3镜像部署安全配置 19](#_Toc175049683)

[附 录 A 20](#_Toc175049684)

[（资料性附录） 20](#_Toc175049685)

[**附 录 B** 40](#_Toc175049721)

[**（资料性附录）** 40](#_Toc175049722)

[**附 录 C** 50](#_Toc175049736)

[**（资料性附录）** 50](#_Toc175049737)

[附 录 D 77](#_Toc175049751)

[（资料性附录） 77](#_Toc175049752)

[参　考　文　献 79](#_Toc175049755)

|  |
| --- |
|  |

1. 前言

本文件按照GB/T 1.1-2020《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本文件由云计算开源产业联盟提出并归口。

本文件起草单位：中国信息通信研究院、华为云计算技术有限公司、阿里云计算有限公司、腾讯云计算（北京）有限责任公司、北京小佑网络科技有限公司、联通数字科技有限公司、中国联通软件研究院、兴业数字金融服务（上海）股份有限公司、中移（苏州）软件技术有限公司、浙江大学、天翼电子商务有限公司、华为技术有限公司、杭州安恒信息技术股份有限公司、北京升鑫网络科技有限公司、绿北京神州绿盟科技有限公司、北京探真科技有限公司、安易科技（北京）有限公司、北京启明星辰信息安全技术有限公司、新华三技术有限公司、中电云计算技术有限公司。

本文件主要起草人：栗蔚、陈屹力、刘如明、杜岚、仇保琪、刘方外、李有永、程艳、刘拓、匡大虎、黄竹刚、张恒、张政、刘英杰、白黎明、卢照旭、王强、赵瑞航、高鹏、孙勇仕、王鑫、梅洪彰、晏如、张潜、申文博、王旭、张兴、王超、娄宇、李淼晶、杜霖、张万兴、张李瑶、何诣莘、凌惜沫、王亮、张向雨、韩超、张暴、袁玉霦。

|  |
| --- |
|  |

云原生安全配置基线规范

2. 范围

本文件规定了云原生安全配置基线扫描应具备的基础规范要求。云原生安全配置基线扫描规范要求包括kubernetes集群安全配置、容器运行时安全配置、镜像安全配置，以及工作负载安全配置。

本文件适用于指导云原生安全Kubernetes配置基线产品建设和相关云原生安全产品基线扫描能力建设，推荐使用Kubernetes 1.23或更高版本以确保安全性。本文件同时适用于公有云、私有云和混合云服务及软件产品，依据交付形式的差异，本标准针对不同使用场景其技术指标要求也有所不同。

1. 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅所注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 25069-2022 信息安全技术 术语

GB/T 18336.1-2015 信息技术 安全技术 信息技术安全评估准则 第1部分：简介和一般模型

1. 术语和定义

GB/T 25069-2022、GB/T 18336.1-2015界定的及下列术语和定义适用于本文件。

* 1. 基线控制 baseline controls

为某一系统或组织建立的最低防护措施的集合。

[来源：GB/T 25069-2022，定义3.262]

* 1. 配置管理 configuration management;CM

应用技术和行政指导及监督的如下行为准则：识别和记录配置项的功能和物理特性，控制对这些特

性的变更，记录和报告变更处理和实施状态，并验证是否符合规定的要求。

［来源：GB/T 18336.1-2015,3.4.3，有修改：删除注等］

1. 框架概述

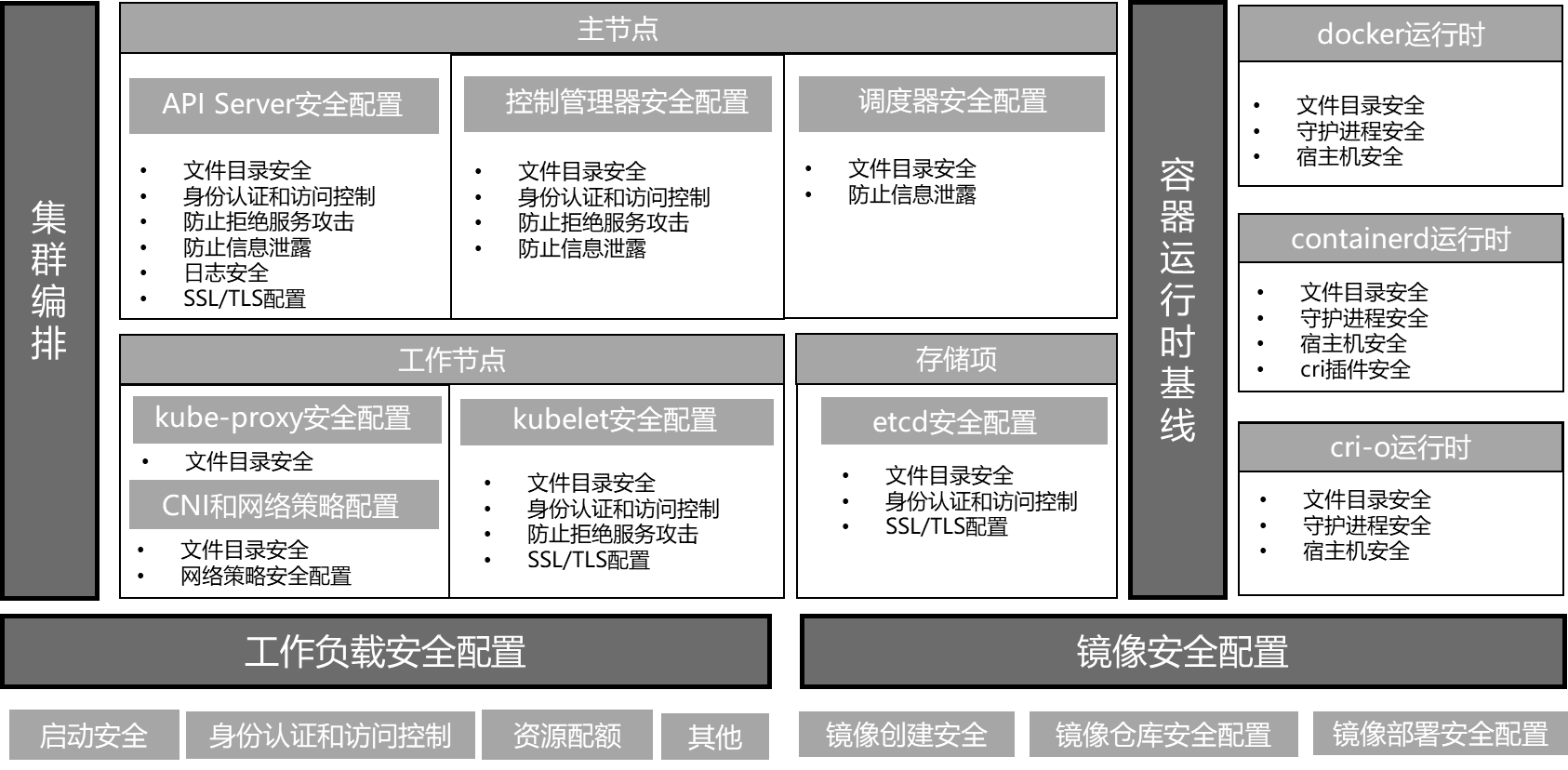
本文件中的云原生安全配置基线要求是对云原生工具Kubernetes的安全配置基线的要求，如图1所示，本文件中的要求主要包括对**Kubernetes的安全配置基线的要求、容器运行时安全配置基线的要求和工作负载的安全配置的要求**。

图1 云原生安全配置框架图

其中，Kubernetes集群的架构如图2所示，其中控制平面主要包含API Server、控制管理器 (control manager)、调度器 (scheduler)、etcd，工作节点平面主要包含kube-proxy、kubelet、工作负载。因此本文件第5章为对API Server安全配置、控制管理器安全配置、调度器安全配置、etcd安全配置、kube-proxy安全配置要求、kubelet安全配置的具体要求，附录A为第5章各配置项的检查方法和恢复方法。

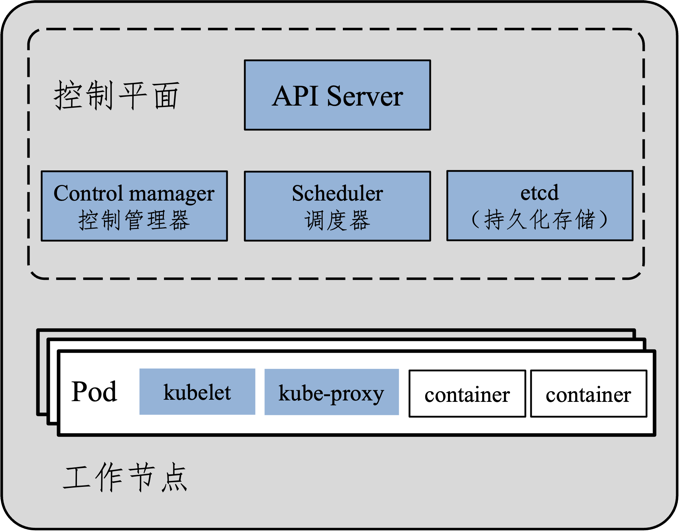


图2 Kubernetes集群架构

除了对Kubernetes集群的安全配置基线有所要求，本文件还对由Kubernetes调用的容器运行时的安全配置基线进行了严格的规范和要求。本文件第6章即为docker、containerd、cri-o三种运行时的配置基线的具体要求，附录B为第6章各配置项的检查方法和恢复方法。同时，本文件还对kubernetes环境下的工作负载安全配置基线，以及镜像安全配置基线有所要求。本文件的第7、8章为对工作负载、镜像安全配置的具体要求，附录C为第7、8章各配置项的检查方法和恢复方法。

并且为了配合自动化基线扫描工具的使用，本文件中的各项要求均附有自动化要求。自动化要求是基于工具对基线具有自动检测的能力，并可以输出为通过/失败状态，但是检测结果是否被采纳并根据检测结果进行工具设计，不在自动化能力表述的范围内，规范中不能自动化的要求是表示不能对其进行技术检测的完全自动化并输出验证结果，一般需要手动步骤来验证配置的状态是否按预期设置，预期状态可能根据环境而异。

1. 集群编排基线要求
   1. API Server安全配置要求

5.1.1.文件目录安全

apiserver的pod描述文件以及相应的PKI目录和文件，包含了组件的启动参数和证书文件，这些都是直接配置apiserver的关键文件。在创建集群的过程中，这些文件会被保留在集群中，用于后续的配置修改。鉴于这些文件的重要性，因此必须确保它们的完整性和机密性，防止它们被任意或恶意篡改。因此，对这些文件的所有权和权限的管理需要严格执行。具体要求见下：

表 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象涉及文件** | **对文件的要求** | **自动化** |
| 5.1.1.1 | apiserver | pod描述文件 | 权限设置应为600或更严格 | 是 |
| 5.1.1.2 | apiserver | pod描述文件 | 文件属主应为root:root | 是 |
| 5.1.1.3 | apiserver | Kubernetes PKI证书文件 | 权限应设置为600或更严格 | 是 |
| 5.1.1.4 | apiserver | Kubernetes PKI证书文件 | 文件所属应设置为root:root | 是 |
| 5.1.1.5 | apiserver | Kubernetes PKI密钥文件 | 权限应设置为600或更严格 | 是 |

5.1.2 身份认证和访问控制

apiserver需要进行身份认证和访问控制，其目的是为了保护集群资源的安全，其过程主要包括三个步骤：认证、授权和准入控制（主要关注资源管理）。认证和授权是限制访问集群资源的关键手段。如果集群配置不当，例如允许匿名访问，网络攻击者可以扫描常用的 Kubernetes 端口，无需认证就能访问集群的数据库或进行 API 调用。因此，需要合理配置apiserver的认证和鉴权设置，以在接收和发起请求时启用认证。同时，应启用适当的准入控制插件，以减少集群暴露的攻击面，从而保护集群的安全。具体要求见下：

表 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.1.2.1 | apiserver | --anonymous-auth | 应为false，若用户使用RBAC授权方式设定匿名用户权限进行健康检查和服务方式，通常可设置为true | 否 |
| 5.1.2.2 | apiserver | --token-auth-file | 不应设置 | 是 |
| 5.1.2.3 | apiserver | --authorization-mode | 不应设置为AlwaysAllow | 是 |
| 5.1.2.4 | apiserver | --authorization-mode | 应包含Node | 是 |
| 5.1.2.5 | apiserver | --authorization-mode | 应包含RBAC | 是 |
| 5.1.2.6 | apiserver | --service-account-lookup | 应设置为true | 是 |
| 5.1.2.7 | apiserver | --service-account-key-file | 应设置 | 是 |
| 5.1.2.8 | apiserver | --kubelet-client-certificate --kubelet-client-key | 应设置 | 是 |
| 5.1.2.9 | apiserver | --kubelet-certificate-authority | 应设置 | 是 |
| 5.1.2.10 | apiserver | --tls-cert-file --tls-private-key-file | 应设置 | 否 |
| 5.1.2.11 | apiserve | --client-ca-file | 应设置 | 是 |
| 5.1.2.12 | apiserver | --etcd-certfile和--etcd-keyfile | 应设置 | 是 |
| 5.1.2.13 | apiserver | --etcd-cafile | 应设置 | 是 |
| 5.1.2.14 | apiserver | --enable-bootstrap-token-auth | 不应设置 | 是 |
| 5.1.2.15 | 准入控制插件 | EventRateLimit | 宜设置 | 否 |
| 5.1.2.16 | 准入控制插件 | AlwaysAdmit | 不应设置 | 是 |
| 5.1.2.17 | 准入控制插件 | AlwaysPullImages | 宜设置 | 否 |
| 5.1.2.18 | 准入控制插件 | SecurityContextDeny (Kubernetes 1.25以下版本) | 如果PodSecurityPolicy没有设置，应确保设置该值 | 否 |
| 5.1.2.19 | 准入控制插件 | ServiceAccount | 应设置 | 是 |
| 5.1.2.20 | 准入控制插件 | NamespaceLifecycle | 应设置 | 是 |
| 5.1.2.21 | 准入控制插件 | NodeRestriction | 应设置 | 是 |
| 5.1.2.22 | 准入控制插件 | DenyServiceExternalIPs | 不应设置- | 是 |
| 注1：AlwaysPullImages设置可能会影响脱机或孤立的集群，这些集群采用预加载镜像方式，且无法访问镜像仓库以提取使用中的镜像，此设置不适用于使用此配置的集群。对系统复位重建、堆栈升级等时长有严苛要求的业务场景，在评估安全风险的前提下，不强制要求开启AlwaysPullImages插件配置。  注2：针对SecurityContextDeny，Kubernetes 1.25及以上版本不做要求，但应采取等同的方式：如开启 Pod Security Standards | | | | |

5.1.3 防止拒绝服务攻击

请求apiserver的超时限制。默认情况下，设置为60秒，这在连接速度较慢时可能会出现问题，一旦请求的数据量超过60秒内可传输的数据量，集群资源将无法访问。但是，将此超时限制设置得太大可能会耗尽apiserver资源，使其易于遭受拒绝服务攻击。因此，建议适当设置此限制，并仅在需要时更改默认限制60秒。

具体要求见下：

表 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.1.3.1 | apiserver | --request-timeout | 应设置 | 否 |
| 5.1.3.2 | apiserver | --max-requests-inflight | 应设置 | 否 |
| 5.1.3.3 | apiserver | --max-mutating-requests-inflight | 应设置 | 否 |

5.1.4 防止信息泄露

apiserver的不当配置会导致记录程序的详细信息（包含程序敏感数据）、使数据未加密或暴露不安全的访问端口，应该修正相关配置。具体要求见下：

表 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.1.4.1 | apiserver | --profiling | 应设置为false | 是 |
| 5.1.4.2 | apiserver | --encryption-provider-config | 应设置 | 否 |
| 5.1.4.3 | apiserver | --secure-port  (Kubernetes 1.24以下版本) | 应正确配置，需为1和65535之间的整数值 | 是 |
| 5.1.4.4 | apiserver | --insecure-bind-address (Kubernetes 1.24以下版本) | 不应设置 | 是 |
| 5.1.4.5 | apiserver | --insecure-port (Kubernetes 1.24以下版本) | 应设置为0 | 是 |

### 5.1.5 日志安全

为保证对集群进行故障排查、安全监控、数据分析以及安全审计，需要合理的记录日志，保障记录的日志能够覆盖安全事件的问题，日志记录的周期足够长且能满足日志间关联分析的场景。应确保创建了审计策略，且审计策略涵盖关键的安全问题。具体要求见下：

表 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.1.5.1 | apiserver | --audit-log-path 或—audit-webhook-config-file | 应设置 | 是 |
| 5.1.5.2 | apiserver | --audit-log-maxage | 如使用log-path模式则此参数应设置为30或合适的值 | 是 |
| 5.1.5.3 | apiserver | --audit-log-maxbackup | 如使用log-path模式则此参数应设置为10或合适的值 | 是 |
| 5.1.5.4 | apiserver | --audit-log-maxsize | 如使用log-path模式则此参数应设置为100或其它合适的值 | 是 |
| 5.1.5.5 | apiserver | --audit-policy-file | 如使用log-path模式则此参数应确保创建最小化的审计策略，Kubernetes可以审计向API服务器发出的请求详情。要启用该日志记录，必须设置--audit-policy-file参数 | 否 |
| 5.1.5.6 | apiserver | --audit-webhook-initial-backoff | 如使用webhook模式则此参数应设置为5或合适的值 | 否 |

### 5.1.6 SSL/TLS配置

apiserver的 SSL/TLS 配置的主要目的是保证数据的安全性和完整性，以及验证通信双方的身份。具体要求见下：

表 6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.1.6.1 | apiserver | --tls-cipher-suites | 应只使用强加密套件 | 否 |
| 5.1.6.2 | apiserver | 无参数 | 人机交互场景下应禁止使用客户端证书用于身份验证 | 否 |

* 1. 控制管理器安全配置要求
     1. 文件目录安全

控制管理器（controller manager）的pod描述文件包含了组件的启动参数和证书文件，是直接配置控制管理器的文件，在创建集群的时候这些文件会保留在集群中用于修改相关的配置，因此需要保证其完整性、机密性且不被随意、恶意篡改，需严格限制该文件的属主和权限，确保只有管理员，或其他具备系统管理员权限的用户才能修改该文件。具体要求见下：

表 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象涉及文件** | **对文件的要求** | **自动化** |
| 5.2.1.1 | controller manager | pod描述文件 | 文件权限为600或更严格 | 是 |
| 5.2.1.2 | controller manager | pod描述文件 | 文件属主应为root:root | 是 |
| 5.2.1.3 | controller manager | controller-manager.conf | 文件权限为600或更严格 | 是 |
| 5.2.1.4 | controller manager | controller-manager.conf | 文件属主应为root:root | 是 |

* + 1. 身份认证和访问控制

控制管理器是集群内部的管理控制中心，它的职责是保证集群中各种资源的状态和用户定义的状态一致, 如果出现偏差, 则修正资源的状态。如replication controller 的核心作用是保障集群中某个 RC 关联的pod副本数与预设值一致。控制管理器内部包含了node controller,namespace controller,daemonset controller等多个控制器，每种控制器都负责一种具体的控制流程。需要确保每类控制器具备其所需的最小权限，且控制器能安全与apiserver建立通信。具体要求见下：

表 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.2.2.1 | controller manager | --use-service-account-credentials | 应设置为true | 是 |
| 5.2.2.2 | controller manager | --service-account-private-key-file | 应设置 | 是 |
| 5.2.2.3 | controller manager | --root-ca-file | 应设置 | 是 |
| 5.2.2.4 | controller manager | --RotateKubeletServerCertificate | 应设置为true | 是 |
| 注：5.2.2.4要求仅适用于让kubelet从apiserver获取证书的情况。如果kubelet证书来自外部权威机构/工具，则需要自己处理轮换问题。 | | | | |

* + 1. 防止拒绝服务

垃圾回收对于确保足够的资源可用性和避免降低性能和可用性很重要。在最坏的情况下，系统可能崩溃或长时间无法使用导致拒绝服务。当前的垃圾收集设置是12500个终止的 Pod，对于系统而言，可能太高了。根据系统资源和测试，选择适当的阈值以激活垃圾收集。与 Kubernetes API 服务器通信时突发峰值请求个数上限。默认情况下，设置为30，控制器管理器突发流量峰值，需根据当前集群规模配置合理的上限值，否则设置太大，将耗尽apiserver计算资源，使其易于被第三方利用攻击，导致整个集群无法使用。与 API 服务器通信时每秒请求数（QPS）限制，默认情况下，设置为20，控制器管理器突发流量峰值，需根据当前集群规模配置合理的上限值，否则设置太大，将耗尽apiserver计算资源，使其易于被第三方利用攻击，导致整个集群无法使用。

