



下载APP



## 19 | 容器安全（1）：我的容器真的需要privileged权限吗？

2020-12-28 李程远

容器实战高手课

[进入课程 >](#)**讲述：李程远**

时长 12:53 大小 11.81M



你好，我是程远。从今天这一讲，我们进入到了容器安全的模块。

容器安全是一个很大的话题，容器的安全性很大程度是由容器的架构特性所决定的。比如容器与宿主机共享 Linux 内核，通过 Namespace 来做资源的隔离，通过 shim/runC 的方式来启动等等。

这些容器架构特性，在你选择使用容器之后，作为使用容器的用户，其实你已经没有多少能力去对架构这个层面做安全上的改动了。你可能会说用 [Kata Container](#)、[gVisor](#) 就是安全“容器”了。不过，Kata 或者 gVisor 只是兼容了容器接口标准，而内部的实现全是另外的技术了。





那么对于使用容器的用户，在运行容器的时候，在安全方面可以做些什么呢？我们主要可以从这两个方面来考虑：第一是赋予容器合理的 capabilities，第二是在容器中以非 root 用户来运行程序。

为什么是这两点呢？我通过两讲的内容和你讨论一下，这一讲我们先来看容器的 capabilities 的问题。

## 问题再现

刚刚使用容器的同学，往往会发现用缺省 docker run的方式启动容器后，在容器里很多操作都是不允许的，即使是以 root 用户来运行程序也不行。

我们用下面的  例子来重现一下这个问题。我们先运行make image 做个容器镜像，然后运行下面的脚本：

 复制代码

```
1 # docker run --name iptables -it registry/iptables:v1 bash
2 [root@0b88d6486149 /]# iptables -L
3 iptables v1.8.4 (nf_tables): Could not fetch rule set generation id: Permissio
4
5 [root@0b88d6486149 /]# id
6 uid=0(root) gid=0(root) groups=0(root)
```

在这里，我们想在容器中运行 iptables 这个命令，来查看一下防火墙的规则，但是执行命令之后，你会发现结果输出中给出了"Permission denied (you must be root)"的错误提示，这个提示要求我们用 root 用户来运行。

不过在容器中，我们现在已经是 root 用户来运行了，么为什么还是不可以运行"iptables"这条命令呢？

你肯定会想到，是不是容器中又做了别的权限限制？如果你去查一下资料，就会看到启动容器有一个"privileged"的参数。我们可以试一下用上这个参数，没错，我们用了这个参数之后，iptables 这个命令就执行成功了。

 复制代码

```
1 # docker stop iptables;docker rm iptables
```

```
2 iptables
3 iptables
4 # docker run --name iptables --privileged -it registry/iptables:v1 bash
5 [root@44168f4b9b24 /]# iptables -L
6 Chain INPUT (policy ACCEPT)
7 target     prot opt source                destination
8
9 Chain FORWARD (policy ACCEPT)
10 target     prot opt source                destination
11
12 Chain OUTPUT (policy ACCEPT)
13 target     prot opt source                destination
```

看上去，我们用了一个配置参数就已经解决了问题，似乎很容易。不过这里我们可以进一步想想，用"privileged"参数来解决问题，是不是一个合理的方法呢？用它会有什么问题吗？

要回答这些问题，我们先来了解一下"privileged"是什么意思。从 Docker 的 [代码](#)里，我们可以看到，如果配置了 privileged 的参数，就会获取所有的 capabilities，那什么是 capabilities 呢？

[复制代码](#)

```
1         if ec.Privileged {
2             p.Capabilities = caps.GetAllCapabilities()
3         }
```

## 基本概念

### Linux capabilities

要了解 Linux capabilities 的定义，我们可以先查看一下"Linux Programmer's Manual"中关于 [Linux capabilities](#) 的描述。

在 Linux capabilities 出现前，进程的权限可以简单分为两类，第一类是特权用户的进程（进程的有效用户 ID 是 0，简单来说，你可以认为它就是 root 用户的进程），第二类是非特权用户的进程（进程的有效用户 ID 是非 0，可以理解为非 root 用户进程）。

