28 | 安全提升：GitOps 在哪些环节需要关注安全问题？

2023-02-10 王炜 来自北京

《云原生架构与GitOps实战》 课程介绍

讲述：王炜

时长 11:32 大小 10.54M

你好，我是王炜。

提到安全，我们可能最常想到的就是业务系统的安全，例如数据库安全、业务逻辑安全、运行环境安全和网络安全等等。

当我们将业务系统迁移到云原生架构下时，这些安全问题仍然是值得关注的。此外，当我们采用 GitOps 时，由于工作流会接触到更多的基础设施，所以必定也会面临更多安全挑战。那么，在这些安全挑战中，我们应该注意哪些方面呢？如何尽可能规避这些安全问题？

这节课，我们就来看看在 GitOps 中有哪些环节是需要格外关注安全问题的，我也会提供相应的建议。根据 GitOps 工作流所涉及到的上下游，我们需要关注以下几个方面：

1. Docker 镜像

2. 业务依赖
3. CI 流水线
4. Docker 镜像仓库
5. Git 仓库
6. Kubernetes 集群
7. 云厂商服务

接下来，我们就来看看为什么这几个方面容易出现安全问题。

Docker 镜像

Docker 镜像是产生安全问题的一个重要的源头。通常，使用一些未知的基础镜像是引入安全问题的重要因素。

未知的基础镜像可能会带来几个方面的问题。首先，镜像内包含的系统工具可能因为版本过低存在一些安全漏洞，这可能会导致镜像在被部署后出现安全攻击。其次，在编写 Dockerfile 时，我们一般很少关注运行用户。在默认的情况下，如果使用 root 权限的用户，就会进一步扩大攻击面，如果应用本身比较脆弱，这时候就很容易受到提权攻击。

此外，当我们拉取一个 Docker 镜像时，默认情况下 Docker 不会验证它的来源和真实性。攻击者可能会制作一个具备相同功能但包含恶意程序的镜像。

最后，当我们自己在构建 Docker 镜像时，需要注意不要将密钥文件复制到镜像内，这可能会导致凭据泄露。

基于上面提到的可能出现的安全问题，我建议你遵循下面几个原则。

1. 使用可信赖的 Docker 基础镜像，通常情况下，它们在 DockerHub 官网具有明确的标识。
2. 尽量使用较小的基础镜像，最大程度减小安全攻击面。
3. 在编写 Dockerfile 的时候，使用非 root 用户来启动业务程序，你也可以借助**开源工具 hadolint 来检查 Dockerfile 是否符合最佳实践。**

4. 使用 Docker Content Trust 来验证 Docker 镜像的签名，你可以通过添加环境变量 DOCKER_CONTENT_TRUST=1 来启用它，同时你也应该对你构建的镜像进行签名。
5. **使用 docker scan 命令来扫描构建的业务镜像，**它内置非常多安全规则，可以最大程度避免出现安全问题。

业务依赖

再来看业务依赖。业务依赖实际上是软件供应链中的一个环节。在编写业务系统时，我们通常会引入第三方的依赖来降低开发工作量。这时候，我们就很容易引入可能包含安全漏洞的第三方业务依赖。比如影响面非常大的 Log4j2 安全漏洞。

要知道业务包含哪些第三方组件，你可以使用 docker sbom 命令来分析第三方依赖。docker sbom 命令采用开源的 syft 项目进行分析，它支持常见的编程语言和包安装工具，分析结果可以通过标准的 spdx-json 格式导出，它是一个标准格式，可以获得与软件包有关的部件和元数据。

以我们在之前课程中构建的 lyzhang1999/frontend:latest 镜像为例，你可以使用下面的命令来获取镜像的 BOM 清单。

```
复制代码
1 docker sbom lyzhang1999/frontend:latest --format spdx-json --output sbom.json
```

SBOM 清单只是用来记录软件包含哪些依赖组件的，你还需要通过 https://osv.dev/ 开源的分布式漏洞数据库来查看 SBOM 内已知的组件漏洞，或者借助 Google 开源的 osv-scanner 项目来批量查询。

将它们集成到 CI 阶段能够有效地及时发现业务依赖漏洞，**一旦发现依赖组件存在安全漏洞，则可以立即停止将构建的镜像推送到镜像仓库，防止其进一步被部署到生产环境。**

CI 流水线

在没有将发布流程迁移到 GitOps 工作流之前，很多团队会在 CI 流水线构建镜像，并执行部署操作。常见的做法是直接在 CI 里执行 kubectl apply 命令来更新 Kubernetes 的镜像版本。

