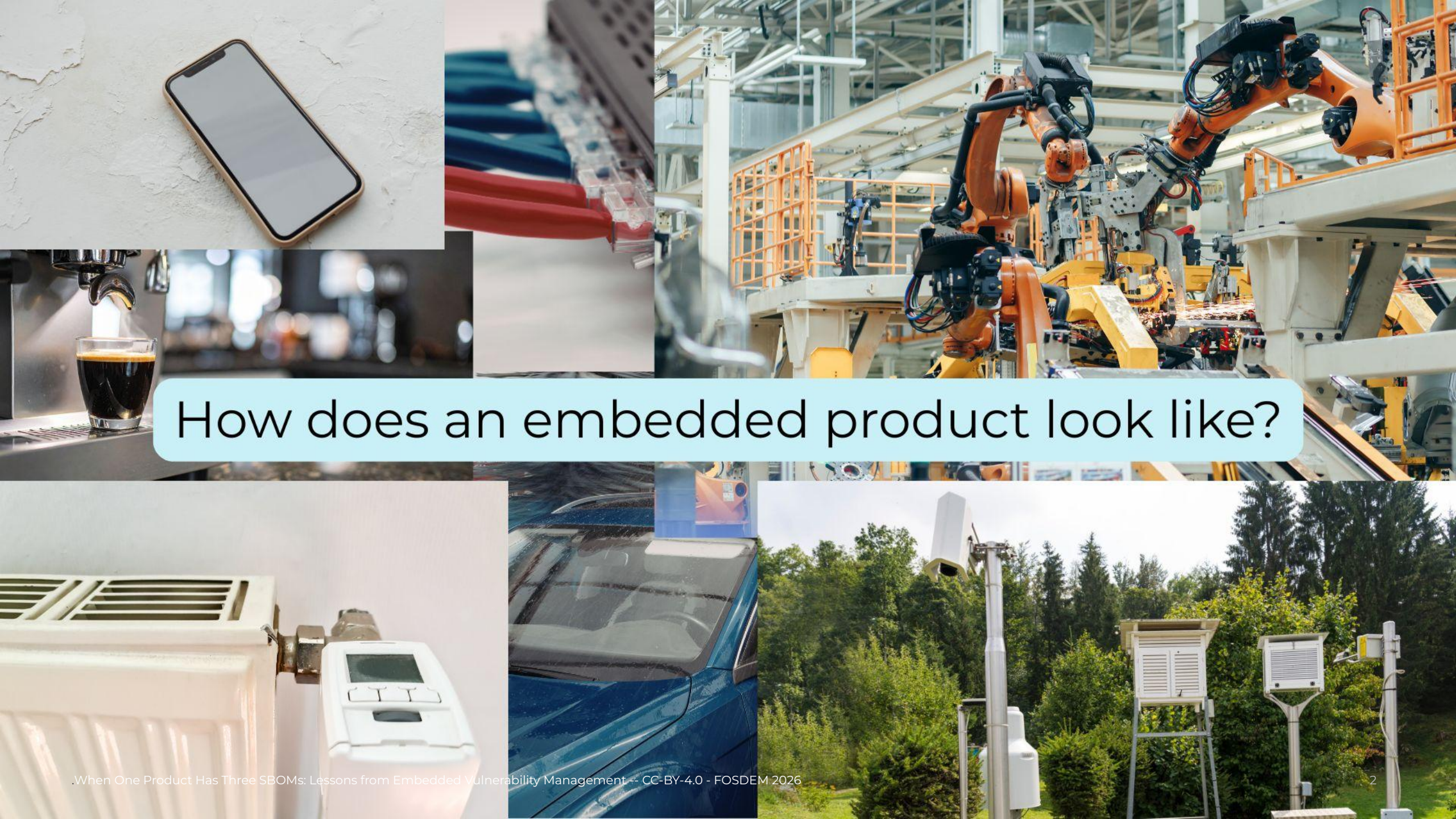
YGREKY

# When One Product Has Three SBOMs: Lessons from Embedded Vulnerability Management

Marta Rybczynska



# How does an embedded product look like?

# How does an embedded product look like?
## More complex than you think!

YGREKY

- The “device”
  - A high-end processor running Linux typically
  - One more more microcontrollers for power, sensors…
  - A GPU (sometimes)
  - Additional processors (chipsets): Bluetooth, WiFI, 4G/5G, industrial standards
- The server side (“remote processing” in the CRA language)
  - Data gathering/aggregation
  - Remote control
  - Updates and onboarding (and fleet management)
  - And more…
- A controlling application
  - Runs on a mobile phone, using mobile technologies

# Let’s create an SBOM for my embedded product!

YGREKY

- One SBOM?
  - There will be more than one
- Tooling is mature, right?!
  - We’re 1,5 years from the CRA
  - We’ll use a simple model, only three processors
- We will follow what CRA recommends
  - Generate SBOM and use it for vulnerability management

# Let’s create an SBOM for my embedded product!

YGREKY

- One SBOM?
  - There will be more than one
- Tooling is mature, right?!
  - We’re 1,5 years from the CRA
  - We’ll use a simple model, only three processors
- We will follow what CRA recommends
  - Generate SBOM and use it for vulnerability management