特权用户进程可以执行 Linux 系统上的所有操作，而非特权用户在执行某些操作的时候就会被内核限制执行。其实这个概念，也是我们通常对 Linux 中 root 用户与非 root 用户的

理解。

从 kernel 2.2 开始，Linux 把特权用户所有的这些“特权”做了更详细的划分，这样被划分出来的每个单元就被称为 capability。

所有的 capabilities 都在 [Linux capabilities](#) 的手册列出来了，你也可以在内核的文件 [capability.h](#) 中看到所有 capabilities 的定义。

对于任意一个进程，在做任意一个特权操作的时候，都需要有这个特权操作对应的 capability。

比如说，运行 iptables 命令，对应的进程需要有 CAP\_NET\_ADMIN 这个 capability。如果要 mount 一个文件系统，那么对应的进程需要有 CAP\_SYS\_ADMIN 这个 capability。

我还要提醒你的是，CAP\_SYS\_ADMIN 这个 capability 里允许了大量的特权操作，包括文件系统，交换空间，还有对各种设备的操作，以及系统调试相关的调用等等。

在普通 Linux 节点上，非 root 用户启动的进程缺省没有任何 Linux capabilities，而 root 用户启动的进程缺省包含了所有的 Linux capabilities。

我们可以做个试验，对于 root 用户启动的进程，如果把 CAP\_NET\_ADMIN 这个 capability 移除，看看它是否还可以运行 iptables。


在这里我们要用到 [capsh](#) 这个工具，对这个工具不熟悉的同学可以查看超链接。接下来，我们就用 capsh 执行下面的这个命令：

复制代码

```
1 # sudo /usr/sbin/capsh --keep=1 --user=root --drop=cap_net_admin -- -c '.
2 Chain INPUT (policy ACCEPT)
3 target      prot opt source                destination
4
5 Chain FORWARD (policy ACCEPT)
6 target      prot opt source                destination
7
8 Chain OUTPUT (policy ACCEPT)
9 target      prot opt source                destination
10 iptables: Permission denied (you must be root).
```

这时候, 我们可以看到即使是 root 用户, 如果把"CAP\_NET\_ADMIN"给移除了, 那么在执行 iptables 的时候就会看到"Permission denied (you must be root)."的提示信息。

同时, 我们可以通过 /proc 文件系统找到对应进程的 status, 这样就能确认进程中的 CAP\_NET\_ADMIN 是否已经被移除了。

 复制代码

```
1 # ps -ef | grep sleep
2 root      22603 22275  0 19:44 pts/1    00:00:00 sudo /usr/sbin/capsh --keep=1
3 root      22604 22603  0 19:44 pts/1    00:00:00 /bin/bash -c ./iptables -L;sle
4
5 # cat /proc/22604/status | grep Cap
6 CapInh:    0000000000000000
7 CapPrm:    00000003fffffefff
8 CapEff:    00000003fffffefff
9 CapBnd:    00000003fffffefff
10 CapAmb:    0000000000000000
```

运行上面的命令查看 /proc//status 里 Linux capabilities 的相关参数之后, 我们可以发现, 输出结果中包含 5 个 Cap 参数。

这里我给你解释一下, 对于当前进程, 直接影响某个特权操作是否可以被执行的参数, 是"CapEff", 也就是"Effective capability sets", 这是一个 bitmap, 每一个 bit 代表一项 capability 是否被打开。

