

15 | Docker安全：在虚拟的环境中，就不用考虑安全了吗？

2020-01-15 何为舟

安全攻防技能30讲

[进入课程 >](#)



讲述：何为舟

时长 15:58 大小 12.80M



你好，我是何为舟。

在 [13 讲](#) 中，我们讲了 Linux 系统安全。但是，当你在和同事讨论 Linux 系统安全的时候，同事表示，公司的服务都是通过 Docker 来进行容器化部署的。开发在操作中，并不会接触实际的 Linux 服务器，所以不会去关注 Linux 安全。而且，因为容器是隔离的，就算容器被黑客攻击了，也只是容器内部受到影响，对宿主的 Linux 系统和网络都不会产生太大影响。



事实上，我知道很多人都有这种想法。但是，你在学习了安全专栏之后，可以试着思考一下，开发使用了 Docker 就一定安全吗？真的可以不用考虑安全问题了吗？

以防你对 Docker 还不是很了解，在解决这些问题之前，我先来解释一下这节课会涉及的 3 个概念，帮你扫清概念障碍。

Docker 服务：Docker 所提供的功能以及在宿主机 Linux 中的 Docker 进程。

Docker 镜像：通过 Dockerfile 构建出来的 Docker 镜像。

Docker 容器：实际运行的 Docker 容器，通常来说，一个 Docker 镜像会生成多个 Docker 容器。Docker 容器运行于 Docker 服务之上。

了解了这 3 个关键概念之后，我们今天就从这些概念入手，来谈一谈 Docker 的安全性。

Docker 服务安全

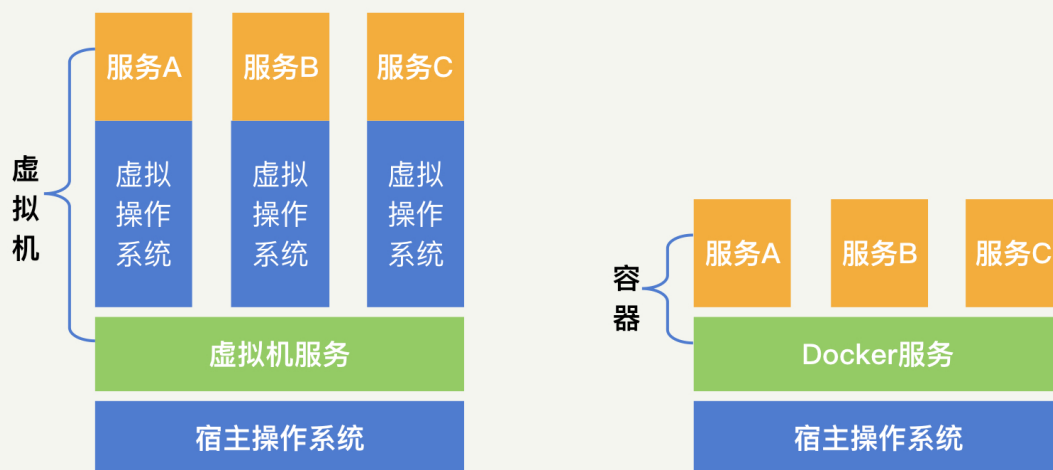
我们首先来看 Docker 服务的安全性。Docker 服务本身需要关注的安全性就是：隔离。如果黑客在控制了容器之后，能够成功对宿主机产生影响，就说明黑客突破了 Docker 服务的隔离保护，也就是我们所说的“Docker 逃逸”。

那么，Docker 服务是如何对容器进行隔离，来防止“Docker 逃逸”的呢？接下来，我就介绍一下这 3 个关键的隔离机制：Namespace 机制、Capabilities 机制和 CGroups 机制。

第 1 个是 **Namespace 机制**。

我们知道，Docker 之所以广泛流行，是因为它提供了一种轻量化的隔离环境，也就是容器。

下面，我们重点解释一下“轻量化”和“隔离”这两个词。首先是轻量化。怎么理解轻量化呢？我们可以对比虚拟机来进行理解。虚拟机是自己创造了一个虚拟内核，让这个虚拟内核去和虚拟机的进程进行沟通，然后虚拟内核再和真实的 Linux 内核进行沟通。而 Docker 提供的容器，简化了这个沟通过程，让 Docker 中的进程直接和 Linux 内核进行沟通。



