Grype Vulnerability scanner

START HERE

`grype` *<image>*
Scan a container image

`grype dir:`*<path>*
Scan a project folder

`grype sbom:`*<file>*
Scan an existing SBOM

syft *<image>* **-o json | grype**
Pipe an SBOM in

--fail-on high
Exit 2 on serious findings

--only-fixed
Hide CVEs without a fix

-o json
Machine-readable report

`grype db update`
Refresh the vuln database

Source image, dir, SBOM, PURL → `catalog` → **Packages** OS and language deps → `match` → **Vuln DB** local, auto-updated → `filter` → **Report** table, json, sarif, cyclonedx

CONCEPTS

Grype matches the packages it finds (or reads from an SBOM) against a local vulnerability DB. It reports; it does not generate SBOMs (that is Syft).

A bare argument is auto-detected (local daemon, then registry, then path). Add a scheme prefix or --from when the guess is wrong.

The DB is downloaded on first scan and re-checked on later ones; offline or CI runs need it pre-fetched with db update or db import.

SCAN SOURCES

`grype` *<image>*
Scan an image (name:tag)

`grype dir:`*<path>*
Scan a directory

`grype sbom:`*<file>*
Scan a Syft/SPDX/CycloneDX SBOM

cat *<sbom.json>* **| grype**
Read an SBOM from stdin

`grype docker:`*<image>*
Force the local Docker daemon

`grype registry:`*<image>*
Force a pull from the registry

`grype podman:`*<image>*
Use the Podman daemon

SCAN SOURCES (cont.)

`grype docker-archive:`*<file.tar>*
Scan a docker save tarball

`grype oci-archive:`*<file.tar>*
Scan an OCI archive

`grype oci-dir:`*<dir>*
Scan an OCI layout directory

`grype file:`*<path>*
Scan a single file

`grype purl:`*<file>*
Scan a file of package URLs

`grype pkg:`*<purl>*
Check one package URL
Example: pkg:apk/openssl@3.2.1?distro=alpine-3.20.3 (give distro for OS packages).

--platform *<os/arch>*
Pick the image platform (linux/arm64)

--scope all-layers
Include layers hidden by later ones
Default is squashed; deep-squashed also looks at more of the final image.

--exclude *<glob>*
Skip matching paths (repeatable)

FILTERING

--only-fixed
Show only vulns with a fix available

FILTERING (cont.)

--ignore-states *<list>*
Drop fix states: fixed,not-fixed,unknown,wont-fix

--vex *<file>*
Apply a VEX document (repeatable)

--by-cve
Report by CVE ID, not GHSA ID

--sort-by *<key>*
Order: risk, severity, epss, kev, package
Default is risk.

--ignore-states wont-fix
Skip issues the vendor will not fix

`grype` **explain --id** *<CVE>*
Explain why a CVE matched

OUTPUT AND CI

--fail-on *<severity>*
Exit code 2 at or above this severity
Levels: negligible, low, medium, high, critical. Without it the exit code stays 0.

-o table
Human-readable table (default)

-o json
Full JSON report

-o sarif
SARIF for code-scanning tools

OUTPUT AND CI (cont.)

-o cyclonedx-json
CycloneDX report with vulns

-o template -t *<file>*
Render a Go template

-o json=<file>
Write one format to a file
Repeat -o to emit several formats in one run.

--file *<path>*
Write the report to a file

-q
Quiet: no log output

-v / -vv
Info / debug logging

`grype` **--version**
Print the version

IGNORE RULES AND CONFIG

.grype.yaml
Config file in the project root
Also read from .grype/config.yaml, ~/.grype.yaml and $XDG_CONFIG_HOME/grype/config.yaml.

-c *<file>*
Use a specific config file

ignore: - vulnerability: *<CVE>*
Suppress one CVE

ignore: fix-state: not-fixed
Suppress by fix state

IGNORE RULES AND CONFIG (cont.)

ignore: package: name: *<regex>*
Suppress by package name

ignore: package: location: *<glob>*
Suppress by file location

ignore: reason: *<text>*
Record why it is ignored
Note only; it does not affect matching.

fail-on-severity: high
Config form of --fail-on

GRYPE_<*KEY*>**=**<*value*>
Set any option via environment
Nested keys join with underscores: GRYPE_DB_AUTO_UPDATE=false.

DATABASE

`grype db update`
Download the latest database

`grype db check`
Is a newer database available

`grype db status`
Show the cached DB's age and validity

`grype db import` *<file|url>*
Load a DB archive (air-gapped)

`grype db search --vuln` *<ID>*
Look up a vulnerability in the DB

DATABASE (cont.)

`grype db search --pkg` *<name>*
Find vulns for a package, CPE or PURL

`grype db delete`
Remove the cached DB

GRYPE_DB_AUTO_UPDATE=false
Do not fetch updates during a scan

GRYPE_DB_VALIDATE_AGE=false
Allow a DB older than 120h

RECIPES

Gate CI on fixable high-severity issues

```
grype <image> --only-fixed --fail-on high
```

Scan once, report as SARIF and JSON

```
grype <image> -o sarif=grype.sarif -o json=grype.json
```

SBOM first, then re-scan it as the DB changes

```
syft <image> -o json=sbom.json
grype db update
grype sbom:sbom.json --fail-on critical
```

Air-gapped scan with a transferred DB

```
# online host
grype db list
# offline host
grype db import <db.tar.zst>
GRYPE_DB_AUTO_UPDATE=false
grype <image>
```