容器镜像安全

本篇是第六部分“安全篇”的第一篇，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。本篇，我们将重点放在容器镜像安全上。

通过前面内容的学习，我们已经知道用 Docker 启动容器的一个必备前提是需要有镜像存在，无论该镜像是存储在本地还是从 registry 下载。

在我们启动容器时，对镜像内容本身是无法进行修改的，如果我们使用了被攻击的恶意镜像，那很可能会带来各种安全问题。

通常情况下，我们提到容器镜像安全，主要是指两个方面：镜像自身内容的安全和镜像分发安全。我分别从这两个方面为你介绍。

镜像自身内容安全

镜像中包含了很多内容，比如基本的 rootfs，此外如果是应用程序的镜像，还会包含应用程序的代码或者二进制文件等。我们先来看看如何检查镜像是否安全。

检查镜像安全

现在已经有很多工具可以帮我们完成这类事情了，比如很老牌的出自 CoreOS 的 Clair（现在属于 Quay 了），或是 Anchore家的 Anchore Engine 等。

这些工具大多都是通过分析镜像各层中的内容，并根据已有的漏洞数据库进行对比，从而判断其是否包含漏洞。

以上提到的这两个工具 Clair 和 Anchore Engine 的使用方法，可以参考其项目的文档。我这里为你介绍另一款目前比较活跃的后起之秀 Trivy，这个工具相比前面提到的两个工具而言，更简单也更适用于 CI 环节中。

安装过程很简单：

复制

```
(MoeLove) → cd /tmp
(MoeLove) → /tmp wget -q https://github.com/aquasecurity/trivy/releases/download
(MoeLove) → /tmp tar -zxf trivy_0.4.3_Linux-64bit.tar.gz
(MoeLove) → /tmp ./trivy --version
trivy version 0.4.3
```

接下来，我以 `alpine:3.11` 镜像为例：

复制

```
(MoeLove) ➜  /tmp ./trivy alpine:3.11
2020-01-29T18:21:16.381+0800    INFO    Need to update DB
2020-01-29T18:21:16.381+0800    INFO    Downloading DB...
14.71 MiB / 14.71 MiB [-------------------------------------------------------
2020-01-29T18:21:49.138+0800    INFO    Reopening DB...
2020-01-29T18:21:49.326+0800    INFO    Detecting Alpine vulnerabilities...

alpine:3.11 (alpine 3.11.0)
===========================
Total: 1 (UNKNOWN: 0, LOW: 0, MEDIUM: 1, HIGH: 0, CRITICAL: 0)

+---------+------------------+----------+-------------------+---------------+-----
| LIBRARY | VULNERABILITY ID | SEVERITY | INSTALLED VERSION | FIXED VERSION |
+---------+------------------+----------+-------------------+---------------+-----
| openssl | CVE-2019-1551    | MEDIUM   | 1.1.1d-r2         | 1.1.1d-r3     | ope
|         |                  |          |                   |               | RSAZ
|         |                  |          |                   |               | x86_
+---------+------------------+----------+-------------------+---------------+-----
```

可以看到，它报告了一个 MEDIUM 级别的问题，漏洞记录是 CVE-2019-1551，关于 OpenSSL，对应的修复版本和当前已安装版本也都列了出来。

处理办法只要升级 OpenSSL 即可。

注意：我这里所使用的 `alpine:3.11` 镜像其 SHA256 签名是

复制

```
sha256:c85b8f829d1f93a25fe91d2ce7dccf7ec723794873a77bb19623d38e926c788c
```

Docker 官方镜像会随时升级更新，并修复漏洞，如果你使用的不是此签名的镜像，得到的结果可能与我此处的不同。

另外，关于我推荐 Trivy 而不是其他工具的主要原因，在于它更简单，更利于和其他自动化工具相结合，关于它和一些主流 CI 工具如何集成，可以参考项目文档的 Continuous Integration 部分。