在这种情况下，由于 CI 流水线能够直接访问集群，所以当 CI 流水线被劫持或者破坏时，很容易将恶意的程序部署到集群中。

但是，随着我们将部署行为迁移到 GitOps 工作流，CI 流水线不再需要直接访问集群，安全风险会有所降低。在 GitOps 工作流中，CI 流水线主要用来构建容器镜像，所以它可能会将包含恶意程序的镜像推送到镜像仓库中，前面提到的在 CI 流水线中增加安全扫描过程正是基于此来考虑的。

此外，当我们使用 GitHub 或 GitLab 托管的 CI 过程中，流水线往往是通过分支或者 Pull Requeust 来触发的，此时需要防范攻击者在提交新的分支或者 Pull Request 时修改 CI 流水线定义文件，并利用它来实施一些攻击。

Docker 镜像仓库

Docker 镜像仓库的安全主要有两方面需要重点关注。

1. 使用私有镜像仓库而不是公开的镜像仓库。
2. 使用不可变的镜像版本。

对于第一点，通常在项目初期，我们会把业务镜像上传到公共镜像仓库例如 DockerHub 中。由于任何人都能够将他的镜像上传到公共的镜像仓库中，所以我们不能把公共的镜像仓库当成集群的可信的镜像源。相反地，在生产环境下，我们应该**完全禁止**从这种公开的镜像仓库中拉取镜像的行为。

比较好的实践是，在组织内部搭建一套可信的镜像仓库。例如我们在第 19 讲提到使用 Harbor 来搭建私有镜像仓库，并将它作为组织内部的唯一可信镜像源，同时定期进行**漏洞扫描**。以此来最大程度确保镜像仓库的可信和安全性。

而第二点则是新手非常容易犯的错误。在早期构建镜像时，为了方便起见，很多团队都会采用固定的版本命名，例如将每次构建的镜像版本都命名为 latest，这是**非常不正确的做法**。

首先，latest 不够语义化，他不能一目了然地看出当前镜像版本对应的代码版本。更好的实践是使用类似 v1.0 或者 Git commit ID 作为镜像版本号。其次，镜像版本不应该能够被覆盖，而是应当遵循“**不可变版本**”的原则对镜像版本号进行锁定。这样，即便是攻击者获得了镜像仓库

的写入权限，也无法用包含恶意程序的镜像覆盖已存在的镜像版本，自然也就无法轻易将恶意的镜像部署到生产环境中了。

Git 仓库

在 GitOps 流程中，Git 仓库是集群资源的唯一可信源，它可能包含 Kubernetes 对象、配置文件和密钥。因此，除了合理配置 Git 仓库的权限以外，我还建议你通过私有化部署的 Git 仓库配合内部网络策略来降低被攻击的可能性。

退一步说，如果你的 Git 仓库因为凭据泄露或其他原因导致攻击者具备访问和写入权限，在 GitOps 工作流中，攻击者必须将它的恶意代码提交到 Git 仓库中才能触发部署，所以这很容易暴露攻击者本身，也非常容易通过审计的手段及时发现。

一个比较好的防范措施是：**为 Git 仓库配置 Main 分支的保护策略，并将 Main 分支作为 GitOps 部署源**。任何人将代码提交到 Main 分支时，都需要经过成员的人工 Review 才能合并，这样就可以有效防范恶意的镜像或者代码部署到生产环境中。

此外，你还需要重点关注在 Git 仓库中存储的明文密钥，例如镜像拉取凭据、数据库连接信息和外部服务凭据。**你应该通过加密的方式存储它们**，这样，即便仓库的读取权限遭到泄露，攻击者也无法取得明文凭据，这就进一步保护了系统安全。

至于如何对 Git 仓库存储的密钥进行加密，我会在下节课为你详细介绍。

Kubernetes 集群

Kubernetes 集群是最重要的基础设施，同样也是攻击者的重点攻击目标。你可以在以下几个方面来检查 Kubernetes 集群处于安全状态。

1. 仅限在内网访问 Kubernetes。
2. 通过 Kubernetes RBAC 提供集群访问凭据，而不是使用管理员的 Kubeconfig。
3. 隔离 Kubernetes 节点而不是将它们暴露在公网当中。
4. 开启了 Kubernetes 的审计日志。

在 GitOps 流程中，我们实际上为开发人员提供了一种全新的集群访问方式：通过 Git 访问而不是直接连接集群。而在传统的发布流程中，我们可能需要对开发者或运维暴露 Kubeconfig

