Docker Image Security Best Practices

Authors:

@liran_tal
Node.js Security WG & Developer Advocate at Snyk

@omerlh
DevSecOps Engineer at Soluto by Asurion

https://snyk.io

1. Prefer minimal base images

In Snyk's State of open source security report 2019, we found each of the top ten docker images to include as many as 580 vulnerabilities in their system libraries.

- Choose images with fewer OS libraries and tools lower the risk and attack surface of the container
- Prefer alpine-based images over full-blown system OS images

2. Least privileged user

Create a dedicated user and group on the image, with minimal permissions to run the application; use the same user to run this process. For example, Node.js image which has a built-in `node` generic user:

```
FROM node:10-alpine
USER node
CMD node index.js
```

3. Sign and verify images to mitigate MITM attacks

We put a lot of trust into docker images. It is critical to make sure the image we're pulling is the one pushed by the publisher, and that no one has tampered with it.

- Sign your images with the help of Notary
- Verify the trust and authenticity of the images you pull

4. Find, fix and monitor for open source vulnerabilities

Scan your docker images for known vulnerabilities and integrate it as part of your continuous integration. Snyk is an open source tool that scans for security vulnerabilities in open source application libraries and docker images.

Use Snyk to scan a docker image:

```
$ snyk test --docker node:10 --file=path/to/
Dockerfile
```

Use Snyk to monitor and alert to newly disclosed vulnerabilities in a docker image:

```
$ snyk monitor --docker node:10
```

5. Don't leak sensitive information to docker images

It's easy to accidentally leak secrets, tokens, and keys into images when building them. To stay safe, follow these guidelines:

- Use multi-stage builds
- Use the Docker secrets feature to mount sensitive files without caching them (supported only from Docker 18.04).
- Use a `.dockerignore` file to avoid a hazardous `COPY` instruction, which pulls in sensitive files that are part of the build context

6. Use fixed tags for immutability

Docker image owners can push new versions to the same tags, which may result in inconsistent images during builds, and makes it hard to track if a vulnerability has been fixed. Prefer one of the following:

- A verbose image tag with which to pin both version and operating system, for example: `FROM node:8-alpine`
- An image hash to pin the exact contact, for example: `FROM node:<hash>`

7. Use COPY instead of ADD

Arbitrary URLs specified for ADD could result in MITM attacks, or sources of malicious data. In addition, ADD implicitly unpacks local archives which may not be expected and result in path traversal and Zip Slip vulnerabilities.

Use COPY, unless ADD is specifically required.

8. Use labels for metadata

Labels with metadata for images provide useful information for users. Include security details as well.

Use and communicate a Responsible Security Disclosure policy by adopting a SECURITY.TXT policy file and providing this information in your images labels.

9. Use multi-stage builds for small secure images

Use multi-stage builds in order to produce smaller and cleaner images, thus minimizing the attack surface for bundled docker image dependencies.

10. Use a linter

Enforce Dockerfile best practices automatically by using a static code analysis tool such as hadolint linter, that will detect and alert for issues found in a Dockerfile.