第二个词是隔离。也就是说，Docker 提供的容器环境是和 Linux 内核隔离的。要实现这种隔离，就需要用到 **Namespace 机制**了。所以，这里我先给你简单解释一下什么是 Namespace 机制。

Namespace 是 Linux 提供的一种标签机制，Linux 内核会对不同 Namespace 之间的进程做隔离，避免不同的进程之间互相产生影响。所以，Docker 服务会为每一个 Docker 容器创建一个单独的 Namespace 空间。这样一来，不同容器之间、容器和系统之间，都是不同的 Namespace，也就实现了隔离。

这种基于 Namespace 的隔离我一般叫它“伪隔离”。因为通过 Namespace 进行的隔离并不彻底。为啥这么说呢？Docker 容器在隔离的环境中，仍然需要使用一些底层的 Linux 进程和设备支持。比如，你在 Docker 容器中仍然需要使用鼠标、键盘等输入输出设备，那么容器就必须挂载 Linux 系统中的 /sys 来获得对应的驱动和配置信息。也就是说，你在 Docker 中看到的 /sys 目录，实际就是 Linux 系统中的 /sys 目录。类似的，还有一些没有被 Namespace 隔离开的目录和模块，包括以下内容：

部分的进程目录 /proc/...

内存映像 /dev/mem

系统设备 /dev/sd*

Linux 内核模块

换一句话说，因为容器和宿主机需要共同使用一些服务（比如容器和宿主机使用的是同一个鼠标），所以上面的这些目录和模块，对于容器和宿主机来说，其实是共享的。从理论上来说，如果你在 Docker 容器中修改了这些目录，那么宿主机当中也会同步相应的修改结果。

第 2 个 **Capabilities** 机制。

我们刚刚说了，Namespace 的伪隔离机制让容器和宿主机共享部分目录。那么，这是不是也意味着，Docker 容器可以通过这些目录来影响宿主机，从而实现“Docker 逃逸”呢？为了避免这种情况，Docker 服务使用了 Capabilities 机制，来限制容器的操作。

Capabilities 提供了更细粒度的授权机制，它定义了主体能够进行的某一类操作。比如，一个 Web 服务需要绑定 80 端口，但 80 端口的绑定是需要 ROOT 权限的。为了防止 ROOT 权限滥用，Docker 会通过 Capabilities，给予这个 Web 服务 `net_bind_service` 这个权限（允许绑定到小于 1024 的端口）。同样地，Docker 对容器的 ROOT 也加了很多默认的限制，比如：

- 拒绝所有的挂载操作；

- 拒绝部分文件的操作，比如修改文件所有者；

- 拒绝内核模块加载。

这里有一点需要你注意，Capabilities 对容器可进行操作的限制程度很难把控。这是因为，过松会导致 Docker 容器影响宿主机系统，让 Docker 隔离失效；过严会让容器和容器内的服务功能受限，无法正常运行。

所以，在默认情况下，Docker 会采用白名单机制（白名单列表你可以在 Docker 源码中查看）进行限制，即只允许 Docker 容器拥有几个默认的能力。那有了白名单限制，即使黑客成功拿到了容器中的 ROOT 权限，能够造成的影响也相对较小。所以我们常说，“Docker 逃逸”是一件不容易的事情。

第 3 个是 **CGroups** 机制。

好了，现在你应该知道 Docker 服务本身是如何防止“Docker 逃逸”的了。作为一个容器，Docker 显然不能过多地占用宿主机资源，不然对宿主机和自身的可用性都会产生影响。那 Docker 是如何实现资源限制的呢？

Docker 服务可以利用 CGroups 机制来实现对容器中内存、CPU 和 IO 等的限制。比如，通过下面的命令，我们就可以限制 Docker 容器只使用 2 个 CPU 和 100MB 的内存来运行了。

复制代码

```
1 docker run -it --cpus=2 --memory="100m" ubuntu:latest /bin/bash
```

所以，当一个宿主机中运行了多个 Docker 容器的时候，我们可以通过 CGroups，给每一个容器弹性地分配 CPU 资源。同样地，这个限制既不能过松，过松会导致某一个 Docker 容器耗尽宿主机资源，也不能过严，过严会使得容器内的服务得不到足够的资源支持。这都需要我们自己经过慎重考量来进行配置，没有默认的安全机制可以辅助我们。

