容器可用安全策略

> 本节是第六部分“安全篇”的第二篇，在这个部分，我将用四篇内容为你介绍包括镜像，容器和 Linux 内核的 LSM 等内容。上篇，我为你介绍了容器镜像安全相关的内容。本篇，我们将重点放在容器安全上。

在第一部分“容器篇”中，我已经带你深入剖析了 Docker 容器的本质，以及其所用到的核心技术 cgroups 和 namespace。

在上一篇中，我为你介绍了镜像安全相关的内容，分享了一些思路及方法，本篇，我们将容器放在容器安全上。

那么容器可能会有哪些安全问题呢？这就要从 Linux 的权限说起了。

Linux 的权限模型

Linux 的权限模型在早期只分为两类：特权进程和非特权进程。

- 特权进程：有效 UID 为 0（一般也称为超级用户或者 root 用户）的进程，特权进程可以绕过所有的内核权限检查。
- 非特权进程：有效 UID 为非 0 的进程，非特权进程需要根据进程的凭证（有效 UID、有效 GID 等）进行相应的权限检查。

在这种模型下，很容易发生权限失控的问题。如果是一般用户想要提升权限执行某些命令的话，通常有两种方法：一种是使用 sudo 来完成，但如果服务器上有很多用户，且这些用户需要有不同的权限控制，那就会很繁琐；另一种是使用 SUID 的方式，允许一般用户来运行一个归属于 root 的文件，并同时具备 root 的权限，但这种情况下便更加不容易控制用户的权限了。

所以从 Linux 2.2 开始便将与超级用户相关联的特权划分为了不同的单元，称之为 capabilities 并且可以独立的启用或者禁用。capabilities 当前作为每个线程的属性存在。

修改之后的权限模型从整体来看流程就变成了：先判断是否为超级用户，如果是超级用户则放行；如果不是超级用户，那么再去检查是否有执行操作所需的相关 capabilities，如果有，则可执行对应的操作。

可用的 capabilities 很多，这里就不一一列出了，可以查看 Linux 手册中 capabilities 的部分来获得完整列表。

容器的权限

说完 Linux 的权限模型，我们再将重点放回到容器上。默认情况下，如果不指定用户，容器中会使用 root 用户来执行操作。例如：

复制

```
(MoeLove) ➜  ~ docker run --rm -it alpine
/ # whoami
root
/ # id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(w
```

由于容器的实现是共享内核的，如果不做任何处理，使用 root 用户是非常危险的。另外，如果攻击者取得容器的权限，则可能会通过容器进而攻击到主机，引起更为严重的安全问题。

考虑到这些方面，Docker 也提供了很多特性。

使用非 root 用户

在 Dockerfile 中通过 USER 指令指定 uid、gid 的方式在之前内容中已经介绍过了，这里重点放在启动容器时的操作上。

先在主机上查看当前的用户及其 uid 等信息：

复制

```
(MoeLove) ➜  ~ whoami
tao
(MoeLove) ➜  ~ id tao
uid=1000(tao) gid=1000(tao) groups=1000(tao),10(wheel),976(docker)
```

可以看到我当前使用的 uid 和 gid 都是 1000，现在我要启动一个 Alpine Linux 的容器：

复制

```
(MoeLove) ➜  ~ docker run --user 1000:1000 --rm -it alpine
/ $ id
uid=1000 gid=1000
/ $ ps
PID   USER     TIME  COMMAND
    1 1000      0:00 /bin/sh
   27 1000      0:00 ps
/ $ apk add bash
ERROR: Unable to lock database: Permission denied
ERROR: Failed to open apk database: Permission denied
/ $ ls root/
ls: can't open 'root/': Permission denied
```

你会发现当使用 `--user` 指定 uid 和 gid 后，容器内的进程会以指定用户运行，并且它只具备一般用户的权限。比如：无法访问 root 用户的目录，无法使用 apk 安装新的软件包之类的。

普通用户的特权操作

我们继续上面的例子，执行一个 ping 命令：

复制

```
/ $ ping moelove.info
PING moelove.info (185.199.109.153): 56 data bytes
ping: permission denied (are you root?)
```

