How to prove the safety of your software conf42.com Python talk

Introduction

- Hello! I am:
  - 👱 Marco Verleun (marco.verleun@i-share.nl)
  - 🏢 Employed by i-share (www.i-share.nl)
  - 👷 Devops/GitOps/Cloud/Container/Cluster/Linux engineer (Pick one)
  - 🎯 Passion: Secure K8S clusters (air-gapped) running secure containers

Short agenda

During this talk I want to share with you how to reveal the safety of your code without revealing the application logic.

And I hope to create a bit more awareness about the environment in which your application will be running.

Let's have a look at a different industry with similar challenges...


SANTA CRUZ
MONROVIA
HAMBURG SÜD

From code to production

Previous part | Back to table of contents | Next part

From code to production

It starts with the code…

From code to production

It starts with the code…

Which gets deployed inside/on:

- An appliance
- A host
- A container

Or is reused as an module.

A (random) app step by step: worker.py

Let's explore what happens to an app (written by Jerome Petazzo) which will run inside a container. The source code can be found here: worker.py

We'll see that the number of CVE's will increase as the code moves on during the build process from app to container image.

We will not focus on the application logic, only on the safety.

We want to assure our customers/users/ops collegues that the code does not contain any critical CVE's.

How can we do this?

How is this done in the food industry?

Let's first look at how the food industry is doing this.


한진 보스톤
HANJIN BOSTON
MAJURO

Food safety

Previous part | Back to table of contents | Next part

Food safety

Would you consume this?

Or this?


FOOD ALLERGY WARNING
Please be advised that our food may have come in contact or contain peanuts, tree nuts, soy, milk, eggs, wheat, shellfish or fish.
MILK
Please ask a staff member about the ingredients used in your meal before ordering. Thank you — Management.

It probably depends…

- A blank container is for the adventurous amongst us and might be delicious.
- Other people might be more interested in nutritious facts.

It is nice to know what's inside

It is nice to know what the contents of a product are before you decide if you want to consume it.

Food labels are ment to do this without revealing a recipe.

Have a look at this

Why not do the same with our

- Hardware
- Software
- Saas solutions
- etc.

...BOMs are there to help

Have a look at https://github.com/CycloneDX/bom-examples

We focus on SBOM during this talk.

Why use SBOMs?

Previous part | Back to table of contents | Next part

Why use SBOMs?

Do you want to be in control of your software?

Let's see some reasons why you want to use SBOM's.

Did you see this?

cURL Vulnerability CVE-2023-38545 for Python Systems

October 10, 2023 • 4 mins

A high-severity vulnerability in cURL and its associated library libcurl was disclosed on 11 October, 2023, with widespread impact likely. This post examines the vulnerability, impacted Python packages, and recommended actions. This article will be updated as new information becomes available throughout the coming hours and days.

Was your app affected?

If so:

- How did you know?
- How long took it to figure it out?