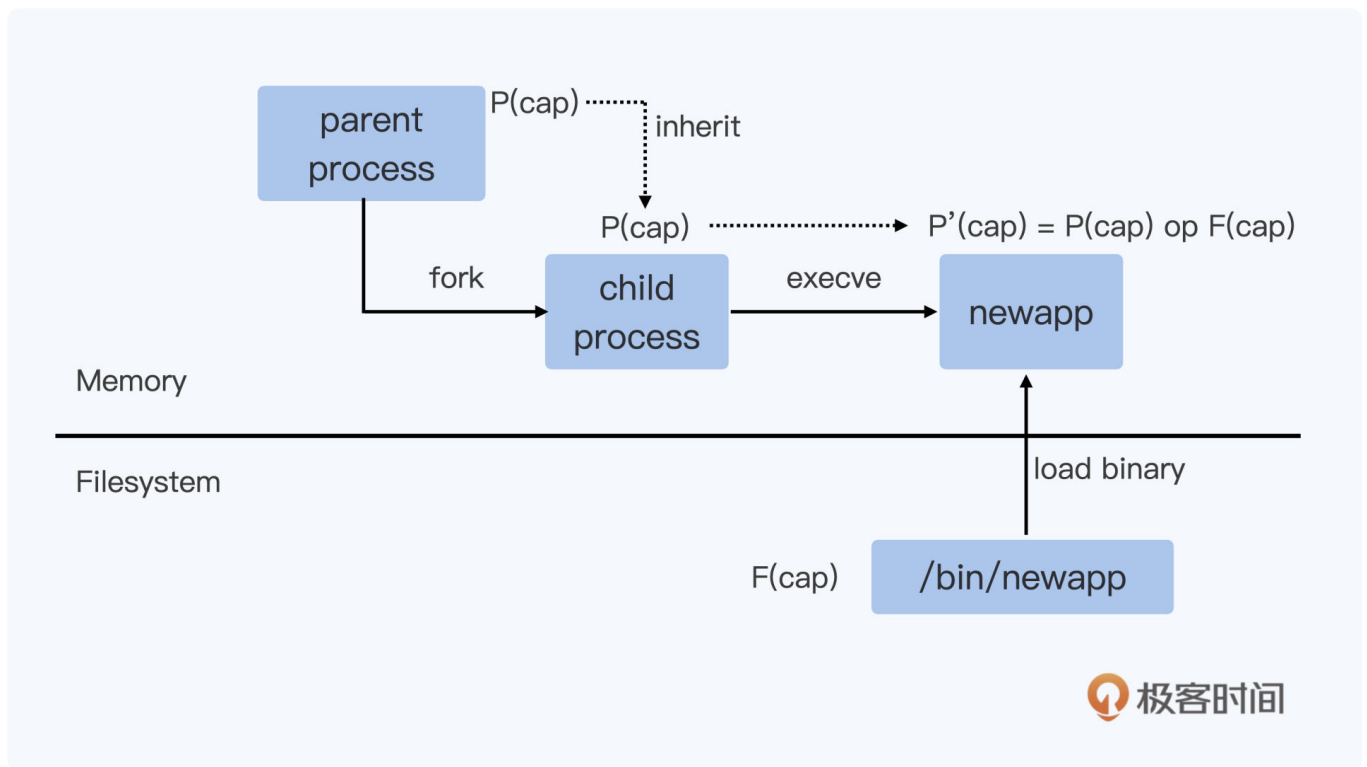
在 Linux 内核 [capability.h](#) 里把 CAP\_NET\_ADMIN 的值定义成 12, 所以我们可以看到"CapEff"的值是"00000003fffffefff", 第 4 个数值是 16 进制的"e", 而不是 f。

这表示 CAP\_NET\_ADMIN 对应的第 12-bit 没有被置位了 ( $0xefff = 0xffff \& (\sim(1 \ll 12))$ ), 所以这个进程也就没有执行 iptables 命令的权限了。

对于进程 status 中其他几个 capabilities 相关的参数, 它们还需要和应用程序文件属性中的 capabilities 协同工作, 这样才能得到新启动的进程最终的 capabilities 参数的值。

我们看下面的图, 结合这张图看后面的讲解:





如果我们要新启动一个程序，在 Linux 里的过程就是先通过 `fork()` 来创建出一个子进程，然后调用 `execve()` 系统调用读取文件系统里的程序文件，把程序文件加载到进程的代码段中开始运行。

就像图片所描绘的那样，这个新运行的进程里的相关 `capabilities` 参数的值，是由它的父进程以及程序文件中的 `capabilities` 参数值计算得来的。


具体的计算过程你可以看 [Linux capabilities](#) 的手册中的描述，也可以读一下网上的这两篇文章：

[Capabilities: Why They Exist and How They Work](#)

[Linux Capabilities in Practice](#)