你会发现，有权限问题，提示不能执行 ping 命令。

此时你先别急，打开另一个终端，执行以下命令：

复制

```
# 使用 root:root 进入容器执行操作
(MoeLove) ➜  ~ docker exec --user root:root -it $(docker ps -ql) sh
/ # id
uid=0(root) gid=0(root)
/ # ping -c 1 moelove.info
PING moelove.info (185.199.108.153): 56 data bytes
64 bytes from 185.199.108.153: seq=0 ttl=51 time=73.257 ms

--- moelove.info ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 73.257/73.257/73.257 ms
/ # which ping
/bin/ping
/ # ls -al /bin/ping
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 /bin/ping -> /bin/busybox
```

通过上面的操作，我们使用 `root:root` 的身份进入了容器，并执行了相应的操作，同时查看 ping 命令的位置，发现它其实是 busybox 提供的，所以我们来安装一个原生的 ping 命令。

复制

```
/ # apk add --no-cache iputils
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.11/community/x86_64/APKINDEX.tar.gz
(1/2) Installing libcap (2.27-r0)
(2/2) Installing iputils (20190709-r0)
Executing busybox-1.31.1-r9.trigger
OK: 6 MiB in 16 packages
/ # ls -al /bin/ping
-rwsr-xr-x    1 root     root         60232 Oct 22 13:16 /bin/ping
```

现在，回到刚才普通用户的终端窗口，再次执行 ping 命令：

复制

```
/ $ id
uid=1000 gid=1000
/ $ ls -al /bin/ping
-rwsr-xr-x    1 root     root         60232 Oct 22 13:16 /bin/ping
/ $ ping -c 1 moelove.info
PING moelove.info (185.199.108.153) 56(84) bytes of data.
64 bytes from 185.199.108.153 (185.199.108.153): icmp_seq=1 ttl=51 time=71.6 ms

--- moelove.info ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 71.568/71.568/71.568/0.000 ms
```

发现可以正常的执行 ping 命令了。有没有一丝好奇？

这里涉及到了两个问题：

- busybox 带有权限检查，这不是本篇的重点，暂且略过；
- 为何原生的 ping 命令可以被普通用户正常执行？

注意：ping 命令需要有 socket 连接（RAW socket），这其实是个特权，普通用户是不具备这种权限的。

这里重点看看刚才 /bin/ping 文件的权限：

复制

```
/ $ ls -al /bin/ping
-rwsr-xr-x    1 root     root         60232 Oct 22 13:16 /bin/ping
```

可以看到它是有 s 位的，这也就是上面提到的 SUID 的方式了。所以普通用户（1000:1000）可以顺利地执行 ping 命令。

我们试试看将 s 位权限去掉。回到 root 用户的终端中：

复制

```
/ # id
uid=0(root) gid=0(root)
/ # chmod u-s /bin/ping
/ # ls -al /bin/ping
-rwxr-xr-x    1 root     root         60232 Oct 22 13:16 /bin/ping
```

再次验证普通用户是否可以执行 ping 命令：

复制

```
/ $ id
uid=1000 gid=1000
/ $ ping -c 1 moelove.info
ping: socket: Operation not permitted
```

发现它不再能够提升权限了。

Docker 容器额外的权限

我们继续这个例子，刚才已经将 /bin/ping 文件权限的 s 位去掉了，普通用户已经不能执行 ping 操作了。

那我们还有其他方法能让这个普通用户正常执行 ping 操作吗?

答案就在刚才介绍的 capabilities 中！为此，我们需要执行一些其他的操作，现在回到 root 用户的终端中，执行下面的步骤：

复制

```
/ # id
uid=0(root) gid=0(root)
/ # getcap /bin/ping
/ # setcap cap_net_raw+p /bin/ping
/ # getcap /bin/ping
/bin/ping = cap_net_raw+p
```

使用 getcap 命令先检查 /bin/ping 文件的 capabilities，发现没有任何结果。

使用 setcap 命令，为 /bin/ping 文件设置 capabilities，将其设置为 cap_net_raw+p，这表示为其设置可使用 cap_net_raw 的能力。

再次使用 getcap 命令查看，发现 capabilities 已经正确设置。

现在回到刚才普通用户的终端中：

复制

```
/ $ id
uid=1000 gid=1000
/ $ ls -al /bin/ping
-rwxr-xr-x    1 root     root         60232 Oct 22 13:16 /bin/ping
/ $ ping -c 1 moelove.info
PING moelove.info (185.199.108.153) 56(84) bytes of data.
64 bytes from 185.199.108.153 (185.199.108.153): icmp_seq=1 ttl=51 time=68.2 ms

--- moelove.info ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 68.231/68.231/68.231/0.000 ms
```

