Docker 安全 CheckList

> 版本：内部交流稿 v1.0
> 编撰：数字安全赋能群
> 时间：2020.12.1

介绍

Docker是最流行的容器化技术。一方面与直接在主机上运行应用程序相比，正确使用Docker可以提高安全级别。另一方面，一些常见的Docker错误配置又可能导致安全级别降低，甚至引入新的漏洞。所以应当根据现实情况进行选择和配置，以平衡业务与安全。

本 CheckList 的目的是提供易于使用的安全性建议列表，以帮助您保护Docker容器。

应当遵循的安全原则（Rules）

Rule 0 保持主机和Docker最新安全更新

目前已经公布的容器逃逸漏洞，会使攻击者直接获取 root 或 Administrator 权限，所以保持系统更新，修补Docker Engine和Docker Machine漏洞至关重要。

此外，容器与虚拟机不同，它与主机共享内核，因此在容器内执行的内核漏洞利用将直接攻击主机内核。例如，在完全独立的容器内执行的内核提权漏洞利用（例如Dirty COW 将导致对主机的 root 访问。

Rule 1 不要公开Docker守护进程套接字（甚至不公开给容器）

Docker socket `/var/run/docker.sock` 是 Docker 监听服务使用的 UNIX 套接字。这是 Docker API 的主要入口点。此套接字的所有者是root。向其它程序或用户授予此socket访问权限等同于向您的主机提供不受限制的root访问权限。

请勿启用 tcp Docker守护程序套接字。如果您使用 `-H tcp://0.0.0.0:XXX` 或类似方式运行docker守护程序，那么将暴露对Docker守护程序的访问，这一访问是未经加密和未经身份验证的直接访问。如果确实需要这样做，则应确保它安全。检查如何执行此操作遵循Docker官方文档.

请勿将 `/var/run/docker.sock` 暴露给其他容器。如果您使用 `-v /var/run/docker.sock://var/run/docker.sock` 或类似的文件运行docker镜像，则应该对其进行更改。注意，以只读方式安装套接字不是解决方案，这只会使其较为难以利用。

docker-compose文件中的类似内容如下所示：

```
volumes:
- "/var/run/docker.sock:/var/run/docker.sock"
```

Rule 2 设置用户

使用非特权用户（非root 或 非sudoer）配置容器，是防止特权升级攻击的最佳方法。可以通过以下三种不同方式来完成此操作：

1.在运行时，使用 `docker run` 命令的-u选项，例如：

```
docker run -u 4000 alpine
```

2.在构建期间。在Dockerfile中简单添加用户并使用它。例如：

```
FROM alpine
RUN groupadd -r myuser && useradd -r -g myuser myuser
<HERE DO WHAT YOU HAVE TO DO AS A ROOT USER LIKE INSTALLING PACKAGES ETC.>
USER myuser
```

3.启用 Docker 用户命名空间（Enable user namespace support）Docker daemon。

更多信息可参考Docker official documentation。

在kubernetes中，可以在 Security Context 使用 `runAsNonRoot` 字段进行配置。例如：

```
kind: ...
apiVersion: ...
metadata:
  name: ...
spec:
  ...
  containers:
  - name: ...
    image: ....
    securityContext:
          ...
          runAsNonRoot: true
          ...
```

作为 Kubernetes 集群管理员，您可以使用 Pod Security Policies 进行配置。

Rule 3 限制功能（仅授予容器所必需的功能）

Linux kernel capabilities 作为一种特权，应按最小特权原则进行授权使用，而Docker，在默认情况下仅需要使用其中一部分功能。

可以使用 `--cap-drop` 等选项来加固docker容器，或更改相应配置，更改或删除某些功能（如果需要添加某些功能，可以使用 `-cap-add` 选项）。

请注意：不要使用带有 `--privileged` 标志的容器，这会将所有Linux内核功能添加到容器中。

最安全的配置方法是，先使用 `--cap-drop all` 删除所有功能，然后再根据需要，仅添加所必需的功能。命令例如：

```
docker run --cap-drop all --cap-add CHOWN alpine
```

请注意：不要运行带有 `-privileged` flag的容器!