具体要求见下：

表 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.2.3.1 | controller manager | --terminated-pod-gc-threshold | 应设置 | 否 |
| 5.2.3.2 | controller manager | --kube-api-burst | 应设置 | 否 |
| 5.2.3.3 | controller manager | --kube-api-qps | 应设置 | 否 |
| 注：如果集群规模较小或资源有限，可以考虑适当降低--terminated-pod-gc-threshold阈值，例如设置为 5000 或更低。如果集群规模较大或资源充足，也可以稍微增加阈值，例如设置为 15000 或更高。需要注意的是，阈值的调整应该是一个渐进的过程，观察调整后的集群运行情况，并根据实际情况进行适当的调整。 | | | | |

* + 1. 防止信息泄露

控制管理器的不当配置会导致记录程序的详细信息（包含程序敏感数据）、使数据未加密或暴露不安全的访问端口，应该修正相关配置。具体要求见下：

表 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.2.4.1 | controller manager | --profiling | 应设置为false | 是 |
| 5.2.4.2 | controller manager | --bind-address | 应设置为127.0.0.1 | 是 |

* 1. 调度器安全配置要求
     1. 文件目录安全

调度器（scheduler）的pod描述文件和配置文件，包含了组件的启动参数和证书文件，这些都是直接配置调度器的关键文件。在创建集群的过程中，这些文件会被保留在集群中，用于后续的配置修改。鉴于这些文件的重要性，需严格限制该文件的属主和权限，确保只有管理员，或其他具备系统管理员权限的用户才能修改该文件。具体要求见下：

表 11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象涉及文件** | **对文件的要求** | **自动化** |
| 5.3.1.1 | scheduler | scheduler pod描述文件 | 文件的权限应设置为600或更严格 | 是 |
| 5.3.1.2 | scheduler | scheduler pod描述文件 | 文件属主应为root:root | 是 |
| 5.3.1.3 | scheduler | scheduler.conf文件 | 文件的权限应设置为600或更严格 | 是 |
| 5.3.1.4 | scheduler | scheduler.conf文件 | 文件属主应为root:root | 是 |

* + 1. 防止信息泄露

调度器的监听端口应限制在127.0.0.1上，且关闭调试API，避免信息泄露。具体要求见下：

表 12

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.3.2.1 | scheduler | --profiling | 应设置为false | 是 |
| 5.3.2.2 | scheduler | --bind-address | 应设置为127.0.0.1 | 是 |

* 1. etcd安全配置要求
     1. 文件目录安全

etcd的pod描述文件和配置文件，以及数据文件，需严格限制该文件的属主和权限，确保只有管理员，或其他具备系统管理员权限的用户才能修改该文件。具体要求见下：

表 13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象涉及文件** | **对文件的要求** | **自动化** |
| 5.4.1.1 | etcd | etcd pod描述文件 | 当etcd是由pod部署时，权限设置应为600或更严格 | 是 |
| 5.4.1.2 | etcd | etcd pod描述文件 | 当etcd是由pod部署时，文件属主应为root:root | 是 |
| 5.4.1.3 | etcd | etcd数据存放目录 | 权限设置应为700或更严格 | 是 |
| 5.4.1.4 | etcd | etcd数据存放目录 | 文件属主应为etcd:etcd | 是 |

* + 1. 身份认证和访问控制

etcd是Kubernetes部署使用的高可用性键值存储，用于持久存储其所有REST API对象。这些对象含有敏感信息，因此etcd在接受请求和发起请求时应正确配置身份认证和访问控制，具体要求见下：

表 14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.4.2.1 | etcd | --cert-file 和 --key-file | 应设置 | 是 |
| 5.4.2.2 | etcd | --client-cert-auth | 应设置为true | 是 |
| 5.4.2.3 | etcd | --auto-tls | 应禁止设置为true | 是 |
| 5.4.2.4 | etcd | --peer-cert-file 和 --peer-key-file | 应设置 | 是 |
| 5.4.2.5 | etcd | --peer-client-cert-auth | 应设置为true | 是 |
| 5.4.2.6 | etcd | --peer-auto-tls | 应禁止设置为true | 是 |
| 5.4.2.7 | etcd | 无参数 | 宜使用独立且唯一的CA证书 | 否 |
| 注：5.4.2.4仅适用于etcd集群。如果您在环境中仅使用一台etcd服务器，则此建议不适用。 | | | | |

* + 1. 防止拒绝服务

etcd应配置事物操作和请求大小参数，防止恶意或异常情况下的资源耗尽，确保 etcd 集群的稳定性和可靠性，具体要求见下：

表 15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.4.3.1 | etcd | --max-txn-ops | 应设置 | 否 |
| 5.4.3.2 | etcd | --max-request-bytes | 应设置 | 否 |
| 注：--max-txn-ops参数用来控制每个事务允许的最大操作数。默认值是 128，集群中存在较大的事务需求，可以适当增大该值，但一般不建议超过 512。  --max-request-bytes参数用来控制每个请求允许的最大字节数。默认值是 1.5 MB，某些特定应用需要更大的请求，可以适当增大该值，但一般不建议超过 10 MB | | | | |

* + 1. SSL/TLS配置

etcd发起TLS协商应启用强加密套件，保障通信安全，具体要求见下：

表 16

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.4.3.1 | etcd | --cipher-suites | 应只使用强加密套件 | 否 |

* 1. kube-proxy安全配置要求
     1. 文件目录安全

kube-proxy的pod描述文件和配置文件，需要严格限制该文件的属主和权限，确保只有管理员，或其他具备系统管理员权限的用户才能修改该文件。具体要求见下：

表 17

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象涉及文件** | **对文件的要求** | **自动化** |
| 5.5.1.1 | kube-proxy | kubeconfig文件 | 权限应设置为600或更严格 | 是 |
| 5.5.1.2 | kube-proxy | kubeconfig文件 | 文件所属应设置为root:root | 是 |
| 注：若kubeconfig文件存储位置为默认存储位置，则可以实现自动化扫描。 | | | | |

* 1. Kubelet安全配置要求
     1. 文件目录安全

kubelet服务文件设置工作节点中kubelet服务行为的各种参数，这些参数设置了工作节点中kubelet服务的行为。需严格限制该文件的属主和权限，确保只有管理员，或其他具备系统管理员权限的用户才能修改该文件。具体要求见下：

表 18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象涉及文件** | **对文件的要求** | **自动化** |
| 5.6.1.1 | kubelet | kubelet服务文件 | 权限设置应为600或更严格 | 是 |
| 5.6.1.2 | kubelet | kubelet服务文件 | 文件所属应设置为root:root | 是 |
| 5.6.1.3 | kubelet | kubeconfig kubelet.conf文件 | 权限设置应为600或更严格 | 是 |
| 5.6.1.4 | kubelet | kubeconfig kubelet.conf文件 | 文件所属应设置为root:root | 是 |
| 5.6.1.5 | kubelet | 客户端CA文件 | 权限设置应为600或更严格 | 是 |
| 5.6.1.6 | kubelet | 客户端CA文件 | 文件所属应设置为root:root | 是 |
| 5.6.1.7 | kubelet | kubelet的--config配置文件 | 权限设置应为600或更严格 | 是 |
| 5.6.1.8 | kubelet | kubelet的--config配置文件 | 文件所属应设置为root:root | 是 |

* + 1. 身份认证和访问控制

kubelet包含客户端证书与服务端证书，客户端证书为kubelet 用来向 apiserver 认证，表明自身身份的证书。kubelet 自身也向外提供一个 HTTPS 服务，包含若干功能特性，该场景则需要使用服务端证书。kubelet作为服务端接受请求和作为客户端发起请求时应进行身份认证，具体要求见下：

表 19

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.6.2.1 | kubelet | --read-only-port | 应设置为0 | 否 |
| 5.6.2.2 | kubelet | --rotate-certificates | 不应设置为false | 是 |
| 5.6.2.3 | kubelet | --tls-cert-file --tls-private-key-file | 应设置为合适值 | 否 |
| 5.6.2.4 | kubelet | --client-ca-file | 应设置为合适值 | 是 |
| 5.6.2.5 | kubelet | --anonymous-auth | 应设置为false或 在 config file文件中配置 | 是 |
| 5.6.2.6 | kubelet | --RotateKubeletServerCertificate | 应设置为true | 否 |
| 5.6.2.7 | kubelet | --authorization-mode | 应未设置为AlwaysAllow，参数处于deprecated状态，推荐使用config file文件来设置 | 是 |

* + 1. 防止拒绝服务

kubelet需能正确配置kubelet与其他服务通信的最大空闲时间以防止拒绝服务攻击，具体要求见下：

表 20

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.6.3.1 | kubelet | --stream-connection-idle-timeout | 不应设置为0或在config file文件中配置 | 否 |
| 5.6.3.2 | kubelet | --pod-max-pids | 宜设置容器内的进程数限制 | 是 |

* + 1. SSL/TLS配置

kubelet发起TLS协商应启用强加密套件，以保障通信加密，具体要求见下：

表 21

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.6.4.1 | kubelet | --tls-cipher-suites | 应使用强加密套件 | 否 |

* + 1. 系统安全

内核参数通常由系统管理员在将系统投入生产之前调整和加固，通过这些参数保护内核和系统，kubelet启动时应仅能修改安全的内核参数，具体要求见下：

表 22

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **对象** | **对象包含的参数** | **对参数的要求** | **自动化** |
| 5.6.5.1 | kubelet | --protect-kernel-defaults | 应设置为true，如启动参数未配置，请在--config所指的配置文件配置 | 是 |
| 5.6.5.2 | kubelet | --make-iptables-util-chains | 应设置为true，如启动参数未配置，请在--config所指的配置文件配置 | 是 |
| 5.6.5.3 | kubelet | --hostname-override | 宜未设置 | 否 |
| 5.6.5.4 | kubelet | --event-qps | 宜设置为0或是以保障捕获适当事件的级别，如启动参数未配置，请在--config所指的配置文件配置 | 否 |
| 5.6.5.5 | kubelet | --allowed-unsafe-sysctls | 应禁止设置，如启动参数未配置，请在--config所指的配置文件配置 | 否 |

* 1. CNI插件和网络策略安全配置要求
     1. 文件目录安全

Kubernetes网络策略一般由CNI插件来执行，因此应控制CNI插件和配置文件的属主和权限，确保只有管理员，或其他具备系统管理员权限的用户才能修改该文件。具体要求见下：

表 23

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 5.8.1.1 | CNI插件 | 应确保容器网络接口文件的权限设置为600或更严格； | 是 |
| 5.8.1.2 | CNI插件 | 应确保容器网络接口文件所属设置为root:root； | 是 |

* + 1. 网络策略安全配置

使用网络策略隔离集群网络中的流量，在同一Kubernete集群上运行不同的应用程序会导致一个受损应用程序攻击相邻应用程序的风险。具体要求见下：

表 24

|  |  |  |  |
| --- | --- | --- | --- |
| **要求序号** | **对象** | **要求** | **自动化** |
| 5.8.2.1 | NetworkPolicy | 宜确保使用的CNI插件支持网络策略(Network Policies) | 否 |
| 5.8.2.2 | NetworkPolicy | 宜设置合理的NetworkPolicy资源，以确保pod之间通信安全，还应确保所有 NetworkPolicy 至少针对一个 Pod | 否 |
| 5.8.2.3 | NetworkPolicy | 宜确保所有的命名空间都定义了网络策略 | 否 |

6 容器运行时基线要求

　　本章前3小节包含了Kubernetes集群中常见使用的运行时组件对应的安全基线要求，包括docker、containerd、cri-o，可根据集群中实际使用的运行时组件选择对应章节评估。

* 1. docker运行时安全要求

docker服务的核心配置目录和依赖文件的属主和权限配置是运行时组件默认安全的基础。同时需要规范docker进程的启动参数，以进一步加固运行时服务安全。具体要求见下:

6.1.1 文件目录安全

表 25

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.1.1.1 | docker配置文件 | 应确保docker.service文件所有权设置为root:root | 是 |
| 6.1.1.2 | docker配置文件 | 应确保将Docker套接字文件所有权设置为root:docker | 是 |
| 6.1.1.3 | docker配置文件 | 应确保将daemon.json文件所有权设置为root:root | 是 |
| 6.1.1.4 | docker配置文件 | 应确保将daemon.json文件权限设置为600或更严格 | 是 |
| 6.1.1.5 | docker配置文件 | 应确保正确设置了docker.service文件权限 | 是 |
| 6.1.1.6 | docker配置文件 | 应确保将/etc/sysconfig/docker文件所有权设置为root:root | 是 |
| 6.1.1.7 | docker配置文件 | 应确保将docker.socket文件所有权设置为root:root | 是 |
| 6.1.1.8 | docker配置文件 | 应确保将docker.socket文件权限设置为640或者更严格 | 是 |
| 6.1.1.9 | docker配置文件 | 应确保将/etc/docker目录所有权设置为root:root | 是 |
| 6.1.1.10 | docker配置文件 | 应确保将/etc/docker目录权限设置为750或更严格 | 是 |

6.1.2 守护进程安全

表 26

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.1.2.1 | docker守护进程 | 应禁止修改dm.basesize的大小 | 是 |
| 6.1.2.2 | docker守护进程 | 应确保容器引擎的live-restore功能已启用 | 是 |
| 6.1.2.3 | docker守护进程 | 应确保日志记录级别设置为"info" | 是 |
| 6.1.2.4 | docker守护进程 | 应确保允许Docker对iptables进行更改 | 是 |
| 6.1.2.5 | docker守护进程 | 应确保未使用aufs存储驱动程序 | 是 |
| 6.1.2.6 | docker守护进程 | 应确保生产环境中未开启实验功能 | 是 |
| 6.1.2.7 | docker守护进程 | 应确保将 --**no-new-privileges参数设置为true** | 是 |

6.1.3 宿主机安全

表 27

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.1.3.1 | 宿主机配置 | 应确保及时更新docker daemon版本以修复安全漏洞 | 是 |
| 6.1.3.2 | 宿主机配置 | 应确保为容器根目录创建独立分区 | 是 |

* 1. containerd运行时安全要求

containerd服务的核心配置目录和依赖文件的属主和权限配置是运行时组件默认安全的基础。同时需要规范containerd进程的启动参数，以进一步加固运行时服务安全。具体要求见下:

6.2.1 文件目录安全

表 28

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.2.1.1 | containerd配置文件 | 应确保及时更新containerd版本以修复安全漏洞 | 否 |
| 6.2.1.2 | containerd配置文件 | 应将containerd daemon 配置的log info 设为'info' | 是 |
| 6.2.1.3 | 宿主机配置 | 应确保为容器根目录创建独立分区 | 是 |
| 6.2.1.4 | containerd配置文件 | 应确保contrainerd配置文件的所属是root:root | 是 |
| 6.2.1.5 | containerd配置文件 | 应确保contrainerd配置文件的权限为640或更严格 | 是 |
| 6.2.1.6 | containerd配置文件 | 应确保contrainerd bin的权限为750或更严格 | 是 |
| 6.2.1.7 | containerd配置文件 | 应确保contrainerd所属组root:root | 是 |
| 6.2.1.8 | containerd配置文件 | 应确保/usr/lib/systemd/system/containerd.service的所属组是root:root | 是 |
| 6.2.1.9 | containerd配置文件 | 应确保/usr/lib/systemd/system/containerd.service的权限640 | 是 |
| 6.2.1.10 | containerd配置文件 | 应确保配置指定的containerd.sock的文件权限为640或更严格 | 是 |
| 6.2.1.11 | containerd配置文件 | 应确保配置指定的containerd.sock的用户组至少是root:root | 是 |
| 6.2.1.12 | containerd配置文件 | 应确保containerd元数据根目录的权限应为700 | 是 |
| 6.2.1.13 | containerd配置文件 | 应确保containerd元数据根目录的属主应为root:root | 是 |

6.2.2 守护进程安全

表 29

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.2.2.1 | containerd进程 | 应禁止使用不带证书的镜像仓库 | 是 |
| 6.2.2.2 | containerd进程 | 应当开启守护进程日志 | 是 |
| 6.2.2.3 | containerd进程 | 应禁止使用aufs作为存储驱动 | 是 |
| 6.2.2.4 | containerd进程 | 应允许containerd管理容器的cgroup | 是 |
| 6.2.2.5 | containerd进程 | 应确保containerd守护进程被终止不会影响容器运行 | 是 |
| 注：在常见的不带证书的镜像仓库下，可以参考使用6.2.2.1 | | | |

6.2.3 宿主机安全

表 30

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | 要求 | **自动化** |
| 6.2.3.1 | 宿主机配置 | 确保/etc/containerd目录属主为root:root | 是 |
| 6.2.3.2 | 宿主机配置 | 确保/etc/containerd目录权限为700 | 是 |
| 6.2.3.3 | 宿主机配置 | 确保Registry证书文件属主为root:root | 是 |

6.2.4 cri插件配置

containerd中的CRI插件，其主要用途是为了使containerd能够作为Kubernetes集群中的容器运行时。

表 31

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.2.4.1 | containerd CRI | 宜使用 systemd 作为 cgroup 管理的后端 | 是 |
| 6.2.4.2 | containerd CRI | 宜启用TLS流通信加密 | 是 |
| 6.2.4.3 | containerd CRI | 应确保disable\_cgroup未设置或设置为false | 是 |

* 1. cri-o运行时安全要求

cri-o服务的核心配置目录和依赖文件的属主和权限配置是运行时组件默认安全的基础。同时需要规范cri-o进程的启动参数，以进一步加固运行时服务安全。具体要求见下:

6.3.1 文件目录安全

表 32

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.3.1.1 | crio配置文件 | 应确保crio.service文件所有权设置为root:root | 是 |
| 6.3.1.2 | crio配置文件 | 应确保正确设置了crio.service文件权限 | 是 |
| 6.3.1.3 | crio配置文件 | 应确保将crio.sock文件属主为root:root | 是 |
| 6.3.1.4 | crio配置文件 | 应确保将crio.sock文件权限设置为640或更严格 | 是 |
| 6.3.1.5 | crio配置文件 | 应确保将/etc/sysconfig/crio文件属主为root:root | 是 |
| 6.3.1.6 | crio配置文件 | 应确保将/etc/sysconfig/crio文件权限设置为640或更严格 | 是 |
| 6.3.1.7 | crio配置文件 | 应确保将/etc/crio目录属主设置为root:root | 是 |
| 6.3.1.8 | crio配置文件 | 应确保将/etc/crio目录权限设置为750或更严格 | 是 |
| 6.2.1.9 | crio配置 | 应确保及时更新crio版本以修复安全漏洞 | 否 |

6.3.2 守护进程安全

表 33

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.3.2.1 | crio进程配置 | 应确保日志记录级别设置为"info" | 是 |
| 6.3.1.2 | crio进程配置 | 应确保未使用aufs存储驱动程序 | 是 |

6.3.3 宿主机安全

表 34

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 6.3.3.1 | 宿主机配置 | 应确保为容器根目录创建独立分区 | 是 |

7 工作负载安全配置

7.1 启动安全

Kubernetes的Yaml文件中通常存在对Pod所使用的网络、CPU、内存及宿主机资源的参数，若此类参数能够被正确配置，则可以减少Pod在运行时的风险，具体要求见下：

表 35

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 7.1.1 | Pod | 宜启用强制访问控制 | 否 |
| 7.1.2 | Pod | 应确保禁止root用户容器准入 | 是 |
| 7.1.3 | Pod | 宜确保最小化添加了capabilities的容器的准入，若必须使用，宜避免使用CAP\_SYS\_ADMIN、CAP\_NET\_ADMIN、CAP\_SYS\_PTRACE权限 | 否 |
| 7.1.4 | Pod | 应禁止使用特权容器，即禁止设置privileged=true | 是 |
| 7.1.5 | Pod | 应禁止容器挂载主机的敏感目录到容器中，特别是在读写模式下 | 是 |
| 7.1.6 | Pod | 应确保容器禁止主机上1024以下的特权端口映射到容器内 | 否 |
| 7.1.7 | Pod | 应确保容器仅开启必要的业务端口 | 否 |
| 7.1.8 | Pod | 应确保容器的根文件系统以只读方式挂载 | 是 |
| 7.1.9 | Pod | 应禁止使用主机网络命名空间 | 是 |
| 7.1.10 | Pod | 不宜共享使用主机IPC，即不宜设置hostIPC | 是 |
| 7.1.11 | Pod | 不宜共享主机进程PID，即不宜设置hostPID | 是 |
| 7.1.12 | Pod | 宜禁止容器共享主机设备 | 是 |
| 7.1.13 | Pod | 应最小化使用NET\_RAW功能的容器准入，若必须使用，宜使用NetworkPolicy限制容器访问网络访问端口或协议 | 是 |
| 7.1.14 | Pod | 应确保启动容器时不挂载运行时套接字文件 | 否 |
| 7.1.15 | Pod | 宜在Pod规格文件中应当将seccomp配置文件设置为docker/default | 是 |
| 7.1.16 | Pod | 应禁止允许特权提升的容器准入 | 是 |
| 7.1.17 | Pod | 应禁止Windows主机进程(HostProcess)容器准入 | 否 |
| 7.1.18 | Pod | 应确保最小化HostPath卷的准入，若必须使用，应限制HostPath挂载权限为只读权限 | 否 |
| 7.1.19 | Pod | 应确保不将容器监听端口映射到主机指定端口，例如使用hostPort | 是 |
| 注：最小化是指需要根据业务场景尽量少地使用规则，其中可使用场景已在附录D中说明。 | | | |