文件没有使用 SUID 的方式，但是普通用户也可以使用 ping 命令了，换句话说就是可以使用 RAW socket 的特权了。

以上，便是使用 capabilities 的例子。

我们再继续进行探索，思考下面的问题：

- 能设置 cap*net*raw 的 capabilities 是否意味着容器本身具备了 cap_net_raw 这个 capabilities?
- 是否所有的容器都有 cap_net_raw 这个 capabilities?
- 如何更改容器的 capabilities?

首先回答第一个问题：一般是的，但并不是绝对。通常情况下，只有对应的 capabilities 确实生效了，才能说明其具备对应的 capabilities，比如，可以通过以下命令查看当前具备了哪些 capabilities：

复制

```
/ # capsh --print
Current: = cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,ca
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid
Ambient set =
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
 secure-no-ambient-raise: no (unlocked)
uid=0(root)
gid=0(root)
groups=
```

为它设置一个当前不具备的 capabilities：

复制

```
/ # setcap cap_sys_admin+p /bin/ping
/ # getcap /bin/ping
/bin/ping = cap_sys_admin+p
```

发现可以正常设置，但实际上容器内并无此 capabilities，自然也就不能具备相应的 capabilities 了。

第二个问题。默认都有，这是 Docker 为了方便用户使用而提供的，是硬编码在代码中的：

复制

```
func DefaultCapabilities() []string {
    return []string{
        "CAP_CHOWN",
        "CAP_DAC_OVERRIDE",
        "CAP_FSETID",
        "CAP_FOWNER",
        "CAP_MKNOD",
        "CAP_NET_RAW",
        "CAP_SETGID",
        "CAP_SETUID",
        "CAP_SETFCAP",
        "CAP_SETPCAP",
        "CAP_NET_BIND_SERVICE",
        "CAP_SYS_CHROOT",
        "CAP_KILL",
        "CAP_AUDIT_WRITE",
    }
}
```

所以，一般情况下，我们不加任何参数启动一个容器时，它总是有能力去执行 ping 命令的。（需要安装 ping 命令）

第三个问题，如果更改容器的 capabilities?

在 `docker run` 的时候，Docker 提供了相应的参数：

复制

```
(MoeLove) ➜  ~ docker run --help |grep cap-
      --cap-add list                   Add Linux capabilities
      --cap-drop list                  Drop Linux capabilities
```

使用这两个参数便可控制容器可用的 capabilities 了。尝试下面的例子：

复制

```
# 排除掉 CAP_NET_RAW
(MoeLove) ➜  ~ docker run --rm -it --cap-drop=CAP_NET_RAW alpine
/ # apk add --no-cache -q iputils
/ # ping -c 1 moelove.info
ping: socket: Operation not permitted
```

另一个例子：

复制

```
# 排除所有 capabilities 唯独增加 CAP_NET_RAW
(MoeLove) ➜  ~ docker run --rm -it --cap-drop=ALL --cap-add=CAP_NET_RAW alpine
/ # apk add --no-cache -q iputils
ERROR: busybox-1.31.1-r9.trigger: script exited with error 127
/ # ping -c 1 moelove.info
PING moelove.info (185.199.109.153) 56(84) bytes of data.
64 bytes from 185.199.109.153 (185.199.109.153): icmp_seq=1 ttl=51 time=70.5 ms

--- moelove.info ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 70.547/70.547/70.547/0.000 ms
```

以上便是修改容器 capabilities 相关的方法了。

总结

本篇，我为你介绍了容器安全相关的内容，主要介绍了 Linux 的权限模型，以及容器 capabilities 相关的内容。

但这尚不是容器安全的全部，比如其中容器时，还有一个特殊的参数 `--privileged` 存在，即特权容器。

复制

```
(MoeLove) ➜  ~ docker run --help |grep privileged
      --privileged                       Give extended privileges to this container
```

在使用 Docker 时，要特别注意跟权限相关的问题，尤其是 `--privileged` 的参数，一定要搞清楚后再去使用。有时，可能只需要通过 `--cap-add` 来指定增加某个 capabilities 便可达到目的了。