现在，你应该已经了解 Docker 服务中的 3 个主要机制了。这里，我把这 3 个主要机制的特点总结成了一张表格，帮助你加深理解。

| 关键机制 | 功能 | 安全隐患 | 默认安全机制 |
|--------------|----------------|-------------------------------|------------------|
| Namespace | 容器和宿主机环境的“伪隔离” | 容器可以修改宿主机的部分环境 | 通过Capabilities限制 |
| Capabilities | 对容器可以进行的操作进行限制 | 限制过松导致隔离失效，限制过严导致容器的功能受影响 | 白名单限制 |
| CGroups | 限制容器的资源使用情况 | 限制过松导致宿主机资源被耗尽，限制过严导致容器的功能受影响 | 无，需要配置 |



Docker 守护进程

想要运行 Docker 镜像，就必须先启动 Docker 的 Daemon 守护进程。而启动这个守护进程需要 ROOT 权限。因此，守护进程本身如果出现漏洞，就会给黑客提供一个权限提升的入口。那通过这个守护进程，黑客能进行哪些操作呢？

首先，作为守护进程，Daemon 具备操控 Docker 容器的全部权限。这也就意味着，黑客可以任意地上线和下线容器、运行黑客自己的镜像、篡改已有镜像的配置等。这么说可能不够直观，我来详细解释一下。黑客通过守护进程，可以将宿主机的根目录共享到镜像中，这样一来，镜像就可以对宿主机的目录进行任意地修改了。另外，除了影响正常的线上容器，黑客还能够通过简单的 `docker exec` 命令获取容器环境中的 Shell，从而执行任意命令了。

那么，黑客怎么才能控制 Daemon 守护进程呢？最简单的方法当然是直接进入宿主机，通过 Docker 命令进行交互。但如果黑客已经进入宿主机，还去操控容器，就是多此一举了。所以，黑客主要是通过远程 API，来对 Docker 守护进程发起攻击。


守护进程提供的 API 接口，是为了方便用户去做一些自动化的工具，来操控 Docker 容器。而在默认情况下，这个 API 接口不需要进行认证。你可以尝试探测一下，你的公司内外网中，是否存在开放的 2375 端口（守护进程 API 默认监听的端口）。如果存在的话，那么你基本上就能够控制这台服务器的 Docker 守护进程了。

为了避免这种无认证的情况发生，Docker 提供了证书的方式来进行认证。开启 API 接口的命令如下所示：

 复制代码

```
1 dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem --tlskey=server-key.pem
```

通过以上命令，我们就能够在宿主机开启远程 API 接口。在客户端中，只需要提供相应的证书信息，就能够完成经过认证的 API 接口调用了。

 复制代码

```
1 curl https://127.0.0.1:2376/images/json --cert cert.pem --key key.pem --cacert ca.pem
```

那通过这样的配置，我们就能解决了 API 接口的认证问题，也就提升了 Docker 守护进程的安全性。

Docker 镜像安全

了解了 Docker 守护进程的安全风险和防护方法之后，我们再来看一下 Docker 镜像的安全。


对于 Docker 镜像来说，它本身就是一个模拟的操作系统，自然也会存在操作系统中的各类安全威胁和漏洞。但是，由于一个 Docker 镜像，一般只会运行某一种服务，也就相当于一个操作系统中只有一个用户。因此，Docker 镜像面临的安全威胁也会小很多。

