sbom-cve-check: Lightweight Python tooling for out-of-build CVE analysis of SPDX3 SBOMs

Benjamin Robin *benjamin.robin@bootlin.com*
Olivier Benjamin *olivier.benjamin@bootlin.com*

Yocto Project Summit 2025.12

Presenters

- Embedded Linux engineer & Security engineer at **Bootlin**
  - Development, consulting and training about **embedded Linux**
  - Open-source focus
- Bootloaders, kernel development, Buildroot and Yocto integration
- Open-source contributor
- Living in **Lyon**, France

sbom-cve-check

What is it?

- A **standalone** python tool, with as few dependencies as we could manage
- Takes as input:
  - an SBOM (currently SPDX 2.2 and SPDX 3.0 formats are supported)
  - (optional) the json VEX manifest generated by Yocto's `vex.bbclass` in upstream oe-core since Yocto 5.2.4 "Walnascar", also in Styhead
  - some enrichment databases
- Looks up which CVEs impact the SBOM, and therefore potentially the systems
- Outputs an assessment in a format that can be fed to downstream tools
- Overall, close to Yocto's `cve-check.bbclass`

Why is it?

- Not being aware of which CVEs affect one's stuff is going to get expensive soon
- `cve-check` is great, but runs **during a build**
- This is inconvenienent because:
  - CVE checking already takes a while, no need to add build time to it
  - once the software is shipped, building an old version might be unnecessary effort
  - what if we wanted to check artifacts of a build we did not/cannot run?
- Easy
  - few dependencies
  - easy to install and use
  - no bells, no whistles

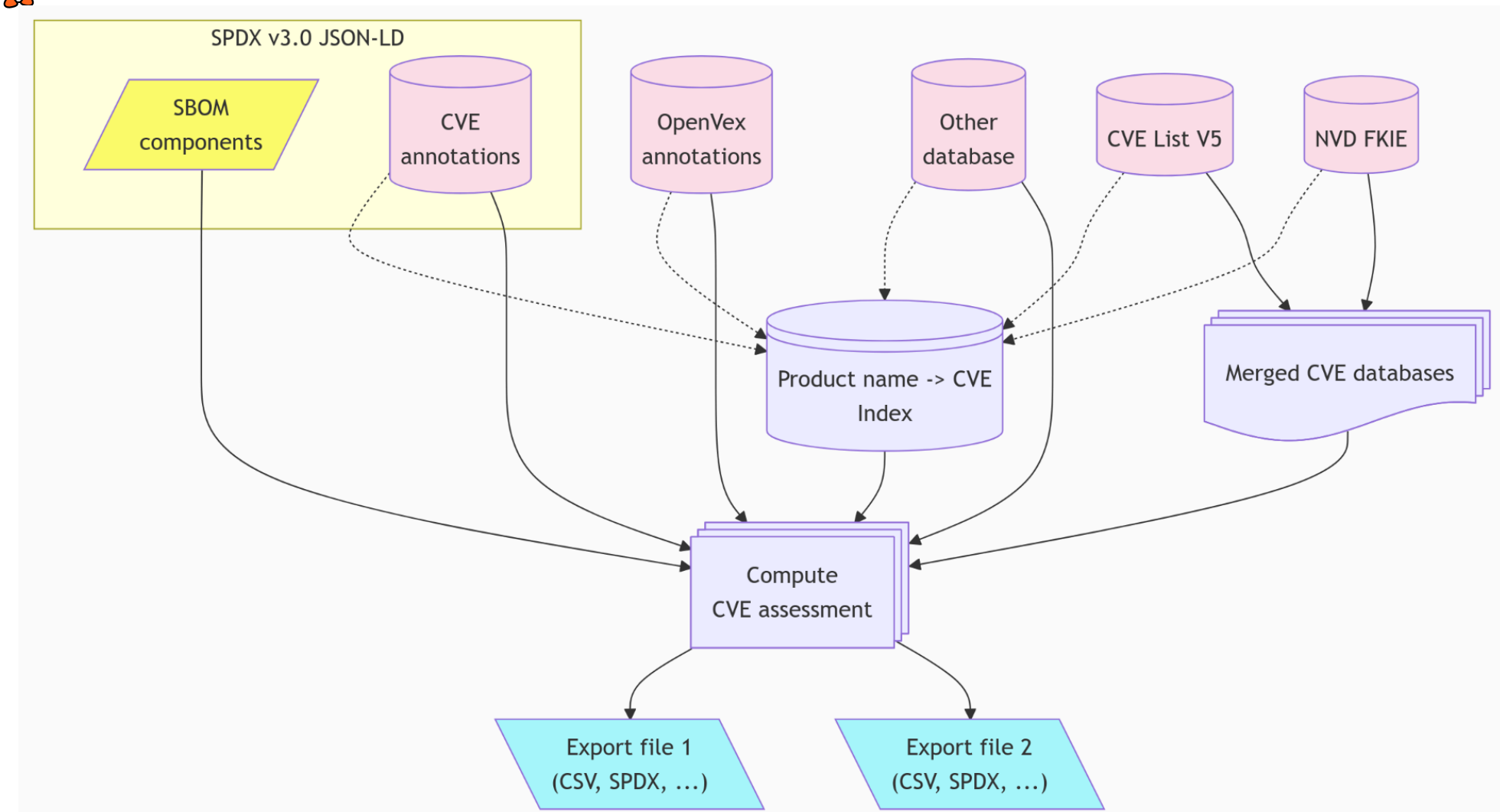
Dependencies

```
dependencies = ["spdx_python_model",]

[project.optional-dependencies]
extra = ["PyYAML>=6.0", "argcomplete>=3.6",]

[dependency-groups]
docs = [
    "myst-parser==4.0.1",
    "sphinx-rtd-theme==3.0.2",
    "sphinxcontrib-mermaid==1.0.0",
]
test = ["pytest>=8.0.0", "pytest-cov>=7.0.0",]
lint = ["ruff>=0.14", "mypy>=1.18",]
```