以便他们能够直接操作集群，这是一种比较危险的做法，也容易导致凭据泄露。

即便是在一些需要直接使用 Kubeconfig 的特殊的场景下，你也应该尽可能通过网络策略来保证集群的安全，比如关闭 Kubernetes 集群和节点的公网访问，使集群只能通过内网进行访问。其次，你应该通过 RBAC 来提供集群凭据，而不是直接对外分发管理员的凭据，并根据“最小权限”的原则对资源和操作赋予权限。

最后，开启 Kubernetes 的审计日志将有助于我们及时发现与 API Server 的异常连接。

云厂商服务

这里提到的云厂商服务其实指的是业务依赖的云厂商资源，例如消息队列、数据库和对象存储等。

这些云厂商的服务虽然和 GitOps 没有直接的关系，但由于它们通常会和 Kubernetes 集群产生直接的交互，所以也需要额外关注它们的安全问题。

例如，当业务系统连接云厂商的数据库服务时，你应该通过内网进行连接而不是通过公网连接，此外，Kubernetes 集群和数据库还应该在同一个 VPC（虚拟网络）下，以便能够在内网环境相互通信。此外，你还可以借助 IP 白名单策略来禁止来自集群外的连接。

另外，当我们使用云厂商集群时，由于它们和云厂商的其他服务深度绑定，所以我们可以很轻松地通过创建 Kubernetes 对象来创建云厂商资源。例如，声明一个 Loadbalancer 类型的 Service 就可以创建负载均衡器，这可能会导致服务被不小心暴露在了外网。还有，声明 PVC 就可以创建云厂商的持久化存储，随着数量的增加，这会额外带来更高的成本。所以，在创建资源时，需要额外注意这些问题。

总结

这节课，我为你总结了在 GitOps 工作流中需要关注的安全问题。根据 GitOps 所涉及到的上下游，我在 7 个方面为你阐述了可能存在的安全隐患以及如何最大程度规避它们。

实际上，在安全领域，通常我们认为世界上**没有绝对的安全，只有成本、便捷性和安全的综合考量**。所以，在我总结的这些安全建议里，你应该根据业务的实际情况进行动态决策，同时根据业务形态来决定在安全上投入的成本，避免过早陷入到大量的安全细节而忽略了业务本身。

这节课虽然在细节的介绍上比较深入，但总结来说，要想在项目早期做好安全策略也没有想象中这么复杂，做好下面几点就可以达到比较高的 GitOps 安全等级。

1. 使用自托管镜像仓库，并开启不可变版本和安全漏洞扫描。
2. 使用云厂商托管的 Kubernetes 集群，并关闭公网访问或仅限白名单访问。
3. 加密 Git 仓库中的密钥，取代明文存储的密钥。

由于 Kubernetes 并没有为我们提供完善的密钥存储机制，在众多潜在的安全问题中，我认为加密 Git 仓库中的机密信息是最有必要的，这也是几乎所有团队都会在早期面临的安全隐患。

所以，在下节课，我会深入介绍这部分内容，带你进一步理解如何加密在 Git 仓库存放的机密信息，例如镜像拉取凭据和数据库连接信息，为 GitOps 工作流提供更强大的安全保障。

思考题

最后，给你留一道思考题吧。

著名的 Log4j 漏洞导致非常多业务系统出现了安全事件。现在，结合这节课的内容，请你聊一聊如何有效地防范这类型的安全漏洞。

欢迎你给我留言交流讨论，你也可以把这节课分享给更多的朋友一起阅读。我们下节课见。

上一篇　27 | 开发互不干扰，如何实现自动多环境管理？

下一篇　29 | 安全提升：如何解决 GitOps 的秘钥存储问题？

精选留言 (2)

写留言

俊采
2023-03-06 来自陕西

方便解释下容器镜像不可变版本么，谢谢

> 作者回复: 镜像不可变版本的意思是一旦镜像版本被创建之后，在实践上不应该能覆盖它，比如为每一个 commit id 构建一个镜像。
> 如果违背了这个原则，就无法知道该镜像版本对应的软件特性，带来管理上的混乱。

2

郑海成
2023-02-15 来自北京

第一道防线：鼓励在开发者进行本地代码开发时，使用ide集成的docker插件进行scan和sbom；第二道防线：在CI流水线中集成上面两种step，设定合理的阈值，超过阈值则不进行后续steps，并结合监控系统告警；第三道防线：管理层面要求开发者去优先处理这些高危漏洞

> 作者回复: 很好的总结，最好结合 CI 过程把安全问题扼杀在代码发布到生产前。