Secrets
ABAP
Apex
AzureResourceManager
C
C#
C++
CloudFormation
COBOL
CSS
Dart
Docker
Flex
Go
HTML
Java
JavaScript
JCL
Kotlin
Kubernetes
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

Docker static code analysis

Unique rules to find Vulnerabilities, Security Hotspots, and Code Smells in your DOCKER code

All rules 44 | Vulnerability 4 | Bug 4 | Security Hotspot 15 | Code Smell 21

Tags | Impact | Clean code attribute | Search by name...

Weak SSL/TLS protocols should not be used
Vulnerability

Disabling builder sandboxes is security-sensitive
Security Hotspot

Exposing administration services is security-sensitive
Security Hotspot

Recursively copying context directories is security-sensitive
Security Hotspot

Using clear-text protocols is security-sensitive
Security Hotspot

Using weak hashing algorithms is security-sensitive
Security Hotspot

Malformed JSON in Exec form leads to unexpected behavior
Bug

Dockerfile should only have one ENTRYPOINT and CMD instruction
Bug

Access variable which is not available in the current scope
Bug

A space before the equal sign in key-value pair may lead to unintended behavior
Bug

Allowing downgrades to a clear-text protocol is security-sensitive
Security Hotspot

Allowing shell scripts execution during package

Credentials should not be hard-coded

Analyze your code

Responsibility - Trustworthy | Security
Vulnerability | Blocker | cwe

Secret leaks often occur when a sensitive piece of authentication data is stored with the source code of an application. Considering the source code is intended to be deployed across multiple assets, including source code repositories or application hosting servers, the secrets might get exposed to an unintended audience.

Why is this an issue? | How can I fix it? | More Info

Best practices recommend using a secret vault for all secrets that must be accessed at container runtime. This will ensure the secret's security and prevent any further unexpected disclosure. Depending on the development platform and the leaked secret type, multiple solutions are currently available.

For all secrets that must be accessed at image build time, it is recommended to rely on Docker Buildkit's secret mount options. This will prevent secrets from being disclosed in image's metadata and build logs.

Additionally, investigations and remediation actions should be conducted to ensure the current and future security of the infrastructure.

Revoke the secret

Revoke any leaked secrets and remove them from the application source code.

Before revoking the secret, ensure that no other applications or processes are using it. Other usages of the secret will also be impacted when the secret is revoked.

Analyze recent secret use

When available, analyze authentication logs to identify any unintended or malicious use of the secret since its disclosure date. Doing this will allow determining if an attacker took advantage of the leaked secret and to what extent.

This operation should be part of a global incident response process.

Code examples

Noncompliant code example

The following code sample generates a new SSH private key that will be stored in the generated image. This key should be considered as compromised. Moreover, the SSH key encryption passphrase is also hardcoded.

```
FROM example

# Noncompliant
RUN ssh-keygen -N "passphrase" -t rsa -b 2048 -f /etc/ssh/rsa_key

RUN /example.sh --ssh /etc/ssh/rsa_key
```

The following code sample uses a seemingly hidden password which is actually leaked in the image metadata after the build.

```
FROM example
ARG PASSWORD

# Noncompliant
RUN wget --user=guest --password="$PASSWORD" https://example.com
```

Compliant solution

```
FROM example

RUN --mount=type=secret,id=ssh,target=/etc/ssh/rsa_key \
    /example.sh --ssh /etc/ssh/rsa_key
```

```
FROM example

RUN --mount=type=secret,id=wget,target=/home/user/.wgetrc \
    wget --user=guest https://example.com
```

For runtime secrets, best practices recommend relying on a vault service to pass secret information to the containers. Docker environment provides Swarm services that implement such a feature.

If such an option can not be considered, store the runtime secrets in an environment file such as `.env` and then start the container with the `--env-file` argument:

```
docker run --env-file .env myImage
```

It is then important to ensure that the environment files are securely stored and generated.

Available In:
sonarlint | sonarcloud | sonarqube