我就不对所有的进程和文件的 `capabilities` 集合参数和算法挨个做解释了，感兴趣的话你可以自己详细去看看。

这里你只要记住最重要的一点，**文件中可以设置 `capabilities` 参数值，并且这个值会影响到最后运行它的进程。**比如，我们如果把 `iptables` 的应用程序加上 `CAP_NET_ADMIN` 的 `capability`，那么即使是非 `root` 用户也有执行 `iptables` 的权限了。

 复制代码

```
1 $ id
2 uid=1000(centos) gid=1000(centos) groups=1000(centos),10(wheel)
3 $ sudo setcap cap_net_admin+ep ./iptables
4 $ getcap ./iptables
5 ./iptables = cap_net_admin+ep
6 $ ./iptables -L
7 Chain INPUT (policy ACCEPT)
8 target      prot opt source                destination
9
10 Chain FORWARD (policy ACCEPT)
11 target      prot opt source                destination
12 DOCKER-USER  all  --  anywhere              anywhere
13 DOCKER-ISOLATION-STAGE-1 all  --  anywhere              anywhere
14 ACCEPT      all  --  anywhere              anywhere             ctstate RELATED,
15 DOCKER      all  --  anywhere              anywhere
16 ACCEPT      all  --  anywhere              anywhere
17 ACCEPT      all  --  anywhere              anywhere
18 ...
```


好了，关于 Linux capabilities 的内容到这里我们就讲完了，其实它就是把 Linux root 用户原来所有的特权做了细化，可以更加细粒度地给进程赋予不同权限。

## 解决问题

我们搞懂了 Linux capabilities 之后，那么对 privileged 的容器也很容易理解了。

**Privileged 的容器也就是允许容器中的进程可以执行所有的特权操作。**

因为安全方面的考虑，容器缺省启动的时候，哪怕是容器中 root 用户的进程，系统也只允许了 15 个 capabilities。这个你可以查看 [runC spec 文档中的 security 部分](#)，你也可以查看容器 init 进程 status 里的 Cap 参数，看一下容器中缺省的 capabilities。

 复制代码

```
1 # docker run --name iptables -it registry/iptables:v1 bash
2 [root@e54694652a42 /]# cat /proc/1/status |grep Cap
3 CapInh:                00000000a80425fb
4 CapPrm:                00000000a80425fb
5 CapEff:                00000000a80425fb
6 CapBnd:                00000000a80425fb
7 CapAmb:                0000000000000000
```

我想提醒你，当我们发现容器中运行某个程序的权限不够的时候，并不能“偷懒”把容器设置为"privileged"，也就是把所有的 capabilities 都赋予了容器。

因为容器中的权限越高，对系统安全的威胁显然也是越大的。比如说，如果容器中的进程有了 CAP\_SYS\_ADMIN 的特权之后，那么这些进程就可以在容器里直接访问磁盘设备，直接可以读取或者修改宿主机上的所有文件了。

所以，在容器平台上是基本不允许把容器直接设置为"privileged"的，我们需要根据容器中进程需要的最少特权来赋予 capabilities。

我们结合这一讲开始的例子来说。在开头的例子中，容器里需要使用 iptables。因为使用 iptables 命令，只需要设置 CAP\_NET\_ADMIN 这个 capability 就行。那么我们只要在运行 Docker 的时候，给这个容器多加一个 NET\_ADMIN 参数就可以了。

[复制代码](#)

```
1 # docker run --name iptables --cap-add NET_ADMIN -it registry/iptables:v1 bash
2 [root@cfedf124dcf1 /]# iptables -L
3 Chain INPUT (policy ACCEPT)
4 target      prot opt source                destination
5
6 Chain FORWARD (policy ACCEPT)
7 target      prot opt source                destination
8
9 Chain OUTPUT (policy ACCEPT)
10 target     prot opt source                destination
```

## 重点小结

这一讲我们主要学习了如何给容器赋予合理的 capabilities。

那么，我们自然需要先来理解什么是 Linux capabilities。其实 Linux capabilities 就是把 Linux root 用户原来所有的特权做了细化，可以更加细粒度地给进程赋予不同权限。