接下来，我就为你详细讲解两种保证 Docker 镜像安全的方式，分别是“使用最精简的镜像”和“最小权限原则”。

使用最精简的镜像

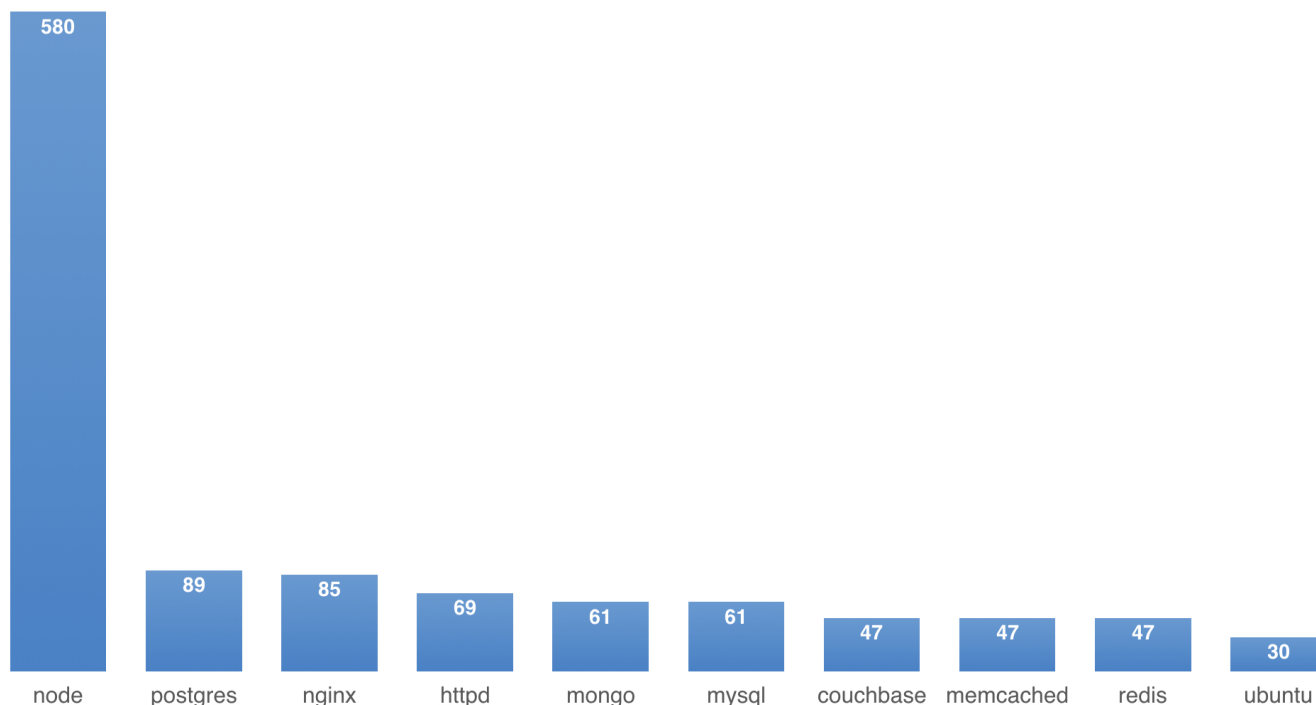
前面我们讲了 Docker 镜像的概念，我们知道，Docker 镜像是通过 Dockerfile 来构建的。而 Dockerfile 构建的第一句是 FROM ***。以 Node.js 的环境为例，你的基础镜像可能是 node，那么 Dockerfile 的第一行应该是 FROM node。

```
1 FROM node
2 COPY . ./
3 EXPOSE 8080
4 CMD ["node", "index.js"]
```

 复制代码

这个基础的 node 镜像实际包含了一个完整的操作系统，但是，在实际应用中，有大部分的系统功能，我们是用不到的。而这些用不到的系统功能，却正好为黑客提供了可乘之机。

Snyk 在 2019 年的 [🔗 Docker 漏洞统计报告](#)称，最热门的 10 个 Docker 基础镜像，包含的已知系统漏洞，最少的有 30 个，最多的有 580 个。



这是非常惊人的。通过一句简单的 FROM node，就能让你的 Docker 镜像中引入 580 个系统漏洞。那我们该如何避免引入漏洞呢？这个时候，我们就需要使用精简版的基础镜像了。一般来说，精简版的 Docker 镜像标签都会带有 slim 或者 alpine。

比如说，如果你采用 node:10-slim，那么漏洞数会降低到 71 个。如果使用 node:10-alpine，那么已知的漏洞数会降为 0。之所以会发生这种现象，是因为使用精简版的基础镜像，可以去除大部分无用的系统功能和依赖库，所以，存在于这些功能中的漏洞自然也就被剔除了。


因此，对于 Docker 来说，通过使用精简的基础镜像，去除一些无用的系统功能，既能够降低最终镜像的体积，又能够降低安全风险，何乐而不为呢？

Docker 中的最小权限原则

除此之外，我们在 Linux 操作系统中提到的最小权限原则，在 Docker 镜像中同样适用。


这是因为，在默认情况下，容器内的进程是都以 ROOT 权限启动的。而 Docker 又是伪隔离，所以，容器就和宿主机拥有一致的 ROOT 权限了。虽然 Docker 通过 Capabilities，对容器内的 ROOT 能力进行了限制。但是，使用 ROOT 权限去运行一个普通的服务很不合适。为此，我们可以通过 USER 关键词，来使用一个低权限的用户运行服务。

