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 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 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

Using clear-text protocols is security-sensitive

Analyze your code

Intentionality - Complete | Security

Security Hotspot | Critical | dockerfile cwe

Clear-text protocols such as `ftp`, `telnet`, or `http` lack encryption of transported data, as well as the capability to build an authenticated connection. It means that an attacker able to sniff traffic from the network can read, modify, or corrupt the transported content. These protocols are not secure as they expose applications to an extensive range of risks:

- sensitive data exposure
- traffic redirected to a malicious endpoint
- malware-infected software update or installer
- execution of client-side code
- corruption of critical information

Even in the context of isolated networks like offline environments or segmented cloud environments, the insider threat exists. Thus, attacks involving communications being sniffed or tampered with can still happen.

For example, attackers could successfully compromise prior security layers by:

- bypassing isolation mechanisms
- compromising a component of the network
- getting the credentials of an internal IAM account (either from a service account or an actual person)

In such cases, encrypting communications would decrease the chances of attackers to successfully leak data or steal credentials from other network components. By layering various security practices (segmentation and encryption, for example), the application will follow the *defense-in-depth* principle.

Note that using the `http` protocol is being deprecated by major web browsers.

In the past, it has led to the following vulnerabilities:

- CVE-2019-6169
- CVE-2019-12327
- CVE-2019-11065

Ask Yourself Whether

- Application data needs to be protected against tampering or leaks when transiting over the network.
- Application data transits over an untrusted network.
- Compliance rules require the service to encrypt data in transit.
- OS-level protections against clear-text traffic are deactivated.

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

Recommended Secure Coding Practices

- Make application data transit over a secure, authenticated and encrypted protocol like TLS or SSH. Here are a few alternatives to the most common clear-text protocols:
  - Use `sftp`, `scp`, or `ftps` instead of `ftp`.
  - Use `https` instead of `http`.

It is recommended to secure all transport channels, even on local networks, as it can take a single non-secure connection to compromise an entire application or system.

Sensitive Code Example

```
RUN curl http://www.example.com/
```

Compliant Solution

```
RUN curl https://www.example.com/
```

See

Documentation

- AWS Documentation - Listeners for your Application Load Balancers
- AWS Documentation - Stream Encryption

Articles & blog posts

- Google - Moving towards more secure web
- Mozilla - Deprecating non secure http

Standards

- CWE - CWE-200 - Exposure of Sensitive Information to an Unauthorized Actor
- CWE - CWE-319 - Cleartext Transmission of Sensitive Information
- STIG Viewer - Application Security and Development: V-222397 - The application must implement cryptographic mechanisms to protect the integrity of remote access sessions.
- STIG Viewer - Application Security and Development: V-222534 - Service-Oriented Applications handling non-releasable data must authenticate endpoint devices via mutual SSL/TLS.
- STIG Viewer - Application Security and Development: V-222562 - Applications used for non-local maintenance must implement cryptographic mechanisms to protect the integrity of maintenance and diagnostic communications.
- STIG Viewer - Application Security and Development: V-222563 - Applications used for non-local maintenance must implement cryptographic mechanisms to protect the confidentiality of maintenance and diagnostic communications.
- STIG Viewer - Application Security and Development: V-222577 - The application must not expose session IDs.
- STIG Viewer - Application Security and Development: V-222596 - The application must protect the confidentiality and integrity of transmitted information.
- STIG Viewer - Application Security and Development: V-222597 - The application must implement cryptographic mechanisms to prevent unauthorized disclosure of information and/or detect changes to information during transmission.
- STIG Viewer - Application Security and Development: V-222598 - The application must maintain the confidentiality and integrity of information during preparation for transmission.
- STIG Viewer - Application Security and Development: V-222599 - The application must maintain the confidentiality and integrity of information during reception.

Available In:

sonarlint | sonarcloud | sonarqube