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...

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 installation is security-sensitive
Security Hotspot

Using host operating system namespaces is security-sensitive
Security Hotspot

Setting loose POSIX file permissions is security-sensitive
Security Hotspot

Reduce the amount of consecutive RUN instructions
Code Smell

Prefer COPY over ADD for copying local resources
Code Smell

WORKDIR instruction should only be used with absolute

Using weak hashing algorithms is security-sensitive

Analyze your code

Responsibility - Trustworthy | Security

Security Hotspot | Critical | cwe

Cryptographic hash algorithms such as `MD2`, `MD4`, `MD5`, `MD6`, `HAVAL-128`, `HMAC-MD5`, `DSA` (which uses `SHA-1`), `RIPEMD`, `RIPEMD-128`, `RIPEMD-160`, `HMACRIPEMD160` and `SHA-1` are no longer considered secure, because it is possible to have `collisions` (little computational effort is enough to find two or more different inputs that produce the same hash).

Ask Yourself Whether

The hashed value is used in a security context like:

- User-password storage.
- Security token generation (used to confirm e-mail when registering on a website, reset password, etc ...).
- To compute some message integrity.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

Safer alternatives, such as `SHA-256`, `SHA-512`, `SHA-3` are recommended, and for password hashing, it's even better to use algorithms that do not compute too "quickly", like `bcrypt`, `scrypt`, `argon2` or `pbkdf2` because it slows down `brute force attacks`.

Sensitive Code Example

```
FROM ubuntu:22.04

# Sensitive
RUN echo "a40216e7c028e7d77f1aec22d2bbd5f9a357016f  go1.20.linux-amd64.tar.gz" | sha1sum -c
RUN tar -C /usr/local -xzf go1.20.linux-amd64.tar.gz
ENV PATH="$PATH:/usr/local/go/bin"
```

Compliant Solution

```
FROM ubuntu:22.04

RUN echo "5a9ebcc65c1cce56e0d2dc616aff4c4cedcfbda8cc6f0288cc08cda3b18dcbf1  go1.20.linux-amd64.tar.gz" | sha256sum -c
RUN tar -C /usr/local -xzf go1.20.linux-amd64.tar.gz
ENV PATH="$PATH:/usr/local/go/bin"
```

See

- OWASP - Top 10 2021 Category A2 - Cryptographic Failures
- OWASP - Top 10 2017 Category A3 - Sensitive Data Exposure
- OWASP - Top 10 2017 Category A6 - Security Misconfiguration
- OWASP - Mobile AppSec Verification Standard - Cryptography Requirements
- OWASP - Mobile Top 10 2016 Category M5 - Insufficient Cryptography
- CWE - CWE-1240 - Use of a Risky Cryptographic Primitive

Available In:

sonarlint | sonarcloud | sonarqube