Docker 与 secco...

> 本篇是第六部分“安全篇”的最后一篇，前面三篇，我为你介绍了镜像及容器安全相关的内容以及 Linux 的 LSM 如何为容器安全保驾护航。本篇，我们将重点放在 Linux 内核中 seccomp 与 Docker 之间的联系。

前面两篇，我分别为你介绍了如何使用 Linux capabilities 和 Linux LSM 模块为 Docker 容器提供安全的能力。本篇，我们将继续容器安全的话题，聊聊 Linux seccomp 与 Docker 之间的关系。

注意：本文所用的 Linux 内核为 5.4.10-100.fc30.x86_64，不同版本内核略有差异。

seccomp 基础

引用 Wiki 上的一小段关于 seccomp 的描述：

> seccomp 其实是 secure computing mode 的缩写，是 Linux 内核中的一个安全计算工具。seccomp 允许进程单向转换为“安全状态”，在此状态下的进程，除了 exit()，sigreturn()，read() 和 write() 已打开的文件描述符外等系统调用外，不允许执行其他任何系统调用。

如果该进程尝试进行其他的系统调用，则内核会使用 SIGKILL 或 SIGSYS 终止该进程。从这个意义上看，它并没有虚拟化系统资源，而是完全将其进行了隔离。

通俗点来说，seccomp 的作用相当于是充当了系统调用（syscall）的防火墙，它利用 BPF 规则来过滤系统调用，并控制如何处理它们。可以通过设置这些规则来限制 Docker 容器访问主机内核的权限。

另外：seccomp 早在 2005 年就已经合并到了 Linux 2.6.12 的主线中了。

如何使用

对 seccomp 有了大概认识后，我们来看看如何使用它。

首先，需要确保系统中已经启用了 seccomp，一般直接通过 CONFIG_SECCOMP 进行设置，也可以在系统启动时，通过设置内核命令行参数来覆盖它。比如，查看当前内核对 seccomp 的配置：

复制

```
(MoeLove) ➜  ~ grep CONFIG_SECCOMP /boot/config-$(uname -r)
CONFIG_SECCOMP=y
CONFIG_SECCOMP_FILTER=y
```

其次需要确认当前 Docker 是否启用了 seccomp 的配置：

```
(MoeLove) ➜  ~ docker info --format '{{ .SecurityOptions }}'
[name=seccomp,profile=default]
```

另外有一个需要额外注意的地方就是 seccomp profiles 需要 seccomp 2.2.1 及以上版本，当前主流的 Linux 发行版均已支持了，如果你遇到了比较旧的发行版，可以尝试去下载使用静态编译的 Docker 二进制程序，而不是使用包管理器直接安装 Docker。

Docker 中 seccomp 的配置

Docker Daemon 在启动时，有个 `--seccomp-profile` 的参数用于覆盖默认写在 Docker 源码中的 seccomp profile 文件，并且如果指定了该参数，则所有新启动的容器均会受该参数的影响，除非单独通过启动容器的参数进行覆盖。

```
(MoeLove) ➜  ~ dockerd --help |grep seccomp
      --seccomp-profile string                  Path to seccomp profile
```

在 Docker 源代码中包含着默认的 seccomp profile 的配置，可以在 Docker 源码的 profiles/seccomp/default.json 找到。篇幅原因，我这里就不贴出来了。

另一种在运行时修改 seccomp profile 配置的方法就是通过启动容器时的 `--security-opt` 参数进行覆盖。这个参数非常有用，除了能指定 seccomp profile 外，例如上一篇中提到的 LSM 对容器的支持，也可以通过 `--security-opt` 设置参数进行控制。

```
(MoeLove) ➜  ~ docker run --help |grep security-opt
      --security-opt list              Security Options
```

另外，**非常值得注意的是 Docker seccomp 使用的是白名单机制**，只有在白名单中的系统调用（syscall()）才被允许。

当然，Docker 也考虑到了如果全靠写配置文件来指定的话，现在 Linux 有三百多个系统调用，对用户而言过于麻烦了。所以 Docker 也提供了简便的控制方法。

还记得前两篇提到的 Linux capabilities 吗？Docker 可通过 `--cap-add` 和 `--cap-drop` 来控制相关的能力，相当于是给了常规的系统调用一个别名。

实践

这里我们通过实际的例子来体验下 seccomp 为我们带来的安全能力。

首先，需要写一个 seccomp profile 配置文件，它是一个 JSON 文件，你可以参考 Docker 项目中默认的 seccomp profile 文件。

禁止全部

前面我已经介绍过了 Docker seccomp 使用的是白名单模式，所以未例如配置文件中的系统调用均会被拒绝。我们可以使用如下的配置文件：

复制

```
{
    "defaultAction": "SCMP_ACT_ERRNO",
    "architectures": [
        "SCMP_ARCH_X86_64"
    ],
    "syscalls": []
}
```

这里未放行任何系统调用。使用此配置文件（deny.json）来启动一个容器：

复制

```
(MoeLove) ➜  ~ docker run --rm -it --security-opt seccomp=deny.json alpine sh
docker: Error response from daemon: OCI runtime start failed: cannot start a cor
```

你会发现是无法启动容器的。

这就是在本文开头为你介绍的 seccomp 的运行机制：如果运行未经放行的系统调用，则内核会杀掉它。这里我们使用的配置文件没有放行任何系统调用，这就导致了它始终无法正常启动。

不做任何限制

这里也有一个比较危险的操作方式，便是你可以在启动容器时，通过参数来设置不使用任何 seccomp profile，以此来放行所有的系统调用。

还记得在“容器篇”我为你介绍过 Docker 容器的原理吧，其中提到使用 unshare 来实现自己的容器。

我们来看看正常情况下容器中执行 unshare 的情况：

复制

```
(MoeLove) ➜  ~ docker run --rm -it alpine sh
/ # unshare
unshare: unshare(0x0): Operation not permitted
```

可以看到，默认是完全不允许执行的。

我们可以通过指定参数来跳过这个限制：

复制

```
(MoeLove) ➜  ~ docker run --rm -it --security-opt seccomp=unconfined alpine sh
/ # whoami
root
/ # unshare --user
21a7d31a75c9:~$ whoami
nobody
21a7d31a75c9:~$ exit
/ # whoami
root
```

通过指定 `seccomp=unconfined` 跳过了默认的限制。

总结

本篇，我为你介绍了 seccomp 相关的内容。在使用中，可以灵活的通过配置 seccomp profile 的放行策略来使用，以保证 Docker 容器的安全性。

这是“安全篇”的最后一篇，通过这个部分的学习，想必你已经发现了 Docker 的安全增强大多数都是基于 Linux 内核的。这和 Docker 的核心实现有关，所有的容器都共享同一内核。所以在使用时，也格外需要注意保障容器的安全性。

下一篇，我们将正式进入“网络篇”，一起探索 Docker 容器网络相关的内容，敬请期待。