以 Node.js 为例，在 node 的基础镜像中，默认创建了 node 这么一个具备较小权限的用户。因此，我们可以在 Dockerfile 中，加入一行 USER node 来使用这个最小权限用户。

 复制代码

```
1 FROM node:10-alpine
2 ...
3 USER node
4 CMD ["node", "index.js"]
```

当然，如果有的基础镜像本身不提供额外的用户，你就需要自己创建一个了。以 ubuntu 为例，我们可以通过 groupadd 和 useradd，创建一个 node 用户，这个用户没有密码、没有 home 目录、也没有 shell，就是一个最小权限用户。Dockerfile 的内容如下：

 复制代码

```
1 FROM ubuntu
2 RUN groupadd -r node && useradd -r -s /bin/false -g node node
3 ...
4 USER node
5 CMD node index.js
```

现在，你应该已经知道 Docker 镜像的两种安全防护方法了，我来简单总结一下。第一个是通过使用最精简的基础镜像，来删减 Docker 镜像中不必要的功能，从而降低出现漏洞的概率。第二个则是采取最小权限原则，以低权限用户来执行服务，限制黑客的能力。

总结

好了，今天的内容讲完了。我们来一起总结回顾一下，你需要掌握的重点内容。

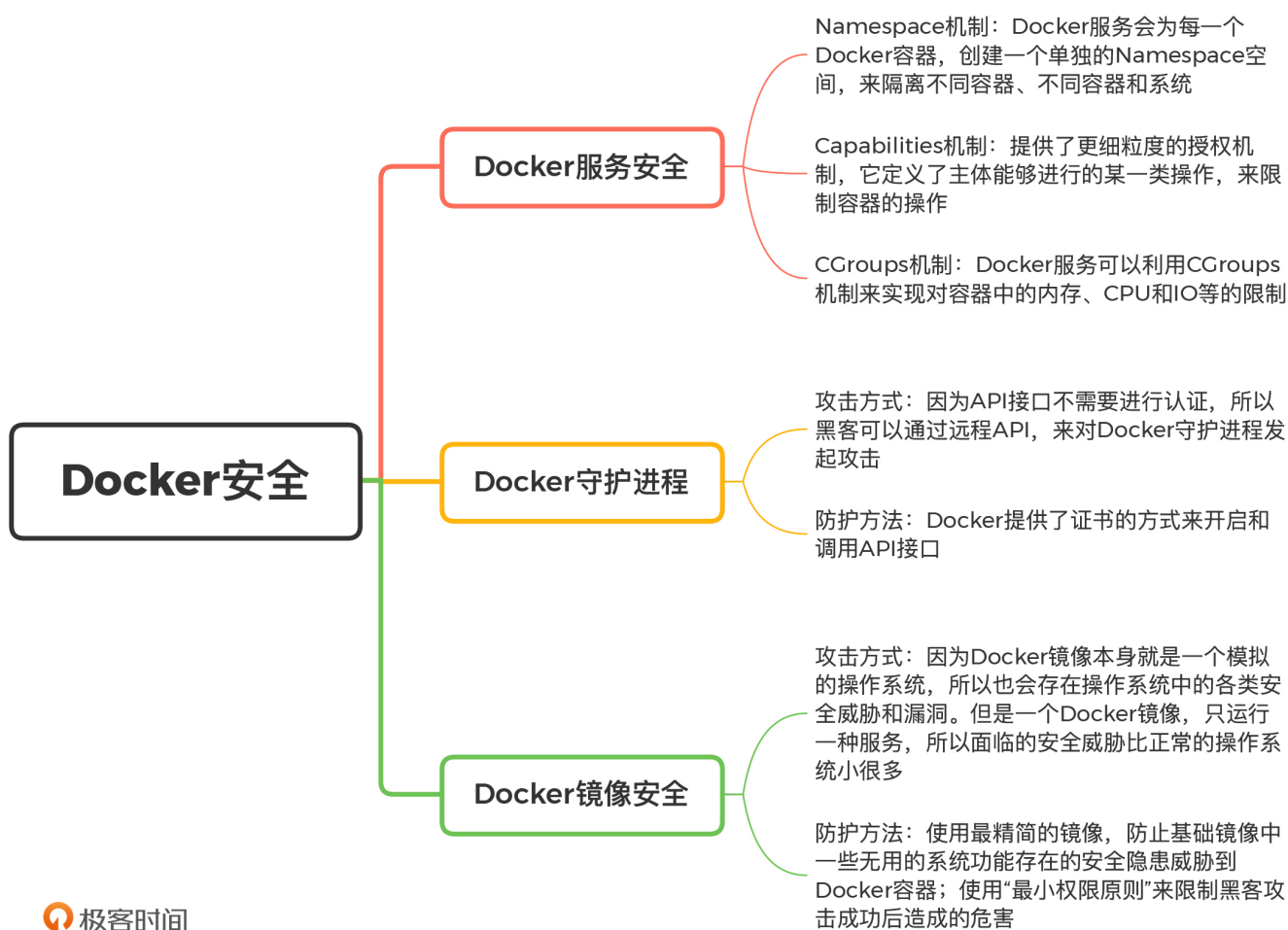
今天，我主要通过 Docker 服务、Docker 守护进程和 Docker 镜像这三个方面，带你学习 Docker 的安全性。