对于 Linux 中的每一个特权操作都有一个对应的 capability，对于一个 capability，有的对应一个特权操作，有的可以对应很多个特权操作。



每个 Linux 进程有 5 个 capabilities 集合参数, 其中 Effective 集合里的 capabilities 决定了当前进程可以做哪些特权操作, 而其他集合参数会和应用程序文件的 capabilities 集合参数一起来决定新启动程序的 capabilities 集合参数。

对于容器的 root 用户, 缺省只赋予了 15 个 capabilities。如果我们发现容器中进程的权限不够, 就需要分析它需要的最小 capabilities 集合, 而不是直接赋予容器"privileged"。

因为"privileged"包含了所有的 Linux capabilities, 这样"privileged"就可以轻易获取宿主主机上的所有资源, 这会对宿主机的安全产生威胁。所以, 我们要根据容器中进程需要的最少特权来赋予 capabilities。

## 思考题

你可以查看一下你的 Linux 系统里 ping 程序文件有哪些 capabilities, 看看有什么办法, 能让 Linux 普通用户没有执行 ping 的能力。

欢迎你在留言区和我交流互动。如果学完这一讲让你有所收获, 也欢迎转发给你的同事、或者朋友, 一起交流探讨容器安全的问题。

提建议

© 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 18 | 容器网络配置 (3) : 容器中的网络乱序包怎么这么高?

下一篇 20 | 容器安全 (2) : 在容器中, 我不以root用户来运行程序可以吗?

## 精选留言 (5)

写留言



莫名

2020-12-29

```
getcap $(which ping)
setcap -r $(which ping)
```

顺便举个之前使用过的例子：普通用户默认没有 tcpdump 抓包权限，可添加 net\_raw、net\_admin caps: ...

展开 ▾

作者回复: 赞!



10

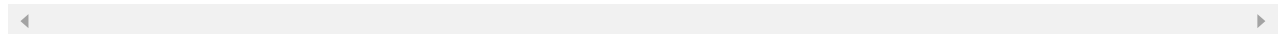


**朱新威**

2020-12-28

已经更新20讲了，莫名有点心慌，生怕这么好的专栏结束了😭

编辑回复: 哈哈别慌哦，咱们后续还有专题加餐，等更新的时间里，你还可以复习已有内容哦。



3



**老酒馆**

2020-12-28

```
getcap /usr/bin/ping 查看ping进程当前cap
setcap cap_net_admin,cap_net_raw+p /usr/bin/ping 设置ping进程cap
```

展开 ▾



1



**morse**

2021-01-11

老师, 您好, 我在 Ubuntu20.04 下删除 ping 的 capabilities 后, 切换别的用户后, 还是可以使用 ping 的, 我进行了以下操作.

```
# getcap /usr/bin/ping # 发现ping 具有cap_net_raw capability
```

```
/usr/bin/ping = cap_net_raw+ep...
```

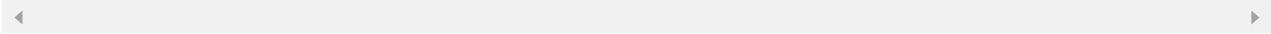
展开 ▾

作者回复: @morse

你用capsh看到的cap\_net\_raw 应该是在Binding Cap而不是在Effective Cap里。

ubuntu20.04 里安装的ping程序本身允许普通用户ping ICMP.

<https://unix.stackexchange.com/questions/617927/why-ping-works-without-capability-and-setuid>



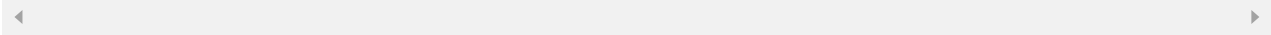
**Tony**

2020-12-28

老师你好。请问在Linux中（比如centos），在允许普通用户使用docker以后，如何如何限制用户不能读取，宿主机上非该普通用户的文件？

展开 ∨

作者回复: 我想你问的问题是关于rootless container的？普通用户可以启动容器，但是用户权限并没有提高，非该用户的文件还是不能读写。



1