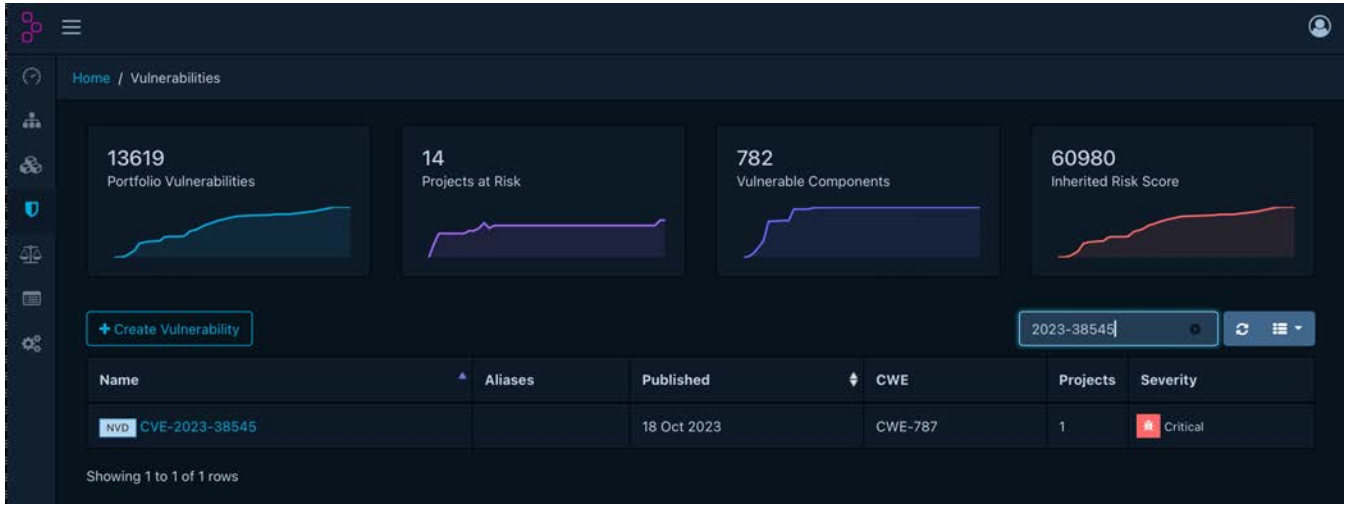
Was your app affected?

If so:

- How did you know?
- How long took it to figure it out?

SBOM's are extremly useful in these situations...

Look at this:

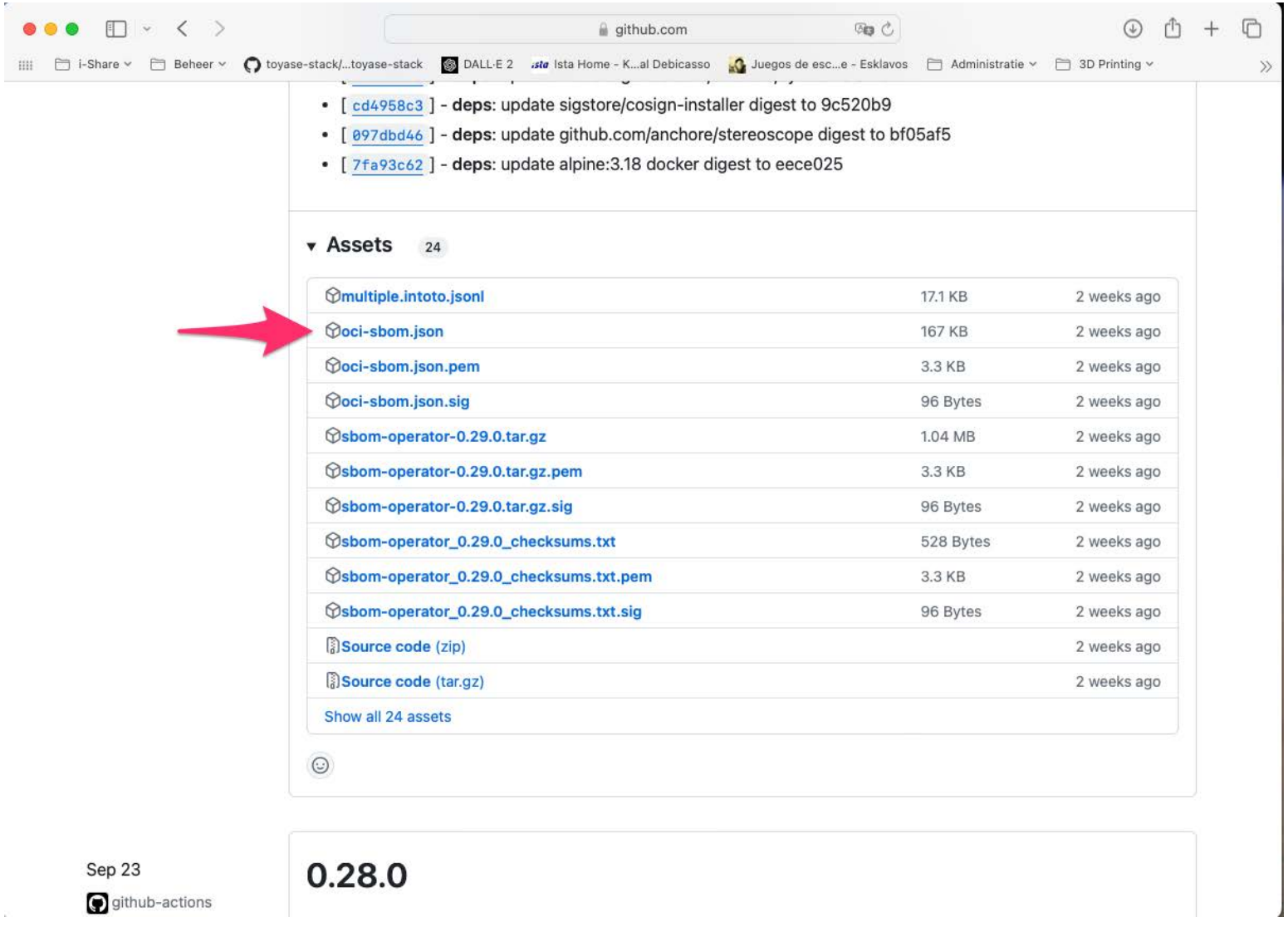


Like food labels SBOMs tell you what's inside

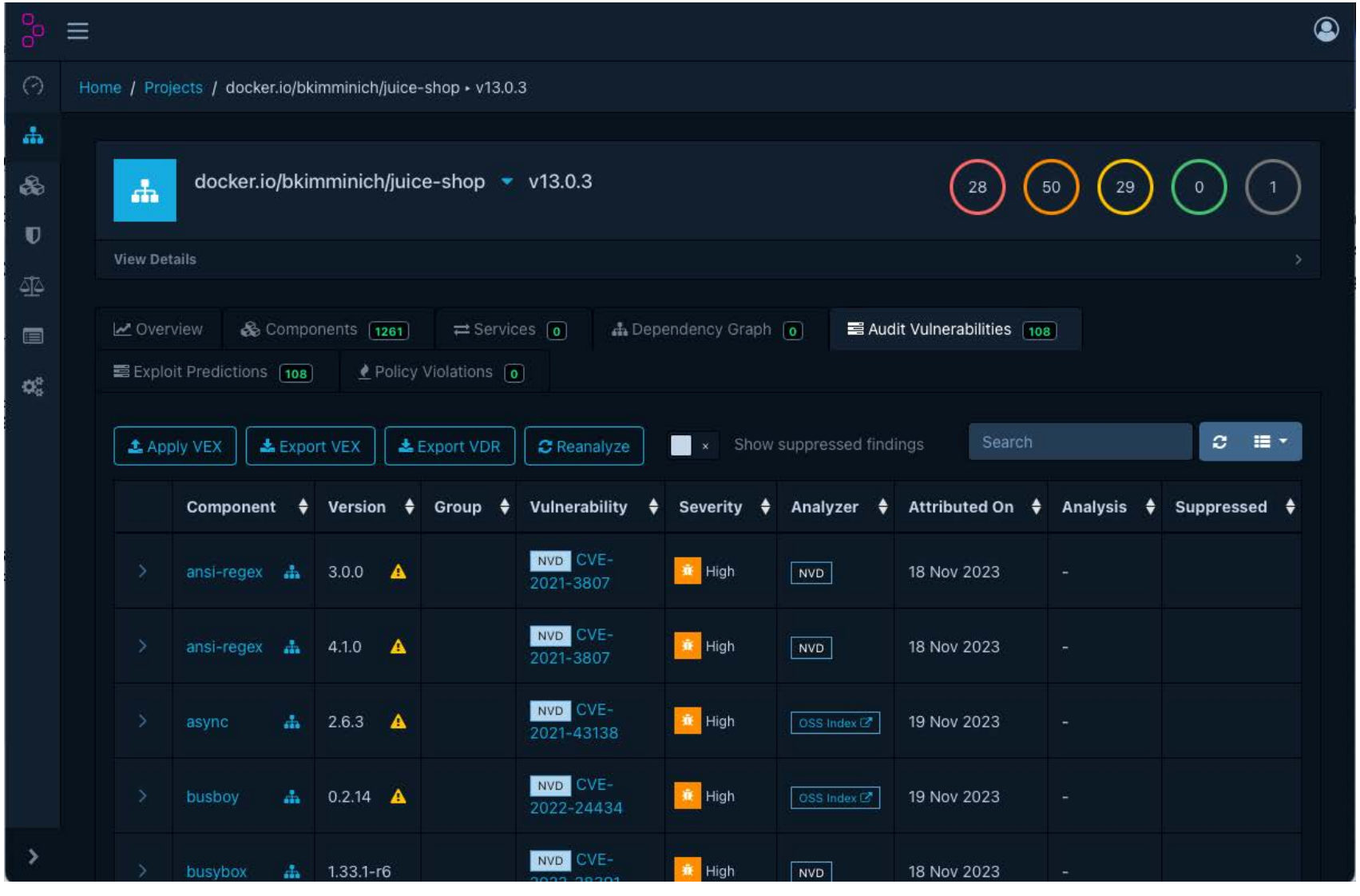
Example SBOM snippet


```
{
      "id": "e1597ba21775e886",
      "name": "certifi",
      "version": "2022.12.7",
      "type": "python",
      "foundBy": "python-package-cataloger",
      "locations": [
        {
          "path": "/usr/local/lib/python3.11/site-packages/certifi-2022.12.7.dist-info
          "layerID": "sha256:2ecbe4cb3d052a933e1cab8d573b14cfb4c50df323e4efde9cece026c
          "annotations": {
            "evidence": "primary"
    ...
      "licenses": [
        {
          "value": "MPL-2.0",
          "spdxExpression": "MPL-2.0",
          "type": "declared",
```