在 Docker 服务中，主要是利用 Namespace、Capabilities 和 CGroups 机制，来对 Docker 容器进行各种隔离和限制；在 Docker 守护进程中，我们通过给远程 API 加上认证功能来保证安全性；在 Docker 镜像中，我们主要是通过最小镜像和最小权限的原则，去提升镜像本身的安全性。

在实际对 Docker 进行安全防护的过程中，我们也可以采取各类针对 Docker 的扫描工具，来发现问题。比如 [Clair](#) ，它会对你的镜像进行静态的扫描分析，并和漏洞库进行比对，从而发现镜像中可能存在的安全漏洞。

以 Docker 为代表的容器技术，可以说是现在应用开发中最常见的技术了。很多开发人员，现在甚至不用使用原始的 Linux 系统，直接基于 Docker 进行开发就好了。因此，我们在开发应用的过程中，要时刻关注 Docker 的安全性。

好了，我把这一讲的重点内容梳理了一个脑图。你可以用它来查漏补缺，也可以自己来梳理看看，加深印象。



思考题

最后，给你留一个思考题。

“容器上云”是目前普遍的技术趋势，你是否有使用过一些容器云的产品？可以研究一下，在容器云中，云平台给容器设置了哪些安全限制。

欢迎留言和我分享你的思考和疑惑，也欢迎你把文章分享给你的朋友。我们下一讲再见！

点击查看 

来参加打卡，攻克 工作中 80% 的安全问题



PC端用户扫码参与



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 14 | 网络安全：和别人共用Wi-Fi时，你的信息会被窃取吗？

下一篇 16 | 数据库安全：数据库中的数据是如何被黑客拖取的？

精选留言 (5)

 写留言

早起不吃虫

2020-01-16

老师每篇结尾的总结真的是太棒了！

展开 



王凯

2020-01-16

学习了两个新知识点：

1. docker远程API
2. slim 或者 alpine标签的意义

谢谢



leslie

2020-01-15

记得2019年11月GOPS问过胥峰老师：云与云之间的安全策略其实是需要考虑的，不过如何做没过多提及；这又是一个现在最常见的场景；不知道老师有没有什么好的策略分享一下，或者放在《特别加餐》里面帮大家解决一下这种公共场景，毕竟企业稍微大点就会出现这种情况-跨云之间的安全策略。

谢谢老师今天的分享：希望老师能针对这种典型场景给大家分享一下老师的解决策略。
展开 ∨

作者回复: 抱歉，云安全这一块没有做过特别深入的研究，所以没有特别多的内容可以分享。我司也有使用阿里云或者华为云等服务，不过都是走的专线连接，相当于把云服务器拉入了内网，我理解其实是不存在额外的云安全问题的，所以并没有关注这个问题。



Cy23

2020-01-15

一直还没有考虑docker还得注意安全性问题，看来以后再看docker的时候需要注意一下。



陈优雅

2020-01-15

由于容器里面也可以运行程序，怎么保证正在运行的容器不被篡改？或者怎么发现容器是否被入侵？

作者回复: 容器内部入侵和Linux系统基本一样。容器不被篡改，就是依靠docker守护进程不被控制。