* 1. 身份认证和访问控制

Service Account是一种机密帐户，在Kubernete集群中为工作负载提供身份。为确保工作负载的安全，所有需要访问Kubernetes API的工作负载都需要创建显式服务帐户并基于RBAC最小化分配权限。

表 36

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 7.2.1 | Service Account | 应确保未主动使用默认服务帐户(service accounts) | 否 |
| 7.2.2 | Service Account | 应确保只在必要时挂载服务帐户令牌(token) | 否 |
| 7.2.3 | Service Account | 宜最小化Service Account的权限，例如特权命名空间下创建pod或创建、更新以及修改pod控制器（如DaemonSets, Deployments, Jobs）的资源权限、绑定clusterrolebindings或在特权命名空间下绑定rolebindings的权限 | 否 |
| 7.2.4 | Service Account | 应当最小化使用service account token创建权限，，若必须要使用，宜避免使用‘\*’、‘edit’、‘delete’权限，并且绑定到具体命名空间 | 是 |
| 7.2.5 | Service Account | 应当最小化使用pods/exec创建权限，若必须使用，宜避免使用‘\*’、edit’、‘delete’权限，并且绑定到具体命名空间 | 是 |
| 7.2.6 | Service Account | 应当最小化使用pods/ephemeralcontainers创建权限，若必须使用，宜避免使用‘\*’、‘edit’、‘delete’、‘patch’权限，并且绑定到具体命名空间 | 是 |
| 7.2.7 | Service Account | 应避免绑定有创建和批准certificatesigningrequests的资源权限 | 是 |
| 7.2.8 | ClusterRoles/ Roles | 应确保避免使用system:masters组 | 否 |
| 7.2.9 | ClusterRoles/ Roles | 应限制Kubernetes集群中Bind、Impersonate和Escalate权限的使用 | 是 |
| 7.2.10 | ClusterRoles/ Roles | 应确保仅在需要时使用cluster-admin角色 | 是 |
| 7.2.11 | ClusterRoles/ Roles | 应尽量减少Roles和ClusterRoles中通配符的使用 | 否 |

* 1. 防止信息泄露

Kubernetes提供了 Secret 资源来保存敏感信息，Secret中的data信息应加密存在，且最小化secrets的访问权限。若有更复杂的秘密管理需求，可以考虑使用外部secrets storage和管理系统，而不是直接使用Kubernetes Secrets。具体要求见下：

表 37

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 7.3.1 | secret | 应避免使用环境变量方式存储密文等敏感信息（secret） | 否 |
| 7.3.2 | secret | 宜使用外部机密管理系统保存secret资源 | 否 |
| 7.3.3 | secret | 应确保Kubernetes集群中证书私钥加密存储 | 否 |
| 7.3.4 | secret | 应最小化对secret的访问权限，宜使用 ClusterRoles限制secret只被特定命名空间下的资源访问 | 否 |

* 1. 资源配额

Kubernetes支持在工作负载和Namespace级别设置资源配额，合理设置资源配额有助于防止DDos攻击。具体要求见下：

表 38

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 7.4.1 | Pod | 应设置CPU资源请求(resources.requests.cpu)及资源限制(resources.limits.cpu)，并根据pod运行优先级为pod中每个容器正确设置资源请求和资源限制的匹配关系 | 是 |
| 7.4.2 | Pod | 应设置内存请求(resources.requests.memory)及内存限制(resources.limits.memory)，并根据pod运行优先级为pod中每个容器正确设置资源请求和资源限制的匹配关系 | 是 |
| 7.4.3 | Namespace | 宜限制每个命名空间能够分配的资源总量，控制的资源包括： CPU、内存、存储、Pods、Services、Deployments、Statefulsets、Daemonsets、PV、PVC、Volume、Storageclass等 | 否 |
| 7.4.4 | Namespace | 宜配置命名空间下容器的Limit ranges | 是 |
| 7.4.5 | Pod | 宜谨慎在Service中使用NodePort，避免绕过NetworkPolicy，若必须使用，宜定义单独的NetworkPolicy，并结合使用LoadBalancer或Ingress来提高可用性和扩展性 | 否 |

* 1. 其他

合理使用Namespace来隔离Kubernetes对象，限制用户权限的范围可以减少错误或恶意活动的影响。具体要求见下：

表 39

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 7.5.1 | Namespace | 应使用命名空间来创建资源之间的管理边界 | 否 |
| 7.5.2 | Namespace | 宜避免使用默认命名空间（针对业务场景） | 否 |
| 7.5.3 | Policy、 Namespace | 应确保集群至少有一个有效的策略控制机制 | 否 |
| 7.5.4 | PersistentVolume | 应当最小化使用创建持久卷权限，宜限制为admin角色创建PersistentVolume，并使用PersistentVolumeClaim创建存储卷，和POD进行绑定 | 否 |
| 7.5.5 | Node | 应当最小化使用节点代理资源权限，宜针对容器进行资源使用限制，权限根据业务逻辑进行合理评估 | 否 |
| 7.5.6 | WebHook | 应当谨慎使用webhook配置资源权限，若必须使用，宜定义webhook应用资源，并且使用RBAC进行权限控制 | 否 |

1. 镜像安全配置

8.1 镜像创建配置

Dockerfile是一个包含多个自定义命令的文本文件，用于构建自定义镜像。Dockerfile的正确配置则可以保证镜像创建的安全。具体要求见下：

表 40

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 8.1.1 | Dockerfile | 应确保部署镜像的版本（tag）不为空，并且值不为latest | 是 |
| 8.1.2 | Dockerfile | 在创建镜像时，应确保镜像只暴露必要的服务端口，减少暴露的服务和引入不必要的镜像层 | 是 |
| 8.1.3 | Dockerfile | 不应存储敏感信息，比如密钥等 | 否 |
| 8.1.4 | Dockerfile | 宜使用官方的、可信的镜像作为基础镜像 | 是 |
| 8.1.5 | Dockerfile | 宜在镜像中移除setuid和setgid权限 | 是 |
| 8.1.6 | Dockerfile | 应优先选择COPY而不是ADD | 是 |
| 8.1.7 | Dockerfile | 应确保镜像构建文件中不存在弃用命令，镜像构建文件中有些构建命令已弃用，如：MAINTAINER | 是 |
| 8.1.8 | Dockerfile | 避免单独或者在单行命令中使用update更新指令 | 是 |
| 8.1.9 | Dockerfile | 应避免使用 apt-get dist-upgrade | 是 |
| 8.1.10 | Dockerfile | 应尽量避免在容器中使用root用户，可以使用非特权用户来运行应用程序 | 是 |
| 8.1.11 | Dockerfile | 应避免文件夹不一致问题，即WORKDIR 路径参数应为绝对路径。 | 是 |
| 8.1.12 | Dockerfile | 应避免干扰dockerfile的可读性，即vim，ssh，kill等无意义的命令不应在RUN指令中被执行 | 是 |
| 8.1.13 | Dockerfile | 应在使用apt-get install 、 pip install 等安装应用时指定软件版本号 | 是 |
| 8.1.14 | Dockerfile | 应尽量通过清理本地缓存减小镜像大小，即检查是否包含 'apt-get clean' 本地缓存清理命令 | 是 |
| 8.1.15 | Dockerfile | 应避免定义多个HEALTHCHECK | 是 |
| 8.1.16 | Dockerfile | 应避免自身复制，即COPY '--from'不应引用当前stage别名。 | 是 |
| 8.1.17 | Dockerfile | 应避免存在多个CMD指令。一个 stage 中存在多个 CMD 指令时仅最后一个 CMD 指令生效 | 是 |
| 8.1.18 | Dockerfile | 应避免生成镜像层数过多，即若存在多个连续 RUN 指令，应该考虑合并。 | 是 |

* 1. 镜像仓库安全配置

镜像仓库作为存储和分发镜像的重要平台，其安全性直接影响到整个应用交付链的安全。为确保镜像仓库的安全，采取限制镜像仓库的访问控制和权限、定期扫描检查并修复存在的安全漏洞、加强传输存储的加密措施。具体要求见下：

表 41

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 8.2.1 | 镜像仓库 | 应确保镜像仓库配置自动化的安全扫描，定期扫描并修复存在的安全漏洞。 | 否 |
| 8.2.2 | 镜像仓库 | 镜像仓库应支持周期性扫描库内镜像，确保及时发现并修复存在的安全问题。 | 否 |
| 8.2.3 | 镜像仓库 | 应确保最小化的镜像仓库访问权限，根据人员，角色，项目空间访问来源等进行权限收敛，限制推送权限的授权范围，降低未授权访问的风险。 | 否 |
| 8.2.4 | 镜像仓库 | 宜通过策略治理限制在集群中可拉取的可信镜像仓库白名单，减少恶意镜像的使用 | 否 |
| 8.2.5 | 镜像仓库 | 应确保镜像仓库具备风险扫描的能力，发现镜像漏洞和弱口令等安全问题。 | 否 |
| 8.2.6 | 镜像仓库 | 镜像仓库应具备细粒度的访问控制，包括IP、角色、项目空间等，以保障镜像仓库的安全性。 | 否 |
| 8.2.7 | 镜像仓库 | 镜像仓库应具备镜像文件加密传输的能力，确保镜像文件在传输过程中的安全性。 | 是 |
| 8.2.8 | 镜像仓库 | 镜像仓库应具备镜像文件加密存储的能力，确保镜像文件在存储过程中的安全性 | 否 |

* 1. 镜像部署安全配置

容器镜像作为容器运行时的介质对象，在部署过程中的安全性影响到后续运行的容器安全性，在镜像部署时，具备一定的准入策略，可以保证镜像部署时的安全，具体要求见下：

表 42

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象** | **要求** | **自动化** |
| 8.3.1 | 容器镜像 | 宜具备合适的签名检验机制，确保镜像的真实性和完整性。 | 是 |
| 8.3.2 | 容器镜像 | 镜像发布到生产前应经过漏洞扫描和修复，保证镜像不含已知安全漏洞。 | 否 |
| 8.3.3 | 容器镜像 | 宜使用ImagePolicyWebhook准入控制插件配置镜像溯源，确保镜像来源可追溯。 | 否 |
| 8.3.4 | 容器镜像 | 宜使用经过安全加固的最小基础镜像，避免不必要的软件或命令，减少攻击面。 | 否 |



# 附 录 A

# （资料性附录）

**检查方法和修复方法**

附录A为第5章所述要求的检查方法和修复方法，其中表格序号列对应第5章所述要求，例如序号5.1.1.1对应第5章要求5.1.1.1。本文件所述检查方法和修复方法仅供参考。

## A.1 API Server安全配置

### A.1.1 文件目录安全

表A. 1

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 5.1.1.1 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600或者更严格  stat -c %a /etc/kubernetes/admin.conf | 参考如下命令，在master节点上执行命令将文件权限修改为600。  chmod 600 /etc/kubernetes/admin.conf |
| 5.1.1.2 | pod描述文件在k8s master节点运行如下命令进行检查，验证文件属主是否为root:root  1stat -c %U:%G /etc/kubernetes/manifests/kube-apiserver.yaml | 参考如下命令，在master节点上执行命令修改文件属主为root:root，默认情况下kube-apiserver.yaml文件属主为root:root。  chown root:root /etc/kubernetes/manifests/kube-apiserver.yaml |
| 5.1.1.3 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600或者更严格。  stat -c %a /etc/kubernetes/pki/\*.crt | 可参考如下命令，在master节点上执行命令将文件权限修改为600，默认情况下\*.crt文件的权限为644。  chmod 600 /etc/kubernetes/pki/\*.crt |
| 5.1.1.4 | 在k8s master节点运行如下命令进行检查，验证文件属主是否为root:root  ls -laR /etc/kubernetes/pki/ | 可参考如下命令，在master节点上执行命令修改文件属主为root:root，默认情况下/etc/kubernetes/pki/目录以及其中包含的所有文件属主为root:root。  chown -R root:root /etc/kubernetes/pki/ |
| 5.1.1.5 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600。  stat -c %a /etc/kubernetes/pki/\*.key | 可参考如下命令，在master节点上执行命令将文件权限修改为600，默认情况下\*.key文件的权限为600。（以实际环境目录为准）  chmod -R 600 /etc/kubernetes/pki/\*.key |

### A.1.2 身份认证和访问控制

表A. 2

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 5.1.2.1 | Master节点上执行如下命令，确认kube-apiserver的启动参数“--anonymous-auth”已经被配置为“false”  ps -ef |grep kube-apiserver | 修改Master节点上kube-apiserver的pod描述文件/etc/kubernetes/manifests/kube-apiserver.yaml，设置如下参数：  --anonymous-auth=false |
| 5.1.2.2 | Master节点上执行如下命令，确认不存在“--token-auth-file”参数  ps -ef |grep kube-apiserver | 参照官方文档使用其它可替代的认证方式，然后去除Master节点上apiserver的描述文件“/etc/kubernetes/manifests/kube-apiserver.yaml”中的--token-auth-file=<filename>参数。  默认未配置“--token-auth-file”参数 |
| 5.1.2.3 | 在Master节点上执行以下命令，验证--authorization-mode参数存在并且未设置为AlwaysAllow  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将“--authorization-mode”参数设置为AlwaysAllow以外的值，如，RBAC，Node，ABAC，Webhook，具体举例见下：  --authorization-mode = RBAC |
| 5.1.2.4 | 在Master节点上执行以下命令，检查“--authorization-mode”参数是否存在并且值中包括“Node”  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--authorization-mode参数设置为包含Node的值，如：  --authorization-mode=Node,RBAC |
| 5.1.2.5 | 在Master节点上执行以下命令，检查“--authorization-mode”参数是否存在并且值中包括“RBAC”  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--authorization-mode参数设置为包含RBAC的值，例如：  --authorization-mode=Node,RBAC |
| 5.1.2.6 | 在Master节点上执行以下命令，检查“--service-account-lookup”是否设置为“true”  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并设置以下参数。或者可以从此文件中删除--service-account-lookup参数，以使默认设置生效。  --service-account-lookup=true |
| 5.1.2.7 | 在Master节点上执行以下命令，检查“--service-account-key-file”是否指定文件  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--service-account-key-file参数设置为服务帐户的公钥文件--service-account-key-file=<key-file-filename> |
| 5.1.2.8 | 在Master节点上执行以下命令，确保已正确配置--kubelet-client-certificate和--kubelet-client-key参数  ps -ef |grep kube-apiserver | 在Master节点上的/etc/kubernetes/manifests/kube-apiserver.yaml并设置kubelet客户端证书和关键参数见下：  --kubelet-client-certificate=<path/to/client-certificate-file>  --kubelet-client-key=<path/to/client-key-file> |
| 5.1.2.9 | 在Master节点上执行以下命令，确保已正确配置“--kubelet-certificate-authority”  ps -ef |grep kube-apiserver | 编辑apiserver的pod规范文件，并将--kubelet-certificate-authority参数设置为ca文件路径。  --kubelet-certificate-authority=<ca-string> |
| 5.1.2.10 | 在Master节点上执行以下命令，检查“--tls-cert-file”和 “--tls-private-key-file”是否已正确配置证书和私钥  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并设置TLS证书和私钥文件参数。  --tls-cert-file=<path/to/tls-certificate-file>  --tls-private-key-file=<path/to/tls-key-file> |
| 5.1.2.11 | 在Master节点上执行以下命令，检查“client-ca-file”参数是否已正确配置。  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml并设置客户端ca文件。  --client-ca-file=<path/to/client-ca-file> |
| 5.1.2.12 | 在Master节点上执行以下命令，检查“--etcd-certfile”和“--etcd-keyfile”是否已正确配置证书和私钥  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml并设置etcd证书和密钥文件参数。  --etcd-certfile=<path/to/client-certificate-file>  --etcd-keyfile=<path/to/client-key-file> |
| 5.1.2.13 | 在Master节点上执行以下命令，检查“etcd-cafile”是否已正确配置  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml并设置etcd证书ca文件参数。  --etcd-cafile=<path/to/ca-file> |
| 5.1.2.14 | 在Master节点上执行以下命令，检查  “enable-bootstrap-token-auth”  是否配置该参数  ps -ef |grep kube-apiserver | 使用Kubernetes提供的替代机制，如使用OIDC（OpenID Connect 协议）取代Bootstrap token认证。 |
| 5.1.2.15 | 在Master节点上执行以下命令，检查“--enable-admission-plugins”参数是否存在并且值中包括“EventRateLimit”  ps -ef | grep “\-\-enable-admission-plugins” | 编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml并设置以下参数：  --enable-admission-plugins=...,EventRateLimit,...  --admission-control-config-file=<path/to/configuration/file> |
| 5.1.2.16 | 在Master节点上执行以下命令，查看是否设置了--enable-admission-plugins参数，若设置了，则其值不应包括AlwaysAdmit  ps -ef |grep “\-\-enable-admission-plugins” | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，然后删除--enable-admission-plugins参数，或者将其设置为不包含AlwaysAdmit的值 |
| 5.1.2.17 | 在Master节点上执行以下命令，检查“--enable-admission-plugins”参数被设置并且其值中包含了“AlwaysPullImages”  ps -ef |grep “\-\-enable-admission-plugins” | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--enable-admission-plugins参数设置为包括AlwaysPullImages |
| 5.1.2.18 | 在Master节点上执行以下命令，检查“--enable-admission-plugins”参数被设置并且其值中包含了“SecurityContextDeny”  ps -ef |grep “\-\-enable-admission-plugins” | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，若不使用PodSecurityPolicy，将--enable-admission-plugins参数需要开启SecurityContextDeny，如  --enable-admission-plugins=...,SecurityContextDeny,... |
| 5.1.2.19 | 在Master节点上执行以下命令，检查“--disable-admission-plugins”参数被设置并且其值中不包含“ServiceAccount”  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并确保--disable-admission-plugins参数设置为不包含ServiceAccount的值 |
| 5.1.2.20 | 在Master节点上执行如下命令，检查“--disable-admission-plugins”参数被设置并且其值中不包含NamespaceLifecycle  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并确保--disable-admission-plugins参数不包含NamespaceLifecycle |
| 5.1.2.21 | 在Master节点上执行以下命令，检查“--enable-admission-plugins”参数被设置并且其值中包含了“NodeRestriction”  ps -ef |grep “\-\-enable-admission-plugins” | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--enable-admission-plugins参数设置为包含NodeRestriction的值：  --enable-admission-plugins=...,NodeRestriction,... |
| 5.1.2.22 | 在Master节点上执行如下命令，检查“--DenyServiceExternalIPs”参数应未设置  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，删除--DenyServiceExternalIPs参数；或 在 kube-apiserver 的 disable-admission-plugins 参数中包含了 DenyServiceExternalIPs |

A.1.3 防止拒绝服务攻击

表A. 3

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.1.3.1 | 在Master节点上执行ps -ef |grep kube-apiserver命令，检查“request-timeout”是否配置以及是否是一个合适的值 | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kubeapiserver.yaml，并根据需要设置以下参数，推荐60s。 例如，  --request-timeout=60s |
| 5.1.3.2 | 在Master节点上执行“ps aux -w w|grep max-requests-inflight”是否配置以及是否是一个合适的值，如没有显性配置，说明配置的默认值。 | 编写master节点上apiserver静态pod的yaml文件（/etc/kubernetes/manifests/kube-apiserver.yaml），根据需要增加该参数 |
| 5.1.3.3 | 在Master节点上执行“ps aux -w w|grep ”max-mutating-requests-inflight“是否配置以及是否是一个合适的值，如没有显性配置，说明配置的默认值。 | 编写master节点上apiserver静态pod的yaml文件（/etc/kubernetes/manifests/kube-apiserver.yaml），根据需要增加该参数 |