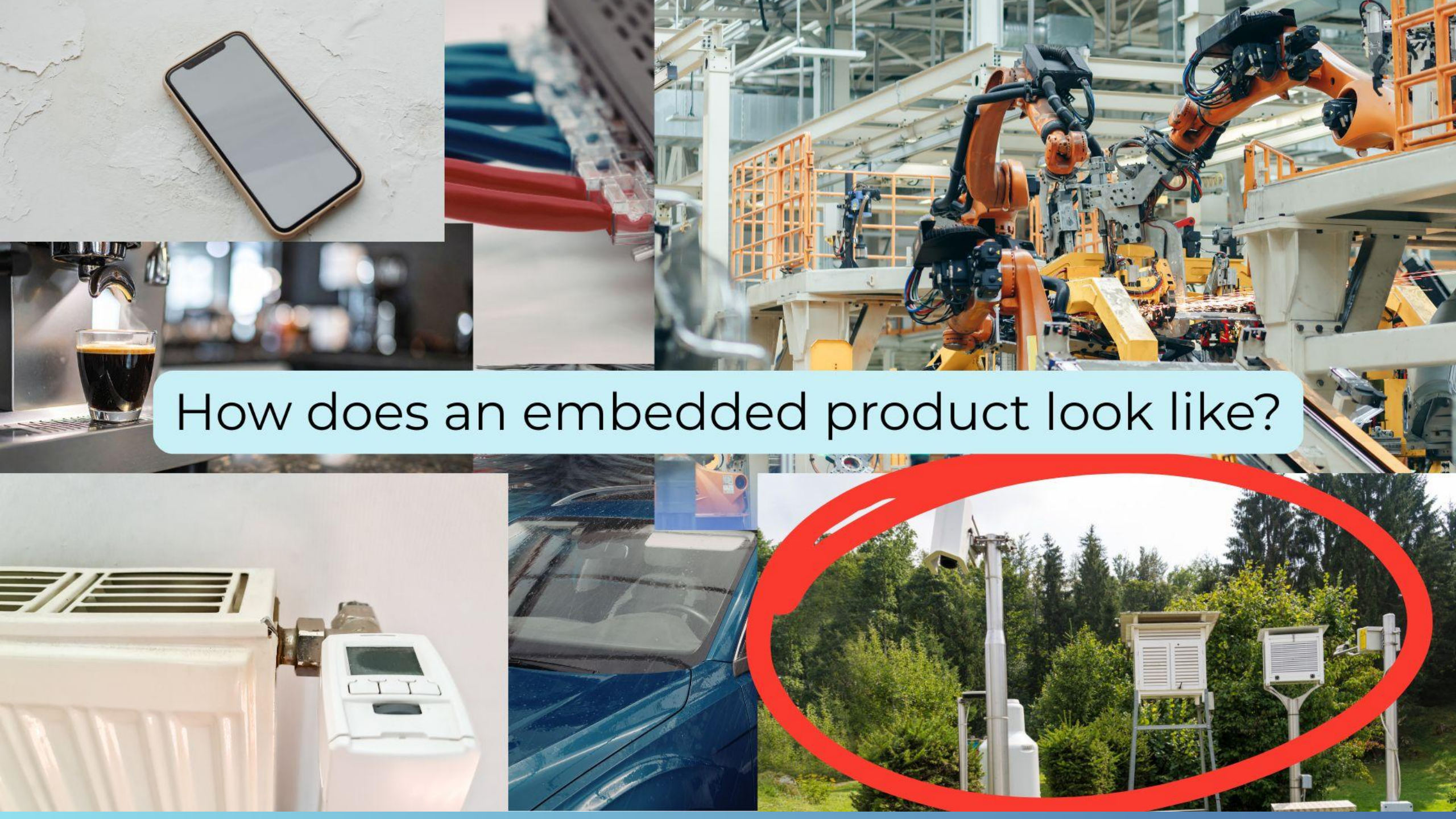
What could go wrong?

# But first... why

# Who is Marta Rybczynska?

YGREKY

- PhD in Telecommunications
  - Network security/anonymity systems
- Open source/embedded developer/architect
  - 20+ years in open source
  - Contributions to the Linux kernel, various RTOSes
  - Co-maintainer of meta-security YP layer
  - Yocto Project security team and Open Embedded Technical Steering Committee member
- Founder/CEO of Ygreky
  - Consulting (processes, architecture, audits)
  - Teaching (“Embedded Security”, webinars)
  - Current contribution to CRA-related standardisation
  - Conference keynotes


How does an embedded product look like?

# A simple embedded product


- The “base station”
  - Running Linux with the Yocto Project
- “Sensors”
  - Running Zephyr
- The “server application”
  - Managing sensor data from multiple devices, REST API with a database, in Python
  - Directly on Linux (venv), could be a container

# A simple embedded product: native SBOMs

YGREKY

- The “base station”
  - Running Linux with the Yocto Project : **SPDX 3.0**
- “Sensors”
  - Running