Input & outputs formats

- The tool pulls from several CVE databases:
  - **NVD**, from `github.com/fkie-cad/nvd-json-data-feeds`
  - **CVE List**, from `github.com/CVEProject/cvelistV5`
- It supports multiple annotation formats:
  - OpenVEX
  - Yocto's custom format, generated from `vex.bbclass`
  - A simple annotations format stored in YAML files.
  - Annotations provided in SPDX 3.0 SBOM file.
- It supports the following export formats:
  - SPDX3: Only if the input SBOM is also an SPDX 3.0 file (for now).
  - CSV
  - Yocto's cve-check output format.

Overall design



Design concepts

sbom-cve-check is designed around 3 types of concepts:

- An input SBOM file
- Databases, which includes:
  - CVE databases, e.g. NVD,
  - Annotation databases
- Output (export) files

Each of these types or formats is implemented by a builtin plugin.

Plugins

sbom-cve-check can be extended by using custom (external) plugins.

This allows to add a new type or format that is not currently supported natively by the tool. For example:

- A new type of SBOM format
- A custom CVE annotation format
- A custom report (export) format (Web page, PDF, ...)

Ignore CVE from compiled sources

Mark a CVE automatically as ignored if affected sources are not compiled.

- Require to set `SPDX_INCLUDE_COMPILED_SOURCES` to `1` in each recipe that need this extra processing.
- The CVE database needs to provide the list of affected files: Most of the Linux kernel CVEs have this information.

Example of command line

```
sbom-cve-check \
  --sbom-path core-image-minimal-qemuarm.rootfs.spdx.json \
  --yocto-vex-manifest core-image-minimal-qemuarm.rootfs.json \
  --export-filter-vulnerable \
  --export-type yocto-cve-check-manifest --export-path out.json
```

- In blue: The input SBOM
- In orange: An annotation file: A Yocto VEX manifest
- In purple: The export configuration flags

Example of configuration file

my_annotations.toml:

```
[databases.my-annotations]
type = "simple-annotations"
git_url = "git@github.com:my-project/my-annotations.git"
path = "${SBOM_CHECK_DATABASES_DIR}/my-annotations"
auto_update_max_age = "10min"
globs = ["common", "kernel-6.12", "boards/imx8mp-lpddr4-evk"]
```

```
sbom-cve-check \
  --sbom-path core-image-minimal-qemuarm.rootfs.spdx.json \
  --config path/to/my_annotations.toml \
  --export-type spdx3 --export-path out.spdx.json
```

Performance

sbom-cve-check executes several steps:

- Downloading/updating the databases.
- Indexing these databases, and store the result into a cache file. This step takes about 1 minute in case of a cache miss.
- Performing the analysis, which takes anywhere from a few tens of seconds to 2 minutes depending on the complexity of the SBOM and the output format.

Sources & documentation

- The source code is available here: `github.com/bootlin/sbom-cve-check`
  - The source code is under GPLv2
  - Contributions are of course welcome :)
- The documentation can be consulted from: `sbom-cve-check.readthedocs.io`

Roadmap & plan for future improvements

- Add support of Ubuntu CVE tracker repository
  - Automatically detect if a patch was backported
- Add more export formats, like for example OpenVEX.
- Add CycloneDX (CDX) SBOM support as input.
- Allow to generate an SBOM (CDX or SPDX 3.0) as output even if the SBOM specified as input is in another format.

Want to try it out?

- Install the tool:
  - Create a Python virtual environment (recommended)
  - `pip install sbom-cve-check[extra]`
- Generate artifacts using Yocto (or use existing ones):
  - SPDX v3.0 is generated by default since Yocto Walnascar (5.2).
  - Add `INHERIT += "vex"` in `local.conf`.
- Retrieve these artifacts from Yocto `deploy` directory:
  - `${IMAGE_NAME}.rootfs.spdx.json`: The SPDX v3.0 SBOM file.
  - `${IMAGE_NAME}.rootfs.json`: File generated by the `vex.bbclass`.
- Execute:

```
sbom-cve-check --sbom-path ${IMAGE_NAME}.rootfs.spdx.json \
  --yocto-vex-manifest ${IMAGE_NAME}.rootfs.json \
  --export-type yocto-cve-check-manifest --export-path out.json
```

Thank you!

Questions?

Benjamin Robin *benjamin.robin@bootlin.com*
Olivier Benjamin *olivier.benjamin@bootlin.com*

https://bootlin.com/pub/conferences/