A.1.4 防止信息泄露

表A. 4

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.1.4.1 | Master节点上执行以下命令，检查“profiling”是否配置为“false”  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kubeapiserver.yaml，并设置以下参数：  --profiling=false |
| 5.1.4.2 | 在Master节点上执行以下命令，检查“--encryption-provider-config”“--experimental-encryption-provider-config “是否已正确配置，确保已涵盖secret等期望被加密存储的资源  ps -ef |grep kube-apiserver | 遵循Kubernetes文档并配置EncryptionConfig文件。 然后，在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--encryption-provider-config参数设置为该文件的路径：  --encryption-provider-config=</path/to/EncryptionConfig/File> |
| 5.1.4.3 | 在Master节点上执行以下命令，验证--secure-port参数是否未设置或设置为介于1和65535之间的整数值  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，将--secure-port参数设置为其他所需的安全端口 |
| 5.1.4.4 | 在Master节点上执行以下命令，确认--insecure-bind-address参数不存在  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并删除--insecure-bind-address参数 |
| 5.1.4.5 | 在Master节点上执行以下命令，检查“--insecure-port”是否配置为“0”  ps -ef |grep kube-apiserver | 在在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml并设置以下参数，  --insecure-port = 0 |

A.1.5 日志安全

表A. 5

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.1.5.1 | 在Master节点上执行以下命令，检查“audit-log-path”是否配置了正确的路径  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--audit-log-path参数设置为要写入审核日志的适当路径和文件，例如：  --audit-log-path=/var/log/apiserver/audit.log |
| 5.1.5.2 | 在Master节点上执行以下命令，检查“audit-log-maxage”参数是否设置为30或适当的设置  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--audit-log-maxage参数设置为30或适当的天数：  --audit-log-maxage=30 |
| 5.1.5.3 | 在Master节点上执行以下命令，检查“audit-log-maxbackup”参数是否设置为10或适当的设置  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，并将--audit-log-maxbackup参数设置为10或适当的值。  --audit-log-maxbackup=10 |
| 5.1.5.4 | 在Master节点上执行以下命令，检查“audit-log-maxsize”参数是否设置为100或适当的设置  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml并将--audit-log-maxsize参数设置为适当的大小，以MB为单位。 例如，将其设置为100 MB：  --audit-log-maxsize=100 |
| 5.1.5.5 | 在其中一个集群主节点上执行以下命令：ps -ef | grep kube-apiserver，检查是否设置了--audit-policy-file。检查指定文件的内容，并确保其中包含有效的审核策略  ps -ef |grep kube-apiserver | 为集群创建审计策略文件。 默认集群中未创建审计策略文件，除非指定--audit-policy-file标志，否则不会执行审计。 |

A.1.6 SSL/TLS配置

表A. 6

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.1.6.1 | 在Master节点上执行以下命令，确保“--tls-cipher-suites”参数值是否为HIGH的加密套件。  ps -ef |grep kube-apiserver | 在Master节点上编辑apiserver的pod规范文件/etc/kubernetes/manifests/kube-apiserver.yaml，推荐使用TLS1.2或TLS1.3协议建立加密通道，TLS1.2加密套件应选择安全程度为HIGH的加密套件，TLS1.3的加密套件请参考RFC 8446。 |
| 5.1.6.2 | 检查用户对集群的访问权限，并确保用户没有使用Kubernetes客户端证书身份验证。默认情况下，客户端证书身份验证处于启用状态。 | 使用Kubernetes提供的替代机制，如使用OIDC（OpenID Connect 协议）取代客户端证书。 |

A.2 控制管理器安全配置

A.2.1文件目录安全

表A.7

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.2.1.1 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600或者更严格。  stat -c %a /etc/kubernetes/manifests/kube-controller-manager.yaml | 如果不符合要求则对文件权限进行修改，可参考如下命令，在master节点上执行命令将文件权限修改为600，默认情况下kube-controller-manager.yaml文件的权限为640。  chmod 600 /etc/kubernetes/manifests/kube-controller-manager.yaml |
| 5.2.1.2 | 在k8s master节点运行如下命令进行检查，验证文件属主是否为root:root。  stat -c %U:%G /etc/kubernetes/manifests/kube-controller-manager.yaml | 如果不符合规则要求则修改文件属主，可参考如下命令，在master节点上执行命令修改文件属主为root:root，默认情况下kube-controller-manager.yaml文件属主为root:root。  chown root:root /etc/kubernetes/manifests/kube-controller-manager.yaml |
| 5.2.1.3 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600或者更严格。  stat -c %a /etc/kubernetes/controller-manager.conf  执行命令返回的权限结果确保为600或更小权限。 | 如果不符合要求则对文件权限进行修改，可参考如下命令，在master节点上执行命令将文件权限修改为600，默认情况下controller-manager.conf文件的权限为640。  chmod 600 /etc/kubernetes/controller-manager.conf |
| 5.2.1.4 | 在k8s master节点运行如下命令进行检查，验证文件属主是否为root:root。  stat -c %U:%G /etc/kubernetes/controller-manager.conf  执行命令返回的属主结果确保root:root。 | 如果不符合规则要求则修改文件属主，可参考如下命令，在master节点上执行命令修改文件属主为root:root，默认情况下controller-manager.conf文件属主为root:root。  chown root:root /etc/kubernetes/controller-manager.conf |

A.2.2 身份认证和访问控制

表A.8

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.2.2.1 | 在Master节点上执行如下命令，确认“--use-service-account-credentials”是否为“true”。  .ps -ef |grep kube-controller-manager | 在master节点上编辑控制管理器的pod规格文件/etc/kubernetes/manifests/kube-controller-manager.yaml，配置如下参数：  --use-service-account-credentials=true |
| 5.2.2.2 | 在Master节点上执行如下命令，确认“--service-account-private-key-file”是否正确配置。  ps -ef |grep kube-controller-manager | 在master节点上编辑控制管理器的pod规格文件/etc/kubernetes/manifests/kube-controller-manager.yaml，将参数--service-account-privatekey-file设置为业务帐号的私钥文件：  --service-account-private-key-file=<filename>  “--service-account-private-key-file”默认值未设置 |
| 5.2.2.3 | 在Master节点上执行如下命令，检查“--root-ca-file”是否存在，并且该参数配置为包含apiserver服务证书根证书的证书捆绑文件。  ps -ef |grep kube-controller-manager | 在master节点上编辑控制管理器的pod规格文件/etc/kubernetes/manifests/kube-controller-manager.yaml，将参数--root-ca-file设置为根ca证书文件：  --root-ca-file=<filename>  “--root-ca-file”默认值未设置 |
| 5.2.2.4 | 在Master节点上执行如下命令，检查RotateKubeletServerCertificate参数是否存在，如果存在，需要确保未被设置为false。  ps -ef |grep kube-controller-manager | 在master节点上编辑控制管理器的pod规格文件/etc/kubernetes/manifests/kube-controller-manager.yaml，将参数--feature-gates的值修改为包含RotateKubeletServerCertificate=true，如下所示：  --feature-gates=RotateKubeletServerCertificate=true  RotateKubeletServerCertificate默认值为true |

A.2.3 防止拒绝服务

表A. 9

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.2.3.1 | 在Master节点上执行如下命令，确认--terminated-pod-gc-threshold是否正确配置。  ps -ef |grep kube-controller-manager | 在master节点上编辑控制管理器的pod规格文件/etc/kubernetes/manifests/kube-controller-manager.yaml，将--terminated-pod-gc-threshold设置为合适的阈值，例如：  --terminated-pod-gc-threshold=10  “terminated-pod-gc-threshold”默认值为“12500” |
| 5.2.3.2 | 在Master节点上执行“ps aux -w w|grep kube-api-burst”是否配置以及是否是一个合适的值，如没有显性配置，说明配置的 | 编写master节点上controller-manager静态pod的yaml文件（/etc/kubernetes/manifests/kube-controoler-manager.yaml），根据需要增加该参数 |
| 5.2.3.3 | 在Master节点上执行“ps aux -w w|grep kube-api-qps”是否配置以及是否是一个合适的值，如没有显性配置，说明配置的默认值。 | 编写master节点上controller-manager静态pod的yaml文件（/etc/kubernetes/manifests/kube-controoler-manager.yaml），根据需要增加该参数 |

A.2.4 防止信息泄露

表A.10

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.2.4.1 | 在Master节点上执行如下命令，检查--profiling参数是否设置为false。  ps -ef |grep kube-controller-manager | 在master节点上编辑控制管理器的pod规格文件/etc/kubernetes/manifests/kube-controller-manager.yaml，配置如下参数：  --profiling=false  “profiling”默认值为“true”。 |
| 5.2.4.2 | 在Master节点上执行如下命令，确认“--bind-address”参数为“127.0.0.1”。  1.ps -ef |grep kube-controller-manager | 编辑控制管理器的Pod规格文件，即/etc/kubernetes/manifests/kube-controller-manager.yaml文件，确保参数--bind-address填写正确，为127.0.0.1。  “--bind-address”默认值为“0.0.0.0”。 |

A.3 调度器安全配置

A.3.1 文件目录安全

表A.11

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.3.1.1 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600或者更严格。  stat -c %a /etc/kubernetes/manifests/kube-scheduler.yaml | 如果不符合要求则对文件权限进行修改，可参考如下命令，在master节点上执行命令将文件权限修改为600，默认情况下ube-scheduler.yaml文件的权限为640。  chmod 600 /etc/kubernetes/manifests/kube-scheduler.yaml |
| 5.3.1.2 | 在k8s master节点运行如下命令进行检查，验证文件属主是否为root:root  stat -c %U:%G /etc/kubernetes/manifests/kube-scheduler.yaml | 如果不符合规则要求则修改文件属主，可参考如下命令，在master节点上执行命令修改文件属主为root:root，默认情况下kube-scheduler.yaml文件属主为root:root。  chown root:root /etc/kubernetes/manifests/kube-scheduler.yaml |
| 5.3.1.3 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600或者更严格。  stat -c %a /etc/kubernetes/scheduler.conf | 如果不符合要求则对文件权限进行修改，可参考如下命令，在master节点上执行命令将文件权限修改为600，默认情况下scheduler.conf文件的权限为640。  stat -c %a /etc/kubernetes/scheduler.conf |
| 5.3.1.4 | 在k8s master节点运行如下命令进行检查，验证文件属主是否为root:root。  stat -c %U:%G /etc/kubernetes/scheduler.conf | 如果不符合规则要求则修改文件属主，可参考如下命令，在master节点上执行命令修改文件属主为root:root，默认情况下scheduler.conf文件属主为root:root。  chown scheduler:scheduler/etc/kubernetes/scheduler.conf |

A.3.2 防止信息泄露

表A. 12

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.3.2.1 | 在Master节点上执行如下命令，检查--profiling参数是否设置为false。  ps -ef |grep kube-scheduler | 在master节点上编辑Scheduler pod规格文件/etc/kubernetes/manifests/kube-scheduler.yaml，配置如下参数。  --profiling=false  “profiling”默认值为“true”。 |
| 5.3.2.2 | 在Master节点上执行如下命令，确认“--bind-address”参数为“127.0.0.1”。  ps -ef |grep kube-scheduler | 编辑Scheduler的Pod规格文件，即/etc/kubernetes/manifests/kube-scheduler.yaml文件，确保参数--bind-address填写正确，为“127.0.0.1”。  “--bind-address”默认值为“0.0.0.0”。 |

A.4 etcd安全配置

A.4.1文件目录安全

表A. 13

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.4.1.1 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600或者更严格  stat -c %a /etc/kubernetes/manifests/etcd.yaml | 如果不符合要求则对文件权限进行修改，可参考如下命令，在master节点上执行命令将文件权限修改为600，默认情况下etcd.yaml文件的权限为640。chmod 600 /etc/kubernetes/manifests/etcd.yaml |
| 5.4.1.2 | 在k8s master节点运行如下命令进行检查，验证文件属主是否为root:root  stat -c %U:%G /etc/kubernetes/manifests/etcd.yaml | 如果不符合规则要求则修改文件属主，可参考如下命令，在master节点上执行命令修改文件属主为root:root，默认情况下etcd.yaml文件属主为root:root。  chown root:root /etc/kubernetes/manifests/etcd.yaml |
| 5.4.1.3 | etcd的数据目录路径视具体情况而定，默认路径在/var/lib/etcd，也可以在etcd服务器节点上通过命令查看--data-dir配置的路径。  ps -ef | grep etcd  stat -c %a <path/to/etcd> | 在etcd服务器节点上执行命令将文件权限修改为700，默认情况下etcd的数据目录权限为755。  chmod 700 <path/to/etcd> |
| 5.4.1.4 | 文件路径视具体情况而定，默认文件路径在/var/lib/etcd，也可以在etcd服务器节点上通过命令查看--data-dir配置的路径。  ps -ef | grep etcd  stat -c %U:%G <path/to/etcd> | 在etcd服务器节点上执行命令修改文件属主为etcd:etcd，默认情况下etcd的数据目录属主为etcd:etcd。  chown etcd:etcd <path/to/etcd> |

A.4.2 身份认证和访问控制

表A. 14

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.4.2.1 | 在etcd服务器节点上执行ps -ef | grep etcd  命令,验证--cert-file和--key-file参数是否设置得适当 | 按照etcd服务文档，配置TLS加密。然后，在主节点上编辑etcd pod规格文件/etc/kubernetes/manifests/etcd.yaml，并设置以下参数。  --cert-file=</path/to/ca-file>  --key-file=</path/to/key-file>  默认情况下，未设置TLS加密。 |
| 5.4.2.2 | 在etcd服务器节点上执行ps -ef | grep etcd  命令, 验证--client-cert-auth参数是否设置为true | 在主节点上编辑etcd pod规格文件/etc/kubernetes/manifests/etcd.yaml，并设置以下参数。  --client-cert-auth="true"  默认情况下，etcd服务可以被未经认证的客户端查询 |
| 5.4.2.3 | 在etcd服务器节点上执行ps -ef | grep etcd  命令, 验证如果--auto-tls参数存在，则不应该设置为true | 在主节点上编辑etcd pod规格文件/etc/kubernetes/manifests/etcd.yaml，删除--auto-tls参数或将其设置为false。  --auto-tls=false  默认情况下，--auto-tls设置为false |
| 5.4.2.4 | 在etcd服务器节点上执行ps -ef | grep etcd命令, 验证--peer-cert-file和--peer-key-file参数是否设置得适当 | 按照etcd服务文档，并根据需要为您的etcd集群配置对等TLS加密。然后，在主节点上编辑etcd pod规格文件/etc/kubernetes/manifests/etcd.yaml，并设置以下参数。  --peer-client-file=</path/to/peer-cert-file>  --peer-key-file=</path/to/peer-key-file> |
| 5.4.2.5 | 在etcd服务器节点上执行ps -ef | grep etcd命令, 验证--peer-client-cert-auth参数是否设置为true | 在主节点上编辑etcd pod规格文件/etc/kubernetes/manifests/etcd.yaml，并设置以下参数。  --peer-client-cert-auth=true |
| 5.4.2.6 | 在etcd服务器节点上执行ps -ef | grep etcd命令, 验证如果--peer-auto-tls参数存在，若存在设置为true | 编辑master节点上的etcd pod规范文件/etc/kubernetes/manifests/etcd.yaml，删除--peer-auto-tls参数或将其设置为false。  --peer-auto-tls=false |
| 5.4.2.7 | 1.在主节点上执行ps -ef | grep etcd命令 2.在主节点上执行ps -ef | grep apiserver命令，检查apiserver的--client-ca-file引用的文件是否与etcd使用的--trusted-ca-file不同 | 遵循etcd文档，并为etcd服务创建专用ca。然后，在主节点上编辑etcd pod规范文件/etc/kubernetes/manifests/etcd.yaml，并设置以下参数。  --trusted-ca-file=</path/to/ca-file> |

A.4.3防止拒绝服务

表A. 15

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.4.3.1 | 执行ps -ef | grep etcd命令，如果存在--max-txn-ops参数，应设置合适的值 | 参数用来控制每个事务允许的最大操作数。默认值是 128，集群中存在较大的事务需求，可以适当增大该值，但一般不建议超过 512。 |
| 5.4.3.2 | 执行ps -ef | grep etcd命令，如果存在--max-request-bytes参数，应设置合适的值 | 参数用来控制每个请求允许的最大字节数。默认值是 1.5 MB，某些特定应用需要更大的请求，可以适当增大该值，但一般不建议超过 10 MB |

A.4.4 SSL/TLS配置

表A. 16

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.4.3.1 | 执行ps -ef | grep etcd命令，如果存在--cipher-suites参数，设置为仅包含安全套件中的值 | 推荐使用TLS1.2或TLS1.3协议建立加密通道，TLS1.2加密套件应选安全程度为HIGH的加密套件，TLS1.3的加密套件请参考RFC 8446。以TLS1.2举例，  按照etcd服务官方文档，配置TLS加密。然后，在主节点上编辑etcd pod规格文件/etc/kubernetes/manifests/etcd.yaml，并将--cipher-suites参数设置为安全加密套件中的值 |

A.5 kube-proxy安全配置

A.5.1 文件目录安全

表A. 17

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.5.1.1 | 在节点上执行如下命令ps -ef |grep kube-proxy  ，找到“--kubeconfig”指定的文件路径  使用如下命令，确认文件权限是否为600或者更严格。默认情况下，文件权限为640。  stat -c %a <proxy kubeconfig file> | 在每个工作节点上运行下面的命令（根据系统上的文件位置）。例如：  chmod 600 <proxy kubeconfig file> |
| 5.5.1.2 | 在节点上执行如下命令ps -ef |grep kube-proxy  ，找到“--kubeconfig”指定的文件路径  使用如下命令确认文件属主是否为root:root，默认情况下，该文件属主为root:root.  stat -c %a <proxy kubeconfig file> | 在每个工作节点上运行下面的命令（根据系统上的文件位置）。例如：  chown root:root <proxy kubeconfig file> |

A. 6 kubelet安全配置

A.6.1 文件目录安全

表A. 18

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.6.1.1 | 在节点上执行如下命令，执行命令返回的权限结果确保为600或更小权限。默认情况下，文件权限为640。  stat -c %a /etc/systemd/system/kubelet.service.d/10-kubeadm.conf | 在每个工作节点上运行下面的命令（基于系统中的文件位置），例如：  chmod 600 /etc/systemd/system/kubelet.service.d/kubeadm.conf |
| 5.6.1.2 | 在节点上执行如下命令，执行命令返回文件属主为root:root。  stat -c %a /etc/systemd/system/kubelet.service.d/10-kubeadm.conf | 在每个工作节点上运行下面的命令（基于系统中的文件位置），例如：  chown root:root /etc/systemd/system/kubelet.service.d/kubeadm.conf |
| 5.6.1.3 | 请执行以下命令查询--kubeconfig对应的kubelet.conf文件的路径，默认情况下，文件位于“/etc/kubernetes/”路径下。  ps -ef | grep kubelet  使用如下命令查看文件权限为600或者更严格。默认情况下，文件权限为640。  stat -c %a /etc/kubernetes/kubelet.conf | 在每个工作节点上运行下面的命令（基于系统中的文件位置），例如：  chmod 600 /etc/kubernetes/kubelet.conf |
| 5.6.1.4 | 执行5.6.1.3命令查询文件路径后，使用如下命令查看文件属主是否为root:root  stat -c %a /etc/kubernetes/kubelet.conf | 在每个工作节点上运行下面的命令（基于系统中的文件位置），例如：  chown root:root /etc/kubernetes/kubelet.conf |
| 5.6.1.5 | 执行如下命令查到“--client-ca-file”参数指定的文件路 | 默认情况下，--client-ca-file未设置。 |
| 5.6.1.5 | 径。  ps -ef |grep kubelet  执行如下命令，确认返回的权限不大于600。  stat -c %a <filename> | 执行以下命令，修改--client-ca-file文件权限:  chmod 600 <filename> |
| 5.6.1.6 | 执行如下命令查到“--client-ca-file”参数指定的文件路径。  ps -ef |grep kubelet  执行如下命令，确认返回的属主为root:root。  stat -c %a <filename> | 默认情况下，--client-ca-file未设置。 执行以下命令，修改--client-ca-file文件属主:  chown root:root <filename> |
| 5.6.1.7 | 在业务节点上执行如下命令查看“--config”指定的文件路径  ps -ef | grep kubelet | grep config  使用命令查看文件权限是否为600或者更严格。  stat -c %a <kubelet config> | 默认情况下，/var/lib/kubelet/config.yaml由kubeadm创建，权限为644。  执行如下命令（使用检查方法中识别的config文件位置）：  chmod 600 <kubelet config>(eg. /var/lib/kubelet/config.yaml) |
| 5.6.1.8 | 在业务节点上执行如下命令查看“--config”指定的文件路径  ps -ef | grep kubelet | grep config  使用命令查看文件属主是否为root:root。  stat -c %a <kubelet config> | 默认情况下属主为root:root。  执行如下命令（使用检查方法中识别的config文件位置）：  chown root:root <kubelet config>(eg. /var/lib/kubelet/config.yaml) |