More and more you can download them upfront



And analyze them before you install something




EMAU 3039937
EMAU 3027407
EMAU 3047883
EMAU 3027629
EMAU 3042741
EMAU 3031104
EMAU 3028340
EMAU 3013111
EMAU 3049628
EMAU 3036981
EMAU 3019350
EMAU 3019752
EUR 075
EMAU 3018355
EMAU 3054541
EMAU 3042455
EMAU 3012400
EMAU 3045570
EMAU 3038433
EMAU 1016093 0
EMAU 1016097
EMAU 1017914
EMAU 1019350
EMAU 1019752
22G1
SNTM
CMAU
© Lode Van de Velde

Back to our app: worker.py

Previous part | Back to table of contents | Next part

Back to our app: worker.py

Let's follow our app from code to deployment in a container.

During each step we'll analyze the software and show some highlights.

Step 1: The code

```
import logging
import os
from redis import Redis
import requests
import time
...
redis = Redis("redis")
...
def work_loop(interval=1):
    deadline = 0
    loops_done = 0
    while True:
        if time.time() > deadline:
            log.info("{} units of work done, updating hash counter"
                     .format(loops_done))
            redis.incrby("hashes", loops_done)
            loops_done = 0
            deadline = time.time() + interval
        work_once()
        loops_done += 1
```

Shipping the app.

We will distribute our app in a docker container.

The following base images are used to see which would be the best image for our app. We have already determined that the app will run fine with all the mentioned images:

- python:alpine
- python:3.9.18-slim
- python:latest

Building the container images

The container build is done with a small `Dockerfile`. The only thing that changes is the `FROM` line where different base images are specified:

```
FROM python:latest
RUN pip install redis
RUN pip install requests
COPY worker.py /
CMD ["python", "worker.py"]
```

```
FROM python:alpine
RUN pip install redis
RUN pip install requests
COPY worker.py /
CMD ["python", "worker.py"]
```

Build result

The build result is as follows:

```
docker image ls
REPOSITORY  TAG             IMAGE ID       CREATED              SIZE
worker      3.9.18-slim     47f85c518f2d   7 seconds ago        237MB
worker      alpine          890af8c86632   About a minute ago   110MB
worker      latest          9011701a671b   3 minutes ago        1.49GB
```

SBOM creation

SBOM's are create from the source code and the images for further analyses. The tool used is `syft`, but it could have been another tool as well.