在 kubernetes 中，可以使用 `capabilities` 字段在 Security Context中进行配置，例如：

```
kind: ...
apiVersion: ...
metadata:
  name: ...
spec:
  ...
  containers:
  - name: ...
    image: ....
    securityContext:
          ...
          capabilities:
            drop:
              - all
            add:
              - CHOWN
          ...
```

对于 Kubernetes Cluster 管理员，可以参考Pod Security Policies对其进行配置。

Rule4 添加 –no-new-privileges 标志

应当始终使用 `--security-opt=no-new-privileges` 运行docker镜像，以防止使用 `setuid` 或 `setgid` 二进制文件非法提升权限。

在kubernetes中，可以使用 `allowPrivilegeEscalation` 字段在Security Context中进行配置，例如：

```
kind: ...
apiVersion: ...
metadata:
  name: ...
spec:
  ...
  containers:
  - name: ...
    image: ....
    securityContext:
          ...
          allowPrivilegeEscalation: false
          ...
```

Kubernetes 集群管理员可以参考 Pod Security Policies 文档对其进行配置。

Rule5 禁用容器间通信（--icc = false）

默认情况下，容器间通信（icc）是启用的。这意味着所有容器都可以使用 `docker0` bridged network实现彼此通信。

可以通过运行带有 `--icc = false` 标志的docker daemon来禁用它。

如果禁用了icc（icc = false），则需要使用 `--link=CONTAINER_NAME_or_ID:ALIAS` 选项告诉那些需要通信的容器。

请参阅Docker documentation - container communication中的更多内容

对于使用 Kubernetes 的情况，可以参考Network Policies 。

Rule 6 使用Linux安全模块（即seccomp，AppArmor或SELinux）

最重要的是：请不要禁用默认的安全配置文件！

考虑使用安全配置文件，例如 seccomp 或 AppArmor。

具体的操作指令，可以在下列文档中找到：

- Security Context documentation
- Kubernetes API documentation

Rule 7 限制资源（内存，CPU，文件描述符，进程，重新启动）

避免DoS攻击的最佳方法是限制资源。可以限制memory, CPU，最大重新启动次数（`--restart=on-failure:<number_of_restarts>`），最大文件描述符数量

（ `--ulimit nofile=<number>` ）和最大进程数（ `--ulimit nproc=<number>` ）。

更多内容可参考：Check documentation for more details about ulimits

在Kubernetes中也可执行此操作，具体参考：

- Assign Memory Resources to Containers and Pods
- Assign CPU Resources to Containers and Pods
- Assign Extended Resources to a Container

Rule 8 将文件系统和卷设置为只读

使用 `--read-only` 标志运行带有只读文件系统的容器。例如：

```
docker run --read-only alpine sh -c 'echo "whatever" > /tmp'
```

如果容器内的应用程序必须暂时保存某些内容，则将 `--read` 标志与 `--tmpfs` 结合使用，如下所示：

```
docker run --read-only --tmpfs /tmp alpine sh -c 'echo "whatever" > /tmp/file'
```

docker-compose文件中的等效项为：

```
version: "3"
services:
  alpine:
    image: alpine
    read_only: true
```

在kubernetes Security Context 中的等效项是：

```
kind: ...
apiVersion: ...
metadata:
  name: ...
spec:
  ...
  containers:
  - name: ...
    image: ....
    securityContext:
          ...
          readOnlyRootFilesystem: true
          ...
```

此外，如果某个卷（volume）仅为了读取而mount，那么应当**mount them as a read-only**。可以通过将： `ro` 附加到 `-v` 来完成，例如：

```
docker run -v volume-name:/path/in/container:ro alpine
```

或者使用 `--mount` 选项：

```
docker run --mount source=volume-name,destination=/path/in/container,readonly alpine
```

Rule 9 使用静态分析工具

有必要使用工具检测容器是否包含已知漏洞，例如可以使用静态分析工具扫描Images。

- 免费工具
  - Clair
  - Trivy
- 商用工具
  - Snyk **(open source and free option available)**
  - anchore **(open source and free option available)**
  - Aqua Security's MicroScanner **(free option available for rate-limited number of scans)**
  - JFrog XRay
  - Qualys

如果要检查Kubernets中的错误配置，可以使用:

- kubeaudit
- kubesec.io
- kube-bench

如果要检查Docker中的错误配置，可以使用：

- inspec.io
- dev-sec.io

Rule 10 将日志记录级别至少设置为 `INFO`

默认情况下，Docker守护程序配置为具有 `info` 的基本日志记录级别，如果不是，则需要将Docker守护程序日志级别设置为 `info` 。除非必要，不要将日志级别设置为 `debug` 。

要在docker-compose中配置日志级别，可以使用：

```
docker-compose --log-level info up
```

Rule 11 在构建时清理Dockerfile

遵循编写Dockerfile的一些最佳实践，可以避免许多问题。在构建pipeline时，将添加安全标签（security linter）作为其中步骤，可以避免很多麻烦。一些值得检查的问题是：

- Ensure a `USER` directive is specified
- Ensure the base image version is pinned
- Ensure the OS packages versions are pinned
- Avoid the use of `ADD` in favor of `COPY`
- Avoid the use of `apt/apk upgrade`
- Avoid curl bashing in `RUN` directives

参考文献：

- Docker Baselines on DevSec
- Use the Docker command line
- Overview of docker-compose CLI
- Configuring Logging Drivers
- View logs for a container or service
- Dockerfile Security Best Practices

相关项目

- OWASP Docker Top 10