A. 6.2 身份认证和访问控制

表A. 19

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.6.2.1 | 在每个节点上执行ps -ef | grep kubelet  命令，验证--read-only-port参数是否存在，如果存在，是否正确设置为0。 如果不存在--read-only-port参数，请检查是否存在由--config指定的Kubelet配置文件。检查如果文件中存在readOnlyPort条目，则将其设置为0。 | 默认情况下，--read-only-port设置为10255  1. 如果使用kubelet config文件，编辑该文件，将“--read-only-port”设置为0。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_SYSTEM\_PODS\_ARGS变量中设置如下参数：--read-only-port=0。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.2.2 | 在每个节点上执行ps -ef | grep kubelet命令，验证--rotate-certificates参数是否不存在，如果存在，是否正确设置为true。 如果--rotate-certificates参数不存在，请验证如果存在由--config指定的Kubelet配置文件，则该文件不包含 rotateCertificates: false | 1. 如果使用kubelet config文件，编辑该文件，添加“rotateCertificates:true”行，或者完全移除，使用默认值。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_CERTIFICATE\_ARGS变量中变量中设置如下参数：--rotate-certificates=true  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.2.3 | 集群节点上执行命令ps -ef | grep kubelet  查看参数“--tls-cert-file”和“--tls-private-key-file”的配置。  如果这些参数不存在，请检查是否存在由--config指定的Kubelet配置，并且它包含tlsCertFile和tlsPrivateKey的适当设置。 | 1. 如果使用kubelet config文件，编辑该文件，将tlsCertFile设置为要用于标识该kubelet的证书文件位置，tlsPrivateKeyFile设置为对应的私钥文件位置。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf ，在KUBELET\_CERTIFICATE\_ARGS变量中设置如下参数：--tls-cert-file=<path/to/tls-certificate-file> 、--tls-private-keyfile=<path/to/tls-key-file>。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.2.4 | 如果不存在--client-ca-file参数，请检查是否存在由--config指定的Kubelet配置文件，并且该文件是否将身份验证：x509: clientCAFile设置为ca文件的位置。  ps -ef | grep kubelet | 1. 如果使用kubelet config文件，编辑该文件，将“authentication: x509: clientCAFile”设置为ca文件的位置。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_AUTHZ\_ARGS变量中设置如下参数：--client-ca-file=<path/to/client-ca-file>。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.2.5 | 在每个工作节点上执行以下命令，验证--anonymous-auth参数是否设置为false。 如果不存在--anonymous-auth参数，请检查是否存在由--config指定的Kubelet配置文件，并查看匿名参数authentication:anonymous: enabled值是否设置为false。  ps -ef | grep kubelet | 默认情况下，kubelet服务器允许匿名访问。  1. 如果使用kubelet config文件，编辑该文件，将“authentication:anonymous: enabled”设置为“false”。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf。 在KUBELET\_SYSTEM\_PODS\_ARGS变量中设置如下参数：--anonymous-auth=false，注意"anonymous-auth"默认值为“true”。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.2.6 | 如果 kubelet 配置文件中的 serverTLSBootstrap 为 true，或者 kubelet config文件上设置了 --rotate-server-certificates 参数，则忽略此检查。在每个节点上执行ps -ef | grep kubelet命令：验证是否存在RotateKubeletServerCertificate参数并将其设置为true | 1.在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，设置：--rotate-certificates=true，在KUBELET\_CERTIFICATE\_ARGS变量中配置如下参数：--feature-gates=RotateKubeletServerCertificate=true  2. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.2.7 | 在每个节点上执行ps -ef | grep kubelet命令，验证--authorization-mode参数，禁止设置AlwaysAllow。如果不存在--authorization-mode参数，请检查是否存在由--config指定的Kubelet配置文件。查看authorization: mode参数值是否设置为Webhook。 | 1. 如果使用kubelet config文件，编辑该文件，将authorization: mode设置为Webhook。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_AUTHZ\_ARGS变量中设置如下参数：--authorization-mode=Webhook。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |

A.6.3 防止拒绝服务

表A. 20

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.6.3.1 | 在每个工作节点上执行ps -ef | grep kubelet命令，验证--streaming-connection-idie-timeout参数是否未设置为0 如果该参数不存在，并且存在由--config指定的Kubelet配置文件，请检查该文件里没有将StreamConnectionldleTimeout设置为0 | “streaming-connection-idle-timeout”默认值为4小时。  1. 如果使用kubelet config文件，编辑该文件，将“--streaming-connection-idle-timeout”设置为满足实际业务场景的最小值。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_SYSTEM\_PODS\_ARGS变量中设置如下参数：--streaming-connection-idle-timeout=5m。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.3.2 | 集群节点上执行命令ps -ef | grep kubelet  查看参数“--pod-max-pids”，如果参数不存在，请检查是否存在由--config指定的Kubelet配置，该配置文件是否包含PodPidsLimit 的适当设置。 | 1. 如果使用kubelet config文件，编辑该文件，添加“PodPidsLimit:1000”行，其中1000可根据业务场景调整，或者使用默认值。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在启动参数中添加 --pod-max-pids配置。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |

A. 6.4 SSL/TLS配置

表A. 21

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.6.4.1 | 执行ps -ef | grep kubelet命令，如果存在--tls-cipher-Suite参数，设置为仅包含安全套件中的值。如果不存在--tls-cipher-Suite参数，请检查是否存在由--config指定的Kubelet配置文件配置是否包含TLScipherSuite，设置为仅包括安全套件中的值 | 推荐使用TLS1.2或TLS1.3协议建立加密通道，TLS1.2加密套件应选择安全程度为HIGH的加密套件，TLS1.3的加密套件请参考RFC 8446 |

A. 6.5 系统安全

表A. 22

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.6.5.1 | 在每个节点上执行ps -ef | grep kubelet命令，检查--protect-kernel-defaults参数是否设置为true。 如果不存在--protect-kernel-defaults参数，请检查是否存在由--config指定的Kubelet配置文件，并且该文件是否将保protectKernelDefaults设置为true | "protect-kernel-defaults"默认值为“false”。  1. 如果使用kubelet config文件，编辑该文件，设置protectKernelDefaults: true。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_SYSTEM\_PODS\_ARGS变量中设置如下参数：--protect-kernel-defaults=true。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.5.2 | 在每个工作节点上执行ps -ef | grep kubelet命令，验证如果--make-iptables-util-chains参数存在，则它是否设置为true。 如果--make-iptables-util-chains参数不存在，并且存在由--config指定的Kubelet配置文件，请验证该文件没有将makelPTablesUtilChains参数设置为false | “make-iptables-util-chains”默认值为“true”。  1． 如果使用kubelet config文件，编辑该文件，设置makeIPTablesUtilChains: true。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_SYSTEM\_PODS\_ARGS变量中去掉--make-iptables-util-chains参数。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.5.3 | 集群节点上执行ps -ef | grep kubelet命令查看参数“--hostname-override”的配置 | 默认情况下，系统未设置，hostname-override参数。  1. 编辑每个worker节点上的kubelet服务文件/etc/kubernetes/kubelet.conf，去掉KUBELET\_SYSTEM\_PODS\_ARGS变量中的--hostname-override参数。  2. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.5.4 | 执行ps -ef | grep kubelet命令命令，查看--event-qps参数的值，并确定是否已将其设置为适当值。当参数为0时，表示捕获所有事件。 如果--event-qps参数不存在，请检查是否存在由--config指定的Kubelet配置文件，并查看此位置中的值 | “event-qps”默认值为“5”。  1. 如果使用kubelet config文件，编辑该文件，将eventRecordQPS：设置为合适的级别。  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_SYSTEM\_PODS\_ARGS变量中设置如下参数：--event-qps=0。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |
| 5.6.5.5 | 在每个节点上执行以下命令，验证--allowed-unsafe-sysctls参数是否存在，禁止配置-allowed-unsafe-sysctls参数。  ps -ef | grep kubelet | 默认情况下， kubelet未开启allowedUnsafeSysctls。  1. 如果使用kubelet config文件，编辑该文件，将allowedUnsafeSysctls参数值置空，或移除，使用默认值；  2. 如果使用可执行参数，则在每个worker节点上编辑kubelet服务文件/etc/kubernetes/kubelet.conf，在KUBELET\_SYSTEM\_PODS\_ARGS变量中去除allowed-unsafe-sysctls参数或置空。  3. 重启kubelet服务。例如：  systemctl daemon-reload  systemctl restart kubelet.service |

A.7 CNI插件和网络策略安全配置要求

A.7.1文件目录安全

表A. 23

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.７.1.1 | 在k8s master节点运行如下命令进行检查，验证文件权限是否为600或者更严格，其中<path/to/cni/file>为容器网络接口文件的路径。  Stat -c %a <path/to/cni/files> | 如果不符合要求则对文件权限进行修改，可参考如下命令，在master节点上执行命令将文件权限修改为600。  Chmod 600 <path/to/cni/files> |
| 5.7.1.2 | 在k8s master节点运行如下命令进行检查，验证文件属主是否为root:root。其中<path/to/cni/file>为容器网络接口文件的路径。  Stat -c %U:%G <path/to/cni/files> | 如果不符合规则要求则修改文件属主，可参考如下命令，在master节点上执行命令修改文件属主为root:root 。  chown root:root <path/to/cni/files> |

A.７.2 网络策略安全配置

表A. 24

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 5.7.2.1 | 查看集群使用的CNI插件的文档，并确认它支持入口和出口网络策略。 | 如果正在使用的CNI插件不支持网络策略，则应考虑使用不同的插件，或找到限制Kubernetes集群中流量的替代机制 |
| 5.7.2.2 | 通过以下方面来设置合理的NetworkPolicy资源，以确保pod之间通信安全，还应确保所有 NetworkPolicy 至少针对一个 Pod。  **1、默认拒绝所有流量**：这是一种保守但安全的策略，它可以防止任何未经授权的流量进入或离开你的集群。  **2、允许特定流量**：在默认拒绝所有流量的基础上，你可以根据你的业务需求来允许特定的流量。你可以通过指定 podSelector、namespaceSelector、ipBlock 等字段来定义允许的流量源或目标，以及通过指定 ports、protocol 等字段来定义允许的流量类型。  **3、隔离不同层次或角色的流量**：在一个复杂的集群中，你可能需要隔离不同层次或角色的流量，例如前端和后端、开发和生产、用户和管理员等。你可以通过使用标签和选择器来实现这一点，给不同层次或角色的 Pod 添加不同的标签，并在网络策略中使用 podSelector 或 namespaceSelector 来选择它们。  **4、允许特定的外部流量**：在一些场景下，你可能需要允许你的 Pod 访问或被访问一些外部的资源，例如互联网、其他集群、其他网络等。你可以通过使用 ipBlock 字段来实现这一点，指定一个 CIDR 范围或一个 IP 地址来定义外部的流量源或目标 | 根据需要配置合理的networkpolicy策略 |
| 5.7.2.3 | 通过以下命令查看所有命名空间下的networkpolicy。  Kubectl get networkpolicy –all-namespaces  确保集群中定义的每个命名空间至少有一个networkpolicy | 根据需要创建networkpolicy对象 |

**附 录 B**

**（资料性附录）**

**检查方法和修复方法**

B.1 docker运行时安全配置

B.1.1 文件目录安全

表B.1

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.1.1.1 | 1. 找出docker.service文件的配置：systemctl show -p FragmentPath docker.service  2. 如果文件不存在，则忽略这条规则。如果文件存在，执行指令：stat -c %U:%G /usr/lib/systemd/system/docker.service | grep -v root:root，如果返回空，则表示该指令检查docker.service文件是否为root权限。 | 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。如果该文件确实存在，  您应该使用正确的文件路径执行以下命令来设置所有权和组。  例如,  chown root:root/usr/lib/systemd/system/docker.service |
| 6.1.1.2 | 执行指令stat -c %U:%G /var/run/docker.sock | grep -v root:docker 确认socket文件的所有者设定是否正确，正确的设定输出应该为空 | 应该执行以下命令:  chown root:docker /var/run/docker.sock  对于默认Docker,这会将所有者设置为root,将组所有者设置为docker |
| 6.1.1.3 | 执行指令stat -c %U:%G /etc/docker/daemon.json | grep -v root:root 确认json配置文件的所有者设定是否正确，正确的设定输出应该为空 | 应该执行以下命令:  chown root:root /etc/docker/daemon.json  对于默认Docker,这会将所有者设置为root,将组所有者设置为root |
| 6.1.1.4 | 执行指令：stat -c %a /etc/docker/daemon.json 该指令检查daemon.json文件的访问权限否为600或更高的等级。 | 如果文件不存在，则此建议不适用。如果该文件存在，  使用正确的文件路径执行以下命令，以将文件权限设置为600。  例如，  chmod 600 /etc/docker/daemon.json |
| 6.1.1.5 | 1. 找出docker.service文件的配置：systemctl show -p FragmentPath docker.service  2. 如果文件不存在，则忽略这条规则。如果文件存在，应该使用正确的文件路径执行指令：stat -c %a /usr/lib/systemd/system/docker.service 该指令检查docker.service文件的访问权限否为644或更高的等级。 | 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，  使用正确的文件路径执行以下命令，以将文件权限设置为644。  例如，  chmod 644 /usr/lib/systemd/system/docker.service |
| 6.1.1.6 | 1、确认是否有/etc/sysconfig/docker文件  2、如果文件不存在，则忽略这条规则。如果文件存在，执行指令stat -c %U:%G /etc/sysconfig/docker | grep -v root:root 确认文件的所有者设定是否正确，正确的设定输出应该为空 | 如果文件不存在，则此建议不适用。如果该文件存在，  应该执行以下命令  chown root:root /etc/sysconfig/docker  这会将文件的所有者和组设置为root. |
| 6.1.1.7 | 1. 找出docker.socket文件的配置：systemctl show -p FragmentPath docker.socket  2. 如果文件不存在，忽略本条规则，如果文件存在，执行指令stat -c %U:%G <path to docker.socket> | grep -v root:root 确认文件的所有者设定是否正确，正确的设定输出应该为空 | 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，  执行以下命令，设置所有者和组为root:root。  例如,  chown root:root/usr/lib/systemd/system/docker.socket |
| 6.1.1.8 | 1. 找出docker.socket文件的配置：systemctl show -p FragmentPath docker.socket  2. 如果文件不存在，忽略本条规则，如果文件存在，执行指令stat -c %a <path to docker.socket>检查访问权限是否为640或者更严格的权限 | 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.如果该文件确实存在,则您  应该执行以下命令,包括正确的文件路径,以验证  文件权限被限制性地设置为640或更高.  例如,  chmod 640 /usr/lib/systemd/system/docker.socket |
| 6.1.1.9 | 执行指令stat -c %U:%G /etc/docker | grep -v root:root 确认文件的所有者设定是否正确，正确的设定输出应该为空 | 要解决此问题，应该运行以下命令:  chown root:root /etc/docker  这会将目录的所有者和组设置为root。 |
| 6.1.1.10 | 执行指令stat -c %a /etc/docker检查访问权限是否为750或者更严格的权限 | 应该运行以下命令:  chmod 750 /etc/docker  这会将目录的权限设置为750. |

B.1.2 守护进程安全

表B.2

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.1.2.1 | 1. 检查/etc/docker/daemon.json中是否有关于dm.basesize的配置 或者  2. 执行ps -ef |grep dockerd检查是否有关于--storage-opt dm.basesize的选项 | dm.basesize 用于控制device mapper的base device分配的大小，如果镜像大于该值会导致镜像不能启动，所以除非需要，否则不要设置 --storage-opt dm.basesize。basesize的默认大小为10G。 |
| 6.1.2.2 | 1. 执行docker info --format '{{ .LiveRestoreEnabled }}'指令，输出应该为true 或者  2. 执行ps -ef|grep dockerd检查--live-restore选项，应该为true | live-restore功能确保在容器引擎异常退出时，正在运行的容器仍然正常运行。在Docker daemon 启动时应该中配置 --live-restore 参数。  例如，  dDockerd --live-restore-实时还原 |
| 6.1.2.3 | 1. 执行 ps -ef|grep dockerd 指令检查--log-level选项，应该为info。或者  2. 检查/etc/docker/daemon.json文件的log-level选项，应该为info。 | 确保Docker守护程序配置文件包含以下配置  "log-level":"info"  或者，如下运行Docker守护程序:  dockerd --log-level ='info' |
| 6.1.2.4 | 1. 执行 ps -ef|grep dockerd 指令检查--iptables选项，应该没有出现或者没有被设定为false。   检查/etc/docker/daemon.json，是否有关于iptables的配置 | 执行 ps -ef|grep dockerd 指令检查--iptables选项，应该没有出现或者没有被设定为false。  检查/etc/docker/daemon.json，不应该有iptables的配置或者不应该设置为false. |
| 6.1.2.5 | 执行指令docker info --format 'Storage Driver: {{ .Driver }}'确保docker没有使用aufs作为storage driver | 不要使用aufs作为存储驱动程序。  例如,请勿按以下方式启动Docker守护程序:  dockerd --storage-driver aufs |
| 6.1.2.6 | 执行docker version --format '{{ .Server.Experimental }}'指令，确保输出为false | 不应在生产环境中将 --experimental 作为运行时参数传递给Docker守护程序。 |
| 6.1.2.7 | 执行 ps -ef | grep dockerd 指令，确保 --no-new-privileges 参数存在且设置为 true。 | 运行 Docker 守护进程使用命令：dockerd --no-new-privileges |

B.1.3 宿主机安全

表B.3

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.1.3.1 | 执行docker version查看当前版本是否在高危漏洞的影响范围内 | 及时升级docker daemon版本以彻底修复安全漏洞 |
| 6.1.3.2 | 1. 执行docker info -f '{{ .DockerRootDir }}'获取docker根目录，比如是/var/lib/docker   2、执行grep '/var/lib/docker\s' /proc/mounts，在返回中查看是否有输出  或者执行mountpoint -- "$(docker info -f '{{ .DockerRootDir }}')"可以查看容器根目录是否是一个挂载点 | 1、执行docker info -f '{{ .DockerRootDir }}'获取docker根目录，比如是/var/lib/docker  2、在节点创建时应将容器根目录/var/lib/docker设置为一个挂载点以独立分区 |

B.2 containerd运行时安全要求

B.2.1 文件目录安全

表B.4

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.2.1.1 | 执行containerd --version查看当前版本是否在高危漏洞的影响范围内 | 及时升级containerd daemon版本以彻底修复安全漏洞 |
| 6.2.1.2 | 执行命令： cat /etc/containerd/config.toml|grep 'level = '  检查是否配置为level = "info" | 确保containerd daemon配置文件/etc/containerd/config.toml中设置了:  [debug]  level = "info" |
| 6.2.1.3 | 执行grep '/var/lib/containerd \s' /proc/mounts，在返回中查看是否输出/var/lib/containerd  执行mountpoint -- "/var/lib/containerd"可以查看容器根目录是否是一个挂载点 | 在节点创建时应将容器根目录/var/lib/containerd设置为一个挂载点以独立分区 |
| 6.2.1.4 | stat -c %U:%G /etc/containerd/config.toml |grep -v root:root，正确的设定输出应该为空。 | 确保containerd配置文件的所属是root:root  可以执行以下命令：  chown root:root /etc/containerd/config.toml |
| 6.2.1.5 | 执行命令:  stat -c %a /etc/containerd/config.toml;  检查访问权限是否为640或者更严格的权限 | 运行如下命令：  chmod 640 /etc/containerd/config.toml  确保containerd配置文件/etc/containerd/config.toml的权限640 |
| 6.2.1.6 | 执行命令:  stat -c %a /usr/bin/containerd;  检查访问权限是否为750或者更严格的权限 | 运行如下命令：  chmod 750 /usr/bin/containerd  确保containerd配置/usr/bin/containerd的权限750 |
| 6.2.1.7 | stat -c %U:%G /usr/bin/containerd |grep -v root:root，正确的设定输出应该为空 | 运行如下命令：  chown root:root /usr/bin/containerd  确保containerd文件/usr/bin/containerd所属组root:root |
| 6.2.1.8 | stat -c %U:%G /usr/lib/systemd/system/containerd.service|grep -v root:root，该指令检查该文件属主是否为root:root,正确的设定输出应该为空 | 运行如下命令：  chown root:root /usr/lib/systemd/system/containerd.service  确保containerd文件/usr/lib/systemd/system/containerd.service所属组root:root |
| 6.2.1.9 | 执行命令:  stat -c %a /usr/lib/systemd/system/containerd.service;  检查访问权限是否为640或者更严格的权限 | 运行如下命令：  chmod 640 /usr/lib/systemd/system/containerd.service  确保/usr/lib/systemd/system/containerd.service的权限640 |
| 6.2.1.10 | 执行命令:  stat -c %a /var/lib/containerd.sock;  检查访问权限是否为640或者更严格的权限 | 运行如下命令：  chmod 640 /var/lib/containerd.sock  确保/var/lib/containerd.sock的权限640 |
| 6.2.1.11 | stat -c %U:%G /var/lib/containerd.sock|grep -v root:root，该指令检查该文件属主是否为root:root,正确的设定输出应该为空 | 运行如下命令：  chown root:root /var/lib/containerd.sock  确保/var/lib/containerd.sock的用户组root:root |
| 6.2.1.12 | 1．执行以下命令获取元数据目录存储位置  grep 'root' /etc/containerd/config.toml  2．执行以下命令查看目录访问权限是否为700  stat -c %a /var/lib/containerd | 1．如果权限不符合要求，则按以下命令修改目录权限。  chmod 700 /var/lib/containerd |
| 6.2.1.13 | 1．执行以下命令获取元数据目录存储位置  grep 'root' /etc/containerd/config.toml  2．执行以下命令查看目录属主数组  stat -c %U:%G /var/lib/containerd | 1．如果属主属组非root，则按以下命令修改目录属主属组。  chown root:root /var/lib/containerd |