Analysis is done with `grype` because it produces output that fits nice in this presentation.

Let's see how each step adds vulnerabilities. Note that the number of reported CVE's was correct at the time of writing. Quite likely more CVE's have been discoverd since then.

Source code analysis:

The source code is quite clean. Only one CVE is reported:

```
grype --add-cpes-if-none sbom-worker.py.json
 ✔ Vulnerability DB                [no update available]
 ✔ Scanned for vulnerabilities     [1 vulnerability matches]
   ├── by severity: 0 critical, 0 high, 1 medium, 0 low, 0 negligible
   └── by status:   1 fixed, 0 not-fixed, 0 ignored
```

Our first image based on python:latest

This is the most tempting image. It seems to be very complete, but maybe it contains to much?

```
grype --add-cpes-if-none sbom-python-latest.json
 ✔ Vulnerability DB                [no update available]
 ✔ Scanned for vulnerabilities     [1700 vulnerability matches]
   ├── by severity: 21 critical, 359 high, 519 medium, 73 low, 721 negligible (7 unkn
   └── by status:   448 fixed, 1252 not-fixed, 0 ignored
```

Wow... We went from only 1 CVE to 1700...

Can we do better: python:3.9.18-slim

A slim image with more than enough to run our application, but much less than python:default.

```
grype --add-cpes-if-none sbom-python-3.9.18-slim.json
 ✔ Vulnerability DB                [no update available]
 ✔ Scanned for vulnerabilities     [101 vulnerability matches]
   ├── by severity: 1 critical, 11 high, 28 medium, 3 low, 55 negligible (3 unknown)
   └── by status:   14 fixed, 87 not-fixed, 0 ignored
```

That's already a huge difference. Especially when you pay attention to the critical and high rated CVE's

Let's try one more image: python:alpine

```
grype --add-cpes-if-none sbom-python-alpine.json
 ✔ Vulnerability DB                [no update available]
 ✔ Scanned for vulnerabilities     [21 vulnerability matches]
   ├── by severity: 0 critical, 1 high, 18 medium, 0 low, 0 negligible (2 unknown)
   └── by status:   9 fixed, 12 not-fixed, 0 ignored
```

Summary

The scores are shown in the table below.

Source	Critial	High	Medium	Low
worker.py	0	0	1	0
python-latest	21	359	519	73 low
python:3.9.18-slim	1	11	28	3 low
python:alpine	0	1	18	0 low

Any idea which image I prefer to deploy?

Storing SBOM files

If you store these SBOM's files you can quickly evaluate if new CVE's are introduced without scanning every component or image again.

Or you can store them in a database like `Dependency Track` which will periodically evaluate the vulnerabilities and, if configured, send you notifications when your attention is required.

Distributing SBOM files

The federal US government expects vendors to provide SBOM files prior to purchasing software or appliances. And they are not alone.

On `github.com` you'll see them appear as well, waiting for you to download them.

Even the new standard for container registries allows you to store SBOM information. The `docker buildx` command can do this as well.

Final words

When working with SBOM tools make sure you're using good ones. When in doubt compare tools and see if the meet your needs. Some are good for source code, others can only identify os components and some can do both. Not all are equally good...

Interesting links

Previous part | Back to table of contents | Next part

Interesting links

By far not complete, but check this out:

Generating SBOMs

https://github.com/kubernetes-sigs/bom

https://github.com/anchore/syft

https://docs.docker.com/engine/sbom/

https://github.com/ckotzbauer/sbom-operator

https://github.com/microsoft/sbom-tool

Storing SBOMs

https://github.com/dlorenc/sbom-oci

https://docs.docker.com/build/attestations/sbom/

https://dependencytrack.org

Analyzing SBOMs

https://dependencytrack.org

https://trivy.dev

https://github.com/openclarity/kubeclarity

https://github.com/CycloneDX/bom-examples