Trivy 和其他工具的对比，在项目文档的 Comparison with other scanners 部分也有说明，我就不再赘述了。

这里我的重点主要是为你介绍方法和思路，工具的选择可因人而异。

镜像分发安全

在前面《镜像构建和分发》一篇中，我为你介绍了几种构建和分发镜像的方式。这里我们将重点放在镜像分发安全上。

你在传输一般文件时，可能也有过类似的经历，比如因为网络原因导致下载的文件不完整；或是遭遇中间人的攻击导致文件被篡改、替换等。

Docker 镜像的分发其实也可能会遇到类似的问题，这就是此处我们要讨论的重点，也就是 Docker Content Trust（DCT）主要解决的问题。

Docker Content Trust 提供了对 Docker 镜像的签名和验证的能力，对于使用而言也就是 `docker trust` 命令所提供的相关功能。

注意：这需要 Docker CE 17.12 及以上版本。

复制

```
(MoeLove) ➜  ~ docker trust

Usage:  docker trust COMMAND

Manage trust on Docker images

Management Commands:
  key         Manage keys for signing Docker images
  signer      Manage entities who can sign Docker images

Commands:
  inspect     Return low-level information about keys and signatures
  revoke      Remove trust for an image
  sign        Sign an image

Run 'docker trust COMMAND --help' for more information on a command.
```

Docker 官方镜像均已启用 DCT 签名，我们来看看启用 DCT 后会有什么差别，可通过 DOCKER_CONTENT_TRUST 环境变量进行控制，如果设置为 0 则表示不启用：

复制

```
(MoeLove) ➜  ~ DOCKER_CONTENT_TRUST=0 docker pull alpine:3.11
3.11: Pulling from library/alpine
Digest: sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d
Status: Image is up to date for alpine:3.11
docker.io/library/alpine:3.11
```

如果设置为 1 则表示启用此功能：

复制

```
(MoeLove) ➜  ~ DOCKER_CONTENT_TRUST=1 docker pull alpine:3.11
Pull (1 of 1): alpine:3.11@sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79c
sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d: Pulling f
Digest: sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d
Status: Image is up to date for alpine@sha256:ab00606a42621fb68f2ed6ad3c88be54397
Tagging alpine@sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4
docker.io/library/alpine:3.11
```

最直观的感受是输出信息是不同的，当然在这个过程中另一个感受可能是感觉启用 DCT 之后 pull 镜像比之前慢了一些，这是因为首次使用时，需要先下载一些必要的信息，它们存储在 $HOME/.docker/trust 目录中。

复制

```
(MoeLove) ➜  ~ tree $HOME/.docker/trust
/home/tao/.docker/trust
├── private
└── tuf
    └── docker.io
        └── library
            └── alpine
                ├── changelist
                └── metadata
                    ├── root.json
                    ├── snapshot.json
                    ├── targets.json
                    └── timestamp.json

7 directories, 4 files
```

另外，我们也可以通过 `docker trust inspect` 命令查看此镜像的签名信息：

复制

```
(MoeLove) ➜  ~ docker trust inspect alpine:3.11
[
    {
        "Name": "alpine:3.11",
        "SignedTags": [
            {
                "SignedTag": "3.11",
                "Digest": "ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172
                "Signers": [
                    "Repo Admin"
                ]
            }
        ],
        "Signers": [],
        "AdministrativeKeys": [
            {
                "Name": "Root",
                "Keys": [
                    {
                        "ID": "a2489bcac7a79aa67b19b96c4a3bf0c675ffdf00c6d2fabe1a5
                    }
                ]
            },
            {
                "Name": "Repository",
                "Keys": [
                    {
                        "ID": "5a46c9aaa82ff150bb7305a2d17d0c521c2d784246807b2dc61
                    }
                ]
            }
        ]
    }
]
```

总结

本篇，我为你介绍了容器镜像安全相关的内容，主要包含镜像自身内容安全和镜像分发安全两部分的内容。

对于镜像自身内容安全，除了我在此处介绍的这种直接使用类似 Trivy 或者 Clair/Anchore Engine 这种方式外，你还可以通过你使用的镜像仓库来得到相关功能的支持。

比如 CNCF 托管的来自 VMware 的开源镜像仓库 Harbor，当前最新版本 v1.10 已经提供了镜像安全扫描的功能，以及插件化的镜像安全扫描支持。具体使用方式可参看其文档中漏洞扫描部分的内容。

另外，对于镜像安全分发 Harbor 也提供了与 Notary 的集成，可以方便地完成镜像签名校验等功能。具体使用方式可查看其文档中与 Notary 集成的部分。

下一篇，我将为你介绍 Docker 容器可用的安全策略。