B.2.2 守护进程安全

表B.5

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.2.2.1 | 方法一 查看/etc/containerd/config.toml中endpoint的值中是否存在以"http://"字段：   1. grep "http://" /etc/containerd/config.toml;   输出结果为空时满足要求 方法二 步骤一：获取<registry配置文件所在目录>   1. grep "config\_path" /etc/containerd/config.toml   步骤二：   1. cd <registry配置文件所在目录>   步骤三：获取<每个registry的配置文件>   1. find . -name hosts.toml   步骤四：   1. grep "http://" <每个registry的配置文件> ;   步骤四输出结果为空时满足要求。 | 确保registry配置为https |
| 6.2.2.2 | 1、查看/etc/containerd/config.toml中“level”的值是否为非空grep "level =" /etc/containerd/config.toml  检测结果及判定依据：输出结果包含info，说明满足要求。  或者  2、查看守护进程启动参数是否包含--log-level  ps -ef|grep containerd|grep log-level,如果不为空则满足要求 | 1、在/etc/containerd/config.toml配置文件中配置level为info级别  [debug]  level = “info”  或者  2、启动守护进程时指定--log-level参数  containerd --log-level info |
| 6.2.2.3 | 查看/etc/containerd/config.toml中“snapshotter”的值是否为“aufs”   1. grep "snapshotter =" /etc/containerd/config.toml   检测结果及判定依据：输出结果中不包含“aufs”字段时满足要求 | 不要配置aufs作为存储驱动 |
| 6.2.2.4 | 1.执行以下命令获取containerd服务文件位置  方法1：搜索服务文件  locate containerd.service  或  find / -name containerd.service  方法2：查找服务文件  systemctl cat containerd.service  2．执行以下命令查看Delegate是否配置为yes  grep 'Delegate' /usr/lib/systemd/system/containerd.service  说明：containerd.service 文件通常位于 /etc/systemd/system/ 或 /usr/lib/systemd/system/ 目录。 | 1．编辑containerd.service服务文件，参考以下内容完成 Delegate 配置，例如：  [Service]  Delegate=yes  2. 加载重启 containerd 服务使配置生效。 |
| 6.2.2.5 | 1.执行以下命令获取containerd服务文件位置  方法1：搜索服务文件  locate containerd.service  或  find / -name containerd.service  方法2：查找服务文件  systemctl cat containerd.service  2．执行以下命令查看KillMode是否配置为process  grep 'KillMode' /usr/lib/systemd/system/containerd.service  说明：containerd.service 文件通常位于 /etc/systemd/system/ 或 /usr/lib/systemd/system/ 目录。 | 1．编辑containerd.service服务文件，参考以下内容完成 KillMode 配置，例如：  [Service]  KillMode=process  2. 加载重启 containerd 服务使配置生效。 |

B.2.3宿主机安全

表B.6

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.2.3.1 | 查看/etc/containerd目录的属主和属组：  stat -c %U:%G /etc/containerd  输出结果如下： root:root | 如果不符合规则要求，则修改文件属主属组，执行如下命令。在节点上执行命令将文件属主属组修改为root:root。  chown root:root /etc/containerd |
| 6.2.3.2 | 查看/etc/containerd目录的权限：  stat -c %a /etc/containerd  输出结果如下： 700 | 如果不符合规则要求，则修改文件权限，执行如下命令。在节点上执行命令将文件权限修改为700。  chmod 700 /etc/containerd |
| 6.2.3.3 | 查看Containerd CA证书路径  cat /etc/containerd/config.toml|grep "ca\_file =" ，路径比如是/opt/caas/ssl/containerd/ca.pem   1. 执行stat -c %U:%G /opt/caas/ssl/containerd/ca.pem 查看属主和属组，结果如果是root:root则满足要求。 2. 执行stat -c %a /opt/caas/ssl/containerd/ca.pem，预期结果应该是600 | 如果不符合规则要求，则修改文件权限，执行如下命令。在节点上执行命令将文件属主属组修改为root:root，将文件权限修改为600。  chown root:root /opt/caas/ssl/containerd/ca.pem  chmod 600 /opt/caas/ssl/containerd/ca.pem |

B.2.4 cri插件安全

表B.7

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.2.4.1 | 1．执行以下命令查看 SystemdCgroup 配置是否为true。  grep 'SystemdCgroup' /etc/containerd/config.toml  说明：如果系统运行在 systemd 上，并且对性能和资源管理有较高要求，尤其是与 Kubernetes 集成的场景，建议启用SystemdCgroup，否则此项不适用。 | 1. 编辑config.toml配置文件，参考以下内容完成 SystemdCgroup 配置，例如：  [plugins."io.containerd.grpc.v1.cri".  containerd.runtimes.runc.options]  SystemdCgroup = true  2. 重启 containerd 服务以使配置生效。 |
| 6.2.4.2 | 1．执行以下命令查看 enable\_tls\_streaming 配置是否为true。  grep 'enable\_tls\_streaming' /etc/containerd/config.toml  说明：启用TLS流式传输（设为true）可以加密kubelet与容器运行时之间的通信，增强集群安全性，但可能会带来性能开销。若运行在受信任的网络环境中，且性能要求极高，此项不适用。 | 1. 编辑config.toml配置文件，参考以下内容完成 SystemdCgroup 配置，例如：  [plugins."io.containerd.grpc.v1.cri"]  enable\_tls\_streaming = true  2. 重启 containerd 服务以使配置生效。 |
| 6.2.4.3 | 1. 执行以下命令查看 disable\_cgroup 是否存在，如果存在是否设置为false   cat /etc/containerd/config.toml | grep disable\_cgroup | 1.编辑config.toml文件，去掉disable\_cgroup参数或将其值改为false  2.重启 containerd 服务以使配置生效。 |

B.3 cri-o运行时安全配置

B.3.1 文件目录安全

表B.8

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.3.1.1 | 1. 找出crio.service文件的配置:  systemctl show -p FragmentPath crio.service  2. 如果文件不存在，则忽略这条规则。如果文件存在，执行指令:  stat -c %U:%G /usr/lib/systemd/system/crio.service |grep root:root，该指令检查crio.service属主是否为root:root。 | 步骤1:找出文件位置:  systemctl show -p FragmentPath crio.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，应该使用正确的文件路径执行以下命令来设置crio.service文件的属主，例如:  chown root:root /usr/lib/systemd/system/crio.service |
| 6.3.1.2 | 1. 找出crio.service文件的配置:  systemctl show -p FragmentPath crio.service  2. 如果文件不存在，则忽略这条规则。如果文件存在，执行指令:  stat -c %a /usr/lib/systemd/system/crio.service 该指令检查crio.service文件的访问权限否为644或更严格等级。 | 步骤1:找出文件位置:  systemctl show -p FragmentPath crio.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，使用正确的文件路径执行以下命令:  chmod 644 /usr/lib/systemd/system/crio.service |
| 6.3.1.3 | 1. 找出crio.socket文件的配置:  systemctl show -p FragmentPath crio.socket  2. 如果文件不存在，忽略本条规则，如果文件存在，执行指令:  stat -c %U:%G /var/run/crio/crio.sock ;  确认文件的所有者设定是否为root:root | 如果文件不存在，则此建议不适用。如果该文件存在，应该使用正确的文件路径执行以下命令来设置crio.sock文件的属主，例如:  chown root:root /var/run/crio/crio.sock |
| 6.3.1.4 | 执行命令:  stat -c %a /var/run/crio/crio.sock 该指令检查crio.sock文件的访问权限否为644或更严格等级。 | 如果文件不存在，则此建议不适用。如果该文件存在，使用正确的文件路径执行以下命令:  chmod 644 /var/run/crio/crio.sock |
| 6.3.1.5 | 执行命令:  stat -c %U:%G /etc/sysconfig/crio | grep -v root:root;  确认文件的所有者设定是否正确，正确的设置应该为空 | 如果文件不存在，则此建议不适用。如果该文件存在，应该使用正确的文件路径执行以下命令来设置crio.sock文件的属主，例如:  chown root:root /etc/sysconfig/crio |
| 6.3.1.6 | 执行命令:  stat -c %a /etc/sysconfig/crio;  检查访问权限是否为644或者更严格的权限 | 如果文件不存在，则此建议不适用。如果该文件存在，使用正确的文件路径执行以下命令:  chmod 644 /etc/sysconfig/crio |
| 6.3.1.7 | 执行命令:  stat -c %U:%G /etc/crio | grep -v root:root;  确认文件的所有者设定是否正确，正确的设置应该为空 | 如果文件不存在，则此建议不适用。如果该文件存在，应该使用正确的文件路径执行以下命令来设置/etc/crio的属主，例如:  chown root:root /etc/crio |
| 6.3.1.8 | 执行命令:  stat -c %a /etc/crio;  检查访问权限是否为755或者更严格的权限 | 应该运行以下命令:  chmod 755 /etc/crio  这会将目录的权限设置为755. |
| 6.2.1.9 | 执行crio --version查看当前版本是否在高危漏洞的影响范围内 | 及时升级crio daemon版本以彻底修复安全漏洞 |

B.3.2 守护进程安全

表B.9

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.3.2.1 | 执行命令:  ps -ef|grep crio;  指令检查--log-level选项，应该为info  如果有文件/etc/crio/crio.conf 检查/etc/crio/crio.conf文件的log-level配置项为: info | 确保CRI-O守护程序配置文件包含以下配置  log-level info |
| 6.3.2.2 | 执行命令:  grep 'driver =' /etc/containers/storage.conf | 不要使用aufs作为存储驱动程序。  例如,请勿按以下方式启动CRI-O守护程序:  criod --storage-driver aufs |

B.3.3 宿主机安全

|  |  |  |
| --- | --- | --- |
| **序号** | **检查方法** | **修复方法** |
| 6.3.3.1 | 在CRI-O主机上执行以下命令:  grep '/var/lib/cri-o \s' /proc/mounts | 对于新安装的系统，请创建一个单独的分区用于挂载点: /var/lib/crio。对于已经安装过的系统，请在Linux中使用逻辑卷管理器(LVM)来创建一个新的分区。 |

**附 录 C**

**（资料性附录）**

**检查方法和修复方法**

C.7 工作负载安全配置

C.7.1 启动安全

表C.1

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 7.1.1 | 执行以下命令，结果不匹配(AppArmorProfile=\s\*$)|(AppArmorProfile=<no value>)  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: AppArmorProfile={{ .AppArmorProfile }}' 2>/dev/null  执行以下命令，结果不匹配(SecurityOpt=\s\*$)|(SecurityOpt=<no value>)  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: SecurityOpt={{ .HostConfig.SecurityOpt }}' 2>/dev/null | 如果Linux操作系统能够适用AppArmor。遵循下列步骤：1. 确认AppArmor已安装。如尚未安装，安装AppArmor。2. 为docker容器创建或导入一个AppArmor配置文件。3. 将该配置文件设置为enforcing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，也可以使用docker的默认AppArmor配置文件。docker的默认AppArmor配置文件docker-default默认适用于运行中的容器，存放在/etc/apparmor.d/docker目录下。  1. 如果你的Linux操作系统适用于SELinux，遵循下列步骤：2. 设置SELinux状态。3. 设置SELinux策略。4. 为docker容器创建或导入SELinux策略模板。以守护进程模式带SELinux启动docker，见下：docker daemon --selinux-enabled使用如下安全选项启动docker容器，见下：docker run --interactive --tty --security-opt label=level:TopSecret centos /bin/bash |
| 7.1.2 | 执行以下命令，匹配User=\s\*.+且不含有User=\s\*root 如果为空，说明是root用户运行。  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: User={{.Config.User}}' 2>/dev/null | egrep 'User=' | 确保容器镜像的Dockerfile包含以下指令：  USER <username or ID> |
| 7.1.3 | 执行以下命令，匹配Not Empty 查询出容器的Capability能力并由用户来确认是否是需要。  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: CapAdd={{ .HostConfig.CapAdd }} CapDrop={{ .HostConfig.CapDrop }}' 2>/dev/null | egrep 'CapAdd=.+\s+CapDrop=.+$' | 删除容器进程明确要求的Capability以外的所有Capability |
| 7.1.4 | 执行以下命令，结果匹配Privileged\s\*=\s\*false。  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: Privileged={{ .HostConfig.Privileged }}' 2>/dev/null | 限制特权容器的准入。在k8s 1.21以下版本，向集群中具有用户工作负载的每个命名空间添加PSP策略；在K8S 1.21以上版本建议开启Pod Security Admission |
| 7.1.5 | 执行以下命令，结果不匹配Source:(/|/boot|/dev|/etc|/lib|/proc|/sys|/usr)\s+.\*RW:true。  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: Volumes={{ .Mounts }}' 2>/dev/null | 尤其是在读写模式下。如果产品由于特殊场景必须c挂载，必须只以只读权限挂载 |
| 7.1.6 | 执行以下命令，结果不匹配(Ports=map\[(\d{1,3})/tcp)|(Ports=map\[10[01]\d/tcp)|(Ports=map\[102[0-3]/tcp))。  docker ps --quiet | xargs docker inspect --format '{{ .Id }}: Ports={{ .NetworkSettings.Ports }}' 2>/dev/null | 在启动容器时不要将容器端口映射到特权主机端口上。另外，确保dockerfile文件里没有类似容器到主机特权端口的声明 |
| 7.1.7 | 执行以下命令，取得所有镜像打开的端口信息由用户确认是否是需要的。  docker ps --quiet | xargs docker inspect --format '{{.Id}}：Ports = {{.NetworkSettings.Ports}}' | 修复容器镜像的dockerfile，只暴露容器化应用程序所需的端口。启动容器时，不使用“-P”（大写）或“--publish-all”标志，即可完全忽略在dockerfile中定义的端口列表。使用“-p”（小写）或“--publish”标志，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 |
| 7.1.8 | 执行以下命令，结果不匹配ReadonlyRootfs=false  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: ReadonlyRootfs={{ .HostConfig.ReadonlyRootfs }}' 2>/dev/null | 添加一个“--read-only”标志，使容器的根文件系统以只读方式挂载。可以与容器卷组合使用，强制容器的进程只写入将被持久化的位置。例如，docker run --interactive --tty --read-only --volume /centdata centos /bin/bash这将运行具有只读根文件系统的容器，并将使用‘centdata’作为写操作的容器卷 |
| 7.1.9 | 执行以下命令，结果不匹配NetworkMode=host  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: NetworkMode={{ .HostConfig.NetworkMode }}' 2>/dev/null | 限制hostNetwork容器的创建。  在k8s 1.21以下版本，向集群中具有用户工作负载的每个命名空间添加PSP策略；在K8S 1.21以上版本建议开启Pod Security Admission |
| 7.1.10 | 执行以下命令，结果不匹配IpcMode=host  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: IpcMode={{ .HostConfig.IpcMode }}' 2>/dev/null | 限制 hostIPC 容器的创建。在k8s 1.21以下版本，向集群中具有用户工作负载的每个命名空间添加PSP策略；在K8S 1.21以上版本建议开启Pod Security Admission |
|  |  |  |
| 7.1.11 | 执行以下命令，结果不匹配PidMode=host  docker ps --quiet --all| xargs docker inspect --format '{{ .Id }}: PidMode={{ .HostConfig.PidMode }}' 2>/dev/null | 限制 hostPID 容器的创建。在k8s 1.21以下版本，向集群中具有用户工作负载的每个命名空间添加PSP策略；在K8S 1.21以上版本建议开启Pod Security Admission |
| 7.1.12 | 执行以下命令，判断结果是否为空，或符合最小的权限的需要  docker ps -q -a |xargs docker inspect --format '{{.Id}}:Devices={{.HostConfig.Devices}}' 2>/dev/null | 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，使用正确的最小权限 |
| 7.1.13 | 列出集群中每个命名空间使用的控制策略，确保每个策略都不允许NET\_RAW容器准入。 获取当前集群的POD名称，根据POD名称确认配置项NET\_RAW是否被删除。 kubectl get pod <name> -o=jsonpath='{.spec.requiredDropCapabilities }' | 限制带有NET\_RAW权限的容器的创建。在k8s 1.21以下版本，向集群中具有用户工作负载的每个命名空间添加PSP策略，确保 .spec.requiredDropCapabilities 包括 NET\_RAW 或 ALL；在K8S 1.21以上版本建议开启Pod Security Admission。 |
| 7.1.14 | 列出集群中每个命名空间使用的控制策略，确保每个策略都不允许使用hostPath.Socket的容器的准入。  以K8S配置项举例：  获取当前集群的POD名称，根据POD名称确认配置项是否存在volumes.hostPath，如存在则确认hostPath定义内容是否包含Socket。  kubectl get pod <name> -o=jsonpath='{.spec.hostPath}' | 确保严格控制对容器运行时套接字所在的文件系统访问。如果可能，限制为仅 root 用户可访问。  使用 Linux 内核命名空间等机制将 kubelet 与节点上运行的其他组件隔离。  确保限制或禁止使用包含容器运行时套接字的 hostPath 挂载， 无论是直接挂载还是通过挂载父目录挂载。此外，hostPath 挂载必须设置为只读，以降低攻击者绕过目录限制的风险。  限制用户对节点的访问，特别是限制超级用户对节点的访问。 |
| 7.1.15 | 列出集群中每个命名空间使用的控制策略，推荐每个策略都包含RuntimeDefault内容。  以K8S配置项举例：  获取当前集群的POD名称，根据POD名称确认配置项每个策略里是否使用了seccompProfile，且类型为type: RuntimeDefault。  kubectl get pod <name> -o=jsonpath='{.spec.seccompProfile}' | 1、对需要使用安全计算模式的容器，使用--seccomp-default命令行标志来运行kubelet  2、在容器配置文件中的securityContext部分增加:  spec:  securityContext:  seccompProfile:  type: RuntimeDefault |
|  | | |
| 7.1.16 | 列出集群中每个命名空间使用的控制策略，确保容器策略都不允许未配置securityContext、capabilities、allowPrivilegeEscalation的容器准入。  以K8S配置项举例：  获取当前集群的POD名称，根据POD名称确认配置项securityContext、capabilities、allowPrivilegeEscalation是否符合权限最小化要求。  kubectl get pod <name> -o=jsonpath='{.spec.allowPrivilegeEscalation }'  kubectl get pod <name> -o=jsonpath='{.spec.runAsUser }'  kubectl get pod <name> -o=jsonpath='{.spec.capabilities }' | 通过配置runAsUser，指定容器使用非root用户运行。  通过配置capabilities，使用capability精确控制容器的特权访问权限。  通过配置allowPrivilegeEscalation, 在不需要容器进程提权的场景，建议关闭“允许特权逃逸”的配置。 |
| 7.1.17 | 获取当前集群的POD名称，根据POD名称确认配置项windowsOptions.hostProcess是否为false，如为True则确认其权限定义内容。  kubectl get pod <name> -o=jsonpath='{.spec.windowsOptions}' | 应限制使用hostProcess卷的容器的准入。在k8s 1.21以下版本，向集群中具有用户工作负载的每个命名空间添加PSP策略；在K8S 1.21以上版本建议开启Pod Security Admission。 |
| 7.1.18 | 列出集群中每个命名空间使用的控制策略，确保策略中尽量避免允许使用HostPath卷策略的容器准入，在必须使用的情况下应检查HostPath卷定义的内容是否为满足运行的最小范围。  以K8S配置项举例：  获取当前集群的POD名称，根据POD名称确认配置项是否存在volumes.hostPath，如存在则确认hostPath定义内容。  kubectl get pod <name> -o=jsonpath='{.spec.hostPath}' | hostPath卷可能会暴露特权系统凭据（例如 Kubelet）或特权 API（例如容器运行时套接字），可用于容器逃逸或攻击集群的其他部分。  应限制使用hostPath卷的容器的准入。在k8s 1.21以下版本，向集群中具有用户工作负载的每个命名空间添加PSP策略；在K8S 1.21以上版本建议开启Pod Security Admission。 |
| 7.1.19 | 列出集群中每个命名空间使用的控制策略，确保策略中尽量对hostPorts进行限制，应检查hostPorts定义的端口范围是否为满足运行的最小范围。  以K8S配置项举例：  获取当前集群的POD名称，根据POD名称确认配置项是否存在hostPorts，如存在则确认hostPorts定义的端口范围。  kubectl get pod <name> -o=jsonpath='{.spec.hostPorts }' | 限制使用HostPorts的容器的准入。在k8s 1.21以下版本，向集群中具有用户工作负载的每个命名空间添加PSP策略；在K8S 1.21以上版本建议开启Pod Security Admission。 |

C.7.2 身份认证和访问控制

表C.2

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 7.2.1 | 执行如下命令，查询pod的使用账户：  kubectl get po -n example-namespace [podname] -o yaml | grep serviceAccount  执行如下命令查看默认用户是否具有对应权限，如果返回为false则默认用户不含有对应权限，一般情况下默认用户不包含任何权限。 | 当Kubernetes工作负载需要特定访问Kubernetes API服务器时，需要创建显式的服务帐户。 修改每个默认服务帐户的配置以包含此值：automountServiceAccountToken: false |
| 7.2.2 | 执行如下命令查看service account属性  kubectl get sa -n test -o yaml  查看集群中的Pod和服务帐户对象，并确保设置了以下选项，除非资源明确要求此访问权限。  automountServiceAccountToken: false | 修改Pod定义和service account定义，禁止不必要的token挂载 |
| 7.2.3 | 检查sc所绑定的集群角色使用的权限，执行如下命令，可以看到该默认用户所使用的集群角色为如下:  kubectl get clusterrolebinding -n example-namespace -o yaml | grep default -C5  roleRef  查看该集群角色所使用的权限见下：  kubectl get clusterrole -n example-namespace test -o yaml | 当Kubernetes工作负载需要特定访问Kubernetes API服务器时，需要创建显式的服务帐户,且按需分配最小化权限。 |
| 7.2.4 | 查看具有Kubernetes API中service account token子资源访问权限的用户。 | 尽可能删除对service account token子资源的访问。 |
| 7.2.5 | 1．安装jq工具  2. 执行如下命令构建目标集群全局和指定ns中包含pod/exec的create权限的RBAC角色索引文件：  kubectl get clusterroles -o json | jq '[.items[] | select(  .rules[] as $rule |  (any($rule.resources[]?; . == "pods/exec")) and  (any($rule.verbs[]?; . == "create"))  ) | .metadata.name] | unique' > /tmp/clusterroles-pod-exec.json  kubectl get roles -n your-namespace -o json | jq '[.items[] | select(  .rules[] as $rule |  (any($rule.apiGroups[]?; . == "")) and  (any($rule.resources[]?; . == "pods/exec")) and  (any($rule.verbs[]?; . == "create"))  ) | .metadata.name] | unique' > /tmp/roles-pod-exec.json  3.基于步骤2中生成的索引文件分别寻找集群维度和指定特权ns中使用了pod/execn的create权限的风险绑定项:  kubectl get clusterrolebindings -o json | jq -r --slurpfile clusterRolesPodExec /tmp/clusterroles-pod-exec.json '  ($clusterRolesPodExec[0] | map({ (.): true }) | add) as $clusterRolesMap |  .items[] |  select($clusterRolesMap[.roleRef.name]) |  "\(.metadata.name) \(.roleRef.name)"'  kubectl get rolebindings -n your-namespace -o json | jq -r --slurpfile rolesPodExec /tmp/roles-pod-exec.json '  ($rolesPodExec[0] | map({ (.): true }) | add) as $rolesMap |  .items[] |  select($rolesMap[.roleRef.name]) |  "\(.metadata.namespace) \(.metadata.name) \(.roleRef.name)"' |  |
| 7.2.6 | 1．安装jq工具  2. 执行如下命令构建目标集群全局和指定ns中包含pod/ephemeralcontainers的create权限的RBAC角色索引文件：  kubectl get clusterroles -o json | jq '[.items[] | select(.rules[] as $rule | (any($rule.resources[]?; . == "pods/ephemeralcontainers")) and (any($rule.verbs[]?; . == "create"))) | .metadata.name] | unique' > /tmp/clusterroles-pod-ephemeral.json  kubectl get roles -n your-namespace -o json | jq '[.items[] | select(  .rules[] as $rule |  (any($rule.apiGroups[]?; . == "")) and  (any($rule.resources[]?; . == "pods/ephemeralcontainers")) and  (any($rule.verbs[]?; . == "create"))  ) | .metadata.name] | unique' > /tmp/roles-pod-ephemeral.json  3.基于步骤2中生成的索引文件分别寻找集群维度和指定特权ns中使用了pod/ephemeralcontainers的create权限的风险绑定项:  kubectl get clusterrolebindings -o json | jq -r --slurpfile clusterRolesPodEphemeral /tmp/clusterroles-pod-ephemeral.json '  ($clusterRolesPodEphemeral[0] | map({ (.): true }) | add) as $clusterRolesMap |  .items[] |  select($clusterRolesMap[.roleRef.name]) |  "\(.metadata.name) \(.roleRef.name)"'  kubectl get rolebindings -n your-namespace -o json | jq -r --slurpfile rolesPodEphemeral /tmp/roles-pod-ephemeral.json '  ($rolesPodEphemeral[0] | map({ (.): true }) | add) as $rolesMap |  .items[] |  select($rolesMap[.roleRef.name]) |  "\(.metadata.namespace) \(.metadata.name) \(.roleRef.name)"' | 在确认业务不依赖的前提下删除检查结果中防护的不当RBAC特权绑定 |
| 7.2.7 | 1．安装jq工具  2. 执行如下命令构建目标集群RBAC权限索引文件，并输出在/tmp/role-permissions-index.json文件中：  kubectl get clusterroles,roles --all-namespaces -o json | jq -r '  reduce .items[] as $item (  {}; .[$item.metadata.name] = {rules: ($item.rules)}  )  ' > /tmp/role-permissions-index.json  3 基于索引文件执行如下命令在集群中检查是否有包含create和approve csr相关权限的高危绑定：  kubectl get rolebindings,clusterrolebindings --all-namespaces -o json | jq -r --slurpfile permissions\_index /tmp/role-permissions-index.json '  $permissions\_index[0] as $index |  .items[] |  select(  any($index[.roleRef.name].rules[];  (  (.apiGroups | if type == "array" then . else [] end) | any(. == "certificates.k8s.io")  ) and  (  ((.verbs | if type == "array" then . else [] end) | any(. == "create" or . == "get" or . == "list" or . == "watch")) and  ((.resources | if type == "array" then . else [] end) | any(. == "certificatesigningrequests"))  ) or  (  ((.verbs | if type == "array" then . else [] end) | any(. == "update")) and  ((.resources | if type == "array" then . else [] end) | any(. == "certificatesigningrequests/approval"))  ) or  (  ((.verbs | if type == "array" then . else [] end) | any(. == "approve")) and  ((.resources | if type == "array" then . else [] end) | any(. == "signers"))  )  )  ) | .metadata.name  ' | 在确认业务不依赖的前提下删除检查结果中防护的不当RBAC特权绑定 |
| 7.2.8 | 查看有权访问集群的所有凭据的列表，并确保未使用system:masters组 | 从集群中的所有用户中删除system:masters组 |
| 7.2.9 | 检查sc所绑定的集群角色使用的权限，执行如下命令，可以看到该默认用户所使用的集群角色为如下:  kubectl get clusterrolebinding -n example-namespace -o yaml | grep default -C5  roleRef  查看该集群角色所使用的权限见下：  kubectl get clusterrole -n example-namespace test -o yaml  apiVersion: rbac.authorization.k8s.io/v1  kind: ClusterRole  metadata:  name: impersonator  rules:  - apiGroups: [""]  resources: ["users", "groups", "serviceaccounts"]  verbs: ["impersonate"] | 集群分配权限时最小化权限，谨慎分配Bind、Impersonate和Escalate权限。如无必要删除相关的verb和 clusterRole 。 |
| 7.2.10 | 通过查看有权访问cluster-admin角色的ClusterRoleBindings输出，获取有权访问cluster-admin角色的主体列表。查看列出的每个子项，确保它必须使用集群管理员权限。  kubectl get clusterrolebindings -o=custom-columns=NAME:.metadata.name,ROLE:.roleRef.name,SUBJECT:.subjects[\*].name |grep cluster-admin | 将所有clusterrolebindings标识到cluster-admin角色。检查是否使用了这些角色，以及它们是否需要此角色，或者它们是否可以使用权限较少的角色。在可能的情况下，首先将用户绑定到较低特权的角色，然后删除clusterrolebinding到cluster-admin角色：  kubectl delete clusterrolebinding [name]  默认只有system:masters关联了cluster-admin角色。 |
| 7.2.11 | 使用如下命令排查roles和clusterroles中的通配符： 检索集群中每个命名空间中定义的角色，并检查通配符（\*）  kubectl get roles --all-namespaces -o yaml  检索集群中定义的集群角色并检查通配符（\*）  kubectl get clusterroles -o yaml | 在可能的情况下，用特定的对象或操作替换clusterroles和roles中的任何通配符使用 |

C.7.3 防止信息泄露

表C.3

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 7.3.1 | 执行如下命令查看所有以环境变量形式使用secret的资源：  kubectl get all -o jsonpath='{range .items[?(@..secretKeyRef)]} {.kind} {.metadata.name} {"\n"}{end}' --all-namespaces | 如果可能的话，重写应用程序代码，从挂载的秘密文件中读取秘密，而不是从环境变量中读取 |
| 7.3.2 | 通过kubectl命令验证secret中敏感信息是否加密，对于较大的集群，可以通过命名空间对secret进行划分。  kubectl get secrets --all-namespaces |  |
| 7.3.3 | 执行如下命令对私钥文件进行检查，确认私钥文件是否明文输出  cat /etc/kubernetes/pki/ca.key | 建议通过集成KMC秘钥库管理工具，在部署集群前使用派生的工作秘钥对私钥文件进行加密。集群各组件在读取私钥文件后，对私钥文件在内存中进行解密 |
| 7.3.4 | 使用以下命令排查对secret拥有get、list或者watch权限的role，由用户自己判断是否需要。  kubectl get roles --all-namespaces -o yaml  kubectl get clusterroles -o yaml | 在可能的情况下，删除对集群中secret对象的get、list和watch访问 |

C.7.4 资源配额

表C.4

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 7.4.1 | 执行如下命令检查每个Pod的container是否都有CPU资源请求(resources.requests.cpu)及资源限制(resources.limits.cpu)  kubectl get pods --all-namespaces | 为Pod内的每个container增加CPU资源请求(resources.requests.cpu)及资源限制(resources.limits.cpu) |
| 7.4.2 | 执行如下命令检查每个Pod的container是否都有memory资源请求(resources.requests.memory)及资源限制(resources.limits.memory)  kubectl get pods --all-namespaces | 为Pod内的每个container增加memory资源请求(resources.requests.memory)及资源限制(resources.limits.memory) |
| 7.4.3 | 执行如下命令检查每个namespaces是否有设置配额  kubectl get ResourceQuota --all-namespaces | 按需为每个namespace设置资源配额，可控制的资源包括：CPU、内存、存储、pods、services、deployments、statefulsets等，详细参考：https://kubernetes.io/zh-cn/docs/concepts/policy/resource-quotas/ |
| 7.4.4 | 执行如下命令检查每个namespaces是否有设置资源的限制范围  kubectl get LimitRange --all-namespaces | LimitRange支持在一个命名空间中实施对每个 Pod 或 Container 最小和最大的资源使用量的限制，也可以在一个命名空间中实施对每个 PersistentVolumeClaim 能申请的最小和最大的存储空间大小的限制。请按需设置。 |
| 7.4.5 | 执行如下命令检查每个service是否有使用nodePort方式发布服务  kubectl get service --all-namespaces | 谨慎评估使用NodePort方式发布服务，若必须使用，建议定义单独的NetworkPolicy，并结合使用LoadBalancer或Ingress来提高可用性和扩展性 |

C.7.5 其他

表C.5

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 7.6.1 | 使用如下命令查看当前集群所有的命名空间：  kubectl get namespaces  并确保资源创建在期望的namespace下，确保这些命名空间是需要的且合理分配 | 按照Kubernetes文档，在需要时为部署中的对象创建命名空间。 不指定namespace时，资源默认创建在default命名空间下 |
| 7.6.2 | 使用下面命令列出默认命名空间中的对象：  kubectl get all  唯一的条目应该是系统管理的资源，如kubernetes服务 | 确保创建命名空间以允许Kubernetes资源的适当隔离，并且所有新资源都在特定命名空间中创建 |
| 7.6.3 | 通过以下方面来设置有效的ClusterRoles策略控制机制。  1. 最小化ClusterRole的使用  尽量使用Role而不是ClusterRole,减少权限范围。只有需要跨命名空间访问的资源才使用ClusterRole。  2. 避免过于宽泛的ClusterRole  不要创建过于宽泛的ClusterRole,访问权限应设置为所需最小权限。避免使用"\*"权限。  3. 注意默认ClusterRole  review默认的ClusterRole,如view、edit、admin等,移除不需要的权限控制。  4. 设置ClusterRoleBinding的服务帐号  使用rbac.authorization.k8s.io/aggregator=true注解,只允许集群管理员用户或服务账户绑定ClusterRole。  5. 合理设置RoleBinding引用review哪些RoleBinding被设置为引用ClusterRole,确保引用是必要的,防止权限范围扩大。  6. 定期审计ClusterRole和Binding开启审计日志,定期审计ClusterRole与ClusterRoleBinding的变更,发现风险权限提升情况。  7. 设置超时时间为ClusterRoleBinding设置时间限制,令牌定期更新,可以自动撤销已泄露的权限。  对于Kubernetes 1.23及以上版本  所有集群默认支持Pod安全准入机制，通过以下命令查看命名空间，检查是否合理设置PSA标签，确认是否有存在其他三方策略控制系统。  kubectl get namespaces -o yaml | 根据具体的需求设置合理的ClusterRoles策略控制机制，防止权限设置不当导致提权风险。  对于Kubernetes 1.23及以上版本  确保集群中每个包含用户工作负载的命名空间，都部署Pod安全准入策略或外部策略控制系统。  Pod安全性标准定义了三种不同的策略（Policy）广泛覆盖安全应用场景。 这些策略是叠加式的，安全级别从高度宽松至高度受限。  Pod安全准入特性被启用或者安装了 Webhook，可以配置名字空间以定义每个名字空间中 Pod 安全性准入控制模式。Kubernetes 定义了一组标签， 可以设置这些标签来定义某个名字空间上要使用的预定义的 Pod 安全性标准级别。 |
| 7.6.4 | 查看对Kubernetes API中PersistentVolume对象具有创建访问权限的用户。 | 尽可能删除对集群中PersistentVolume对象的创建访问权限。 |
| 7.6.5 | 查看有权访问Kubernetes API中节点对象的代理子资源的用户。 | 尽可能删除对节点对象的代理子资源的访问权限。 |
| 7.6.6 | 查看具有Kubernetes API中validatingwebhookconfigurations或mutatingwebhookconfigurations权限访问权限的用户。 | 尽可能删除用户validatingwebhookconfigurations或mutatingwebhookconfigurations的权限。 |

C.8 镜像安全配置要求

C.8.1 镜像创建安全

表C.6

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 8.1.1 | 应确保部署镜像的版本（tag）不为latest  使用docker images查看镜像版本是否是latest   1. docker images | grep latest   # 判断返回值若返回0则表示存在latest镜像，否则不存在   1. echo $?   使用kubectl查看部署的负载镜像版本是否是latest  # 取镜像地址  ①kubectl -n <命名空间> get <资源对象类型> <资源对象名称> -oyaml | grep image: | awk '{print $2}'  # 对镜像地址进行判断，若为空值或latest则应进行调整  ②echo <镜像地址> | awk -F ':' '{print $2}' | 不使用版本latest产生镜像  不使用版本latest设置负载中容器镜像版本 |
| 8.1.2 | 使用docker inspect image，查看NetworkSetting中的Ports，查看暴露的端口是否是必须的服务端口  # 逐项检查输出的端口是否为必须端口  ①docker inspect <镜像地址> | jq '.[0].ContainerConfig.ExposedPorts'  自动化检查函数  check\_8.1.2(){  # $1=<镜像地址>  # 通用变量  chk\_ok="[ok]"  chk\_er="[err]"  chk\_repair="[repair]"  chk\_itm="检查项[8.1.2] => "  # 检查逻辑  fp=$(docker inspect $1 | jq '.[0].ContainerConfig.ExposedPorts' )  echo "$fp"  echo "${chk\_itm}逐项检查输出的端口是否为必须端口;若不是则修改Dockerfile重新生成镜像"  } | 修改Dockerfile重新生成镜像 |
| 8.1.3 | 查看Dockerfile是否存储了敏感信息如口令、密钥 | 去掉Dockerfile的口令、密钥后重新生成镜像 |
| 8.1.4 | 查看Dockerfile中From指令，检查基础镜像是否是可信的镜像  宜使用官方的、可信的镜像作为基础镜像  # 查看Dockerfile中的基础镜像是否可信  ① grep "FROM" <Dockerfile文件> | grep -v '#' | awk '{ print $2 }' | head -n 1  自动化检查函数  check\_8.1.4(){  # $1=<Dockerfile文件路径>  # 通用变量  chk\_ok="[ok]"  chk\_er="[err]"  chk\_repair="[repair]"  chk\_itm="检查项[8.1.4] => "  # 检查逻辑  fp=$(grep "FROM" $1 | grep -v '#' | awk '{ print $2 }' | head -n 1)  echo "${chk\_itm}此Dockerfile中基础镜像是$fp,请检查是否是可信的镜像宜使用官方的、可信的镜像作为基础镜像;如果不是则修改Dockerfile，在From指令中使用可信的镜像"  } | 修改Dockerfile，在From指令中使用可信的镜像 |
| 8.1.5 | # 查找是否有文件设置了setuid或setgid权限，该命令会返回目标文件列表，根据列表进行进一步判断  ①docker run --rm <镜像> bash -c 'find / -perm /6000 -type f -exec ls -ld {} +'  自动化检查函数  check\_8.1.5(){  # $1=<镜像>  # 通用变量  chk\_ok="[ok]"  chk\_er="[err]"  chk\_repair="[repair]"  chk\_itm="检查项[8.1.5] => "  # 检查逻辑  fp=$(docker run --rm $1 bash -c 'find / -perm /6000 -type f -exec ls -ld {} +')  echo "$fp"  echo "${chk\_itm} 根据列表进行进一步判断;使用chmod a-s 对这些文件去除setuid、setgid权限"  } | 使用chmod a-s 对这些文件去除setuid、setgid权限 |
| 8.1.6 | 通过以下命令查看Dockerfile构建文件中是否存在ADD命令  # 查看Dockerfile文件中是否包含ADD指令  ①grep -iE "^\s\*ADD\s+.\*" <Dockerfile文件>  # 判断返回值若返回0则表示存在ADD指令，需调整，否则不存在  ②echo $?  自动化检查函数  check\_8.1.6(){  # $1=<Dockerfile文件路径>  # 通用变量  chk\_ok="[ok]"  chk\_er="[err]"  chk\_repair="[repair]"  chk\_itm="检查项[8.1.6] => "  # 检查逻辑  fp=$(grep -iE "^\s\*ADD\s+.\*" $1 )  if [[ $? == 0 ]];then  echo "${chk\_er}${chk\_itm}$1构建文件中存在ADD指令,不符合规范,请参考如下命令进行修复"  echo "${chk\_repair}${chk\_itm} 根据需要将Dockerfile中的ADD指令修改为COPY指令"  else  echo "${chk\_ok}${chk\_itm}构建文件中不存在ADD指令,符合规范,检查通过"  fi  } | 根据需要将Dockerfile中的ADD指令修改为COPY指令 |
| 8.1.7 | # 查看Dockerfile文件中是否包含弃用指令  ①grep -iE "^\s\*MAINTAINER\s+.\*" <Dockerfile文件>  # 判断返回值若返回0则表示存在弃用指令，需调整  ②echo $?  自动化检查函数  check\_8.1.7(){  # $1=<Dockerfile文件路径>  # 通用变量  chk\_ok="[ok]"  chk\_er="[err]"  chk\_repair="[repair]"  chk\_itm="检查项[8.1.7] => "  # 检查逻辑  fp=$(grep -iE "^\s\*MAINTAINER\s+.\*" $1 )  if [[ $? == 0 ]];then  echo "${chk\_er}${chk\_itm}$1构建文件中包含弃用指令,不符合规范,请参考如下命令进行修复"  echo "${chk\_repair}${chk\_itm}MAINTAINER指令已经被弃用，使用 LABEL命令替换MAINTAINER来添加元数据，如LABEL maintainer=\"John Doe <john.doe@example.com>\""  else  echo "${chk\_ok}${chk\_itm}构建文件中不包含弃用指令,符合规范,检查通过"  fi  } | MAINTAINER指令已经被弃用，使用 LABEL命令替换MAINTAINER来添加元数据，如LABEL maintainer="John Doe <john.doe@example.com>" |
| 8.1.8 | # 查看Dockerfile文件中是否包含update相关指令  ①grep -iE "^ \* +(apt-get|yum|apk) +update" <Dockerfile文件>  # 判断返回值若返回0则表示存在弃用指令，需调整  ②echo $?  自动化检查函数  check\_8.1.8(){  # $1=<Dockerfile文件路径>  # 通用变量  chk\_ok="[ok]"  chk\_er="[err]"  chk\_repair="[repair]"  chk\_itm="检查项[8.1.8] => "  # 检查逻辑  fp=$(grep -iE "^ \* +(apt-get|yum|apk) +update" $1 )  if [[ $? == 0 ]];then  echo "${chk\_er}${chk\_itm}$1构建文件中包含弃用指令,不符合规范,请参考如下命令进行修复"  echo "${chk\_repair}${chk\_itm}不同的镜像仓库开启自动化安全扫描的配置不一样，请查阅用户手册为镜像开启自动扫描"  else  echo "${chk\_ok}${chk\_itm}构建文件中不包含弃用指令,符合规范,检查通过"  fi  } | 不同的镜像仓库开启自动化安全扫描的配置不一样，请查阅用户手册为镜像开启自动扫描 |
| 8.1.9 | 通过以下命令查看Dockerfile构建文件中USER值是否为root或0  grep -iE "^\s\*USER\s+.\*" Dockerfile  自动化检查函数  check\_8.1.9(){  # $1=<Dockerfile文件路径>  # 通用变量  chk\_ok="[ok]"  chk\_er="[err]"  chk\_repair="[repair]"  chk\_itm="检查项[8.1.9] => "  # 检查逻辑  fp=$(grep -iE "^\s\*USER\s+.\*" $1 )  echo "$fp"  echo "${chk\_itm}请判断USER值是否为root或0,若是root或0则需要指定为普通用户"  } | 容器构建USER需指定为普通用户 |
| 8.1.10 | 通过以下命令查看Dockerfile构建文件中RUN指令是否存在通过sudo执行的情况  grep -iE "^\s\*RUN\s+.\*sudo\s+.\*" Dockerfile  自动化检查函数  check\_8.1.10(){  # $1=<Dockerfile文件路径>  # 通用变量  chk\_ok="[ok]"  chk\_er="[err]"  chk\_repair="[repair]"  chk\_itm="检查项[8.1.10] => "  # 检查逻辑  fp=$(grep -iE "^\s\*RUN\s+.\*sudo\s+.\*" $1 )  if [[ $? == 0 ]];then  echo "${chk\_er}${chk\_itm}$1构建文件中RUN指令存在通过sudo执行的情况,不符合规范,请参考如下命令进行修复"  echo "${chk\_repair}${chk\_itm}在Dockerfile中使用sudo通常不被推荐，主要是为了避免与权限提升相关的潜在安全风险;在 Dockerfile中使用sudo将会授予执行上下文额外的权限"  else  echo "${chk\_ok}${chk\_itm}构建文件中RUN指令不存在通过sudo执行的情况,符合规范,检查通过"  fi  } | 在Dockerfile中使用sudo通常不被推荐，主要是为了避免与权限提升相关的潜在安全风险。在 Dockerfile中使用sudo将会授予执行上下文额外的权限。 |
| 8.1.11 | get\_work\_dir[output] {  workdir := dockerfile.workdir[\_]  arg := workdir.value[0]  not regex.match("^[\"']?(/[A-z0-9-\_+]\*)|([A-z0-9-\_+]:\\\\.\*)|(\\$[{}A-z0-9-\_+].\*)", arg)  output := {  "cmd": workdir,  "arg": arg,  }  }  deny[res] {  output := get\_work\_dir[\_]  msg := sprintf("WORKDIR path '%s' should be absolute", [output.arg])  msg\_cn := sprintf("WORKDIR '%s' 应使用绝对路径", [output.arg])  res := {  "message": msg,  "message\_cn": msg\_cn,  "cause": output.cmd,  }  } | 每个 stage 中尽量合并多个RUN指令成一个，并重新构建镜像 |
| 8.1.13 | deny[res] {  output := get\_pip[\_]  msg := sprintf("pin versions in pip: '%s'", [output.args])  msg\_cn := sprintf("pip安装时需要指定软件的版本: '%s'", [output.args])  res := {  "cause": output.cmd,  "message": msg,  "message\_cn": msg\_cn,  }  }  get\_pip[output] {  run := dockerfile.run[\_]  args := concat(" ", run.value)  # regex.match(`pip\s+?install\s+\w+$`, args)  # not regex.match(`pip\s+?install\s+\w+\==\d`, args)  # regex.match(`pip\s+?install\s+http.+?(\w+$)`, args)  # not regex.match(`pip\s+?install\s+\w+.+?\@\d`, args)  unversioned := [x |  x := trim\_space(regex.split(`\s\*&&\s\*`, args)[\_])  is\_pip\_install(x)  not is\_install\_versioned(x)  ]  count(unversioned) > 0  output := {  "cmd": run,  "args": args,  }  }  args\_value\_pattern := "(?:\".\*?\"|'.\*?'|\\S)+"  install\_pattern := `pip((\d+\.)?(\d+\.)?(\\*|\d+))?\s+(-(-)?[a-zA-Z=]+\s\*)\*install`  arg\_index\_url\_pattern := sprintf(`(?:%s\s+|%s(?:=|\s+))%s`, ["-i", "--index-url", args\_value\_pattern])  arg\_extra\_index\_url\_pattern := sprintf(`(?:%s(?:=|\s+))%s`, ["--extra-index-url", args\_value\_pattern])  arg\_find\_link\_pattern := sprintf(`(?:%s\s+|%s(?:=|\s+))%s`, ["-f", "--find-links", args\_value\_pattern])  semver\_pattern := `(?:[v]?(\d+\.)?(\d+\.)?(\\*|\d+))`  pip\_version\_pattern := sprintf(`(?:\w+==%s|\S+@%s)`, [semver\_pattern, semver\_pattern])  is\_pip\_install(cmd) {  regex.match(install\_pattern, cmd)  }  is\_install\_versioned(cmd) {  s := clean(cmd)  pkgs := [p |  found := regex.find\_n("(?:\".\*?\"|'.\*?'|\\S)+", s, -1)  p := found[\_]  # ignore flags like: pip install sax --no-index  p = regex.replace(p, `^(-(-)?[a-zA-Z]+\s\*)\*`, "")  p = trim\_space(p)  p != ""  ]  unversioned := [p |  p := pkgs[\_]  not regex.match(pip\_version\_pattern, p)  ]  count(unversioned) == 0  }  is\_install\_versioned(cmd) {  # pip install [options] -r <requirements file> [package-index-options] ...  re := sprintf(`(?:%s\s+|%s(?:\s+|=))%s`, ["-r", "--requirement", args\_value\_pattern])  regex.match(re, cmd)  }  clean(cmd) = res {  re := sprintf("^.\*%s", [install\_pattern])  s := regex.replace(cmd, re, "")  s1 := regex.replace(s, arg\_index\_url\_pattern, "")  s2 := regex.replace(s1, arg\_extra\_index\_url\_pattern, "")  s3 := regex.replace(s2, arg\_find\_link\_pattern, "")  res := s3  } | 使用 apt-get install 安装软件时建议指定安装包的版本号，并重新构建镜像 |
| 8.1.14 | get\_apt\_get[output] {  run := dockerfile.run[\_]  args := concat(" ", run.value)  regex.match(`apt-get\s+install`, args)  not regex.match(`apt-get\s+clean`, args)  not regex.match(`rm\s+-rf\s+\/var\/lib\/apt\/lists\/\\*`, args)  output := {  "args": args,  "cmd": run,  }  }  deny[res] {  output := get\_apt\_get[\_]  msg := sprintf("You should delete the apt-get lists after installing something: '%s'", [output.args])  msg\_cn := sprintf("apt-get install 安装完成后应使用 'apt-get clean' 清理缓存: '%s'", [output.args])  res := {  "cause": output.cmd,  "message": msg,  "message\_cn": msg\_cn,  }  } | 在执行 apt-get 命令后加上 'apt-get clean' 清理本地缓存，并重新构建镜像 |
| 8.1.15 | deny[res] {  healthchecks := dockerfile.stage\_healthcheck[stage]  cnt := count(healthchecks)  cnt > 1  msg := sprintf("There are %d duplicate HEALTHCHECK instructions in the stage '%s'", [cnt, stage.name])  msg\_cn := sprintf("在stage '%s' 中定义了 %d 个的HEALTHCHECK指令", [stage.name, cnt])  res := {  "cause": healthchecks[1],  "message": msg,  "message\_cn": msg\_cn,  }  } | Dockerfile中仅保留一条HEALTHCHECK指令，并重新构建镜像 |
| 8.1.16 | deny[res] {  output := get\_alias\_from\_copy[\_]  msg := sprintf("'COPY --from' should not mention current alias '%s' since it is impossible to copy from itself", [output.args])  msg\_cn := sprintf("'COPY --from' 不应引用当前 stage 别名 '%s'", [output.args])  res := {  "cause": output.cmd,  "message": msg,  "message\_cn": msg\_cn,  }  }  get\_alias\_from\_copy[output] {  copies := dockerfile.stage\_copies[stage]  copy := copies[\_]  flag := copy.flags[\_]  contains(flag, "--from=")  parts := split(flag, "=")  is\_alias\_current\_from\_alias(stage.name, parts[1])  args := parts[1]  output := {  "args": args,  "cmd": copy,  }  }  is\_alias\_current\_from\_alias(current\_name, current\_alias) = allow {  current\_name\_lower := lower(current\_name)  current\_alias\_lower := lower(current\_alias)  #expecting stage name as "myimage:tag as dep"  [\_, alias] := regex.split(`\s+as\s+`, current\_name\_lower)  alias == current\_alias  allow = true  } | 更改'--from'，使其不引用自身，并重新构建镜像 |
| 8.1.17 | deny[res] {  cmds := dockerfile.stage\_cmd[stage]  cnt := count(cmds)  cnt > 1  msg := sprintf("There are %d duplicate CMD instructions in stage '%s'", [cnt, stage.name])  msg\_cn := sprintf("在 stage '%s' 中存在 %d 个 CMD 指令", [stage.name, cnt])  res := {  "cause": cmds[1],  "message": msg,  "message\_cn": msg\_cn,  }  } | Dockerfile 中应当只保留一个 CMD 指令 |
| 8.1.18 | deny[res] {  runs := dockerfile.stage\_run[stage]  cnt := count(runs)  cnt > 1  msg := sprintf("There are %d RUN instructions in stage '%s'", [cnt, stage.name])  msg\_cn := sprintf("在 stage '%s' 中存在 %d 个 RUN 指令", [stage.name, cnt])  res := {  "cause": runs[1],  "message": msg,  "message\_cn": msg\_cn,  }  } | 每个 stage 中尽量合并多个RUN指令成一个，减少镜像层数 |

### C.8.2 镜像仓库安全配置

表C.7

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 8.2.1 8.2.2 | 不同的镜像仓库配置扫描周期的方式不一样，应在镜像仓库或者镜像扫描服务中确认是否配置了扫描周期 | 不同的镜像仓库开启自动化安全扫描的配置不一样，请查阅用户手册为镜像设置合理的扫描周期 |
| 8.2.3 | 从以下几个方面检查镜像仓库访问权限，确保镜像仓库访问权限最小化：  1、镜像仓库的可见性：控制哪些用户或者组织可以查看或者搜索到镜像仓库。公开的镜像仓库可以被任何人查看和拉取，但是只有拥有者或者协作者可以推送。私有的镜像仓库只能被拥有者或者指定的用户或者组织查看、拉取和推送。内部的镜像仓库只能被同一组织的成员查看、拉取和推送。为了最小化访问权限，建议将镜像仓库设置为私有或者内部，除非有特殊的需求。  2、镜像仓库的访问控制：控制哪些用户或者组织可以对镜像仓库进行拉取或者推送操作。为了最小化访问权限，建议将推送权限只授予必要的用户或者组织，例如开发人员或者持续集成服务。同时，建议使用不同的账号或者凭证来区分不同的操作场景，例如开发、测试、生产等。  3、镜像仓库的安全扫描：控制哪些类型的镜像可以被拉取或者推送到镜像仓库，以及如何检测和修复镜像中的漏洞。为了最小化访问权限，建议启用安全扫描功能，并且设置合理的策略来限制哪些类型的镜像可以被拉取或者推送，例如官方镜像、认证发布商镜像、组织自己的镜像等 | 根据需要设置镜像仓库的可见性、镜像仓库的访问控制权限、配置合理的镜像仓库扫描策略 |
| 8.2.4 | 检查集群是否开启了策略治理能力：   1. 检查K8S集群是否启用了准入控制功能。 2. 查看集群是否安装了策略治理组件如Gatekeeper 或Kyverno 3. 检查集群是否配置了可信镜像仓库白名单策略，仅允许白名单镜像在k8s集群部署。 4. 验证可信镜像仓库白名单策略是否符合预期 | 根据具体的需求开启集群策略治理能力，并配置可信镜像仓库白名单策略，确保未知镜像无法部署到集群 |
| 8.2.5 | 通过以下方面来检查镜像仓库自身风险扫描能力：  1、通过镜像扫描工具对镜像仓库自身的漏洞进行扫描，检查是否存在漏洞。  2、通过镜像扫描工具对镜像仓库自身的弱口令进行扫描，检查是否存在弱口令。  3、通过镜像扫描工具对镜像仓库自身的配置进行扫描，检查是否在弱配置项。  4、通过镜像扫描工具对镜像仓库自身的恶意文件进行扫描，检查是否存在植入的恶意软件 | 解决扫描发现的问题 |
| 8.2.6 | 通过以下方面来对镜像仓库进行细粒度的访问控制。  1、使用角色或用户组或项目空间来管理不同的访问者，根据他们的身份和职责分配合适的权限  2、使用 IP 白名单或黑名单来控制哪些 IP 地址可以访问仓库，以防止恶意攻击或泄露敏感信息  3、使用 HTTPS 协议来加密仓库的通信，以保护数据的完整性和隐私性  4、使用 token 或证书来验证访问者的身份，以防止伪造或盗用 | 根据具体的需求对镜像仓库进行细粒度的访问控制，配置合理的策略，如配置IP、角色、项目空间等策略，确保权限最小化 |
| 8.2.7 | 通过以下命令查看镜像仓库是否具备镜像文件加密传输的能力。  curl -v https://<镜像仓库地址> 2>&1 | grep -iE "ssl|tls"  如果命令输出显示了有效的证书和协议信息，说明该镜像仓库具备文件机密传输的能力，否则不具备 | 根据具体的镜像仓库设置镜像仓库支持HTTPS以加密传输镜像文件 |
| 8.2.8 | 通过以下方面来对镜像仓库进行文件加密存储配置  1、开启仓库存储文件加密配置  2、配置仓库文件所在卷加密 | 根据具体的需求对镜像仓库进行加密存储配置。 |

### C.8.3 镜像部署配置

表C.8

|  |  |  |
| --- | --- | --- |
| **要求** | **检查方法** | **修复方法** |
| 8.3.1 | 确认是否为容器镜像开启了签名验签机制  通过如下命令查看镜像签名信息  ①docker trust inspect <镜像名称> | 实现容器镜像签名验签的方式很多，典型的开源实现包括Kritis, cosign等。如下以 cosign 为例：  签名  cosign sign -key cosign.key dustise/sleep:v0.9.6  验签：  cosign verify -key cosign.pub repository/image:v0.9.6 |
| 8.3.2 | 使用商用软件或开源软件(如Trivy, Clair等)扫描扫描容器镜像，查看输出结果是否有漏洞  ，不同的工具有不同的使用命令，如Trivy，使用如下命令进行扫描  trivy <镜像名称> | 升级镜像替换有漏洞的组件，生成新的镜像 |
| 8.3.3 | 宜使用Kubernetes admission controller来控制镜像准入  kube-apiserver -h | grep enable-admission-plugins  或使用命令：  ps -ef |grep kube-apiserver |grep enable-admission-plugins  查看是否有需要的admission controller设置 | 根据具体的需求开启集群准入配置能力，在安装集群时开启集群准入控制配置 |
| 8.3.4 | 基础镜像中若使用不必要的软件，可能会加大容器的攻击面，同时也违背了最小和精简容器镜像的概念。因此，不要为容器安装或运行任何不必要的软件。 | 精简基础镜像。 |

# 附 录 D

# （资料性附录）

**部分要求的场景限制**

以下为正文7.1节部分要求的使用场景限制。

## D.1 特权容器使用场景

业务确实有使用特权容器的场景，在满足安全最小化原则，并充分做好安全风险评估前提下，允许以特权容器运行，典型场景包括：

• 业务需要容器内修改时间同步修改所在宿主机的时间；容器存储接口（CSI），需要将宿主机存储实例挂载到容器内部，需要获取宿主机文件访问权限，获取与主机一致的挂载信息；

• OS升级，需要将虚拟机根目录挂载到容器的根目录上，因此容器内下载系统包、解压、安装的操作需要root权限；

• 组件需要访问宿主机相关资源来采集指标或驱动设置，包括宿主机网络信息，进程信息，文件信息（如/sys/、/proc/等目录下的信息），因此所在的容器需要能够访问主机网络命名空间、主机进程命名空间、主机系统敏感目录；

• 网络负载均衡器需要将外部网络接入到容器内部，完成网络接入和转发功能。

## D.2 共享主机使用场景

如果主机的PID命名空间与容器共享，则基本上允许容器中的进程查看主机系统上的所有进程。 这打破了主机和容器之间的进程级隔离。

有权访问容器的人最终可以知道在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。 因此，不要与容器共享主机的进程命名空间。

应至少定义一个准入控制策略，禁止容器共享主机PID命名空间。

1. 业务确实有使用容器共享主机的进程命名空间的场景，在满足安全最小化原则，并充分做好安全风险评估前提下，允许使用容器共享主机的进程命名空间，典型场景包括：

• 组件需要访问宿主机相关资源来采集指标，包括宿主机网络信息，进程信息，文件信息（如/sys/、/proc/等目录下的信息），因此所在的容器需要能够访问主机网络命名空间、主机进程命名空间、主机系统敏感目录；

1. 业务确实有使用容器共享主机的IPC命名空间的场景，在满足安全最小化原则，并充分做好安全风险评估前提下，允许容器共享主机的IPC命名空间，典型场景包括：

• 组件需要访问宿主机相关资源来采集指标，包括宿主机网络信息，进程信息，文件信息（如/sys/、/proc/等目录下的信息），因此所在的容器需要能够访问主机的IPC命名空间、主机网络命名空间、主机进程命名空间、主机系统敏感目录；

1. 业务确实有使用容器共享主机网络命名空间的场景，在满足安全最小化原则，并充分做好安全风险评估前提下，允许使用容器共享主机网络命名空间，典型场景包括：

组件需要访问宿主机相关资源来采集指标或设置网卡驱动，包括宿主机网络信息，进程信息，文件信息（如/sys/、/proc/等目录下的信息），因此所在的容器需要能够访问主机网络命名空间、主机进程命名空间、主机系统敏感目录；

④ 业务确实有需要在容器内提权的场景，在满足权限最小化原则，并充分做好安全风险评估前提下，允许开启allowPrivilegeEscalation，典型场景包括：

-因root权限最小化，导致稳态以普通用户权限运行，且需要瞬时提权至root权限的业务；

⑤ 业务确实有使用NET\_RAW的场景，在满足安全最小化原则，并充分做好安全风险评估前提下，允许容器使用NET\_RAW能力，典型场景包括：

• 网络负载均衡器需要将外部网络接入到容器内部，完成网络接入和转发功能；

• 容器需要实时修改宿主机iptables规则来保持VIP的时效性；

参　考　文　献

[1] CIS Kubernetes Benchmarks

[2] kubernetes官方服务文档 [DB/OL].[2023-07-27] https://kubernetes.io/docs/home/

[3] docker官方服务文档[DB/OL]. [2023-07-27].

https://docs.docker.com/get-started/overview/

[4] RFC 8446 [DB/OL]. [2023-07-27]. https://datatracker.ietf.org/doc/html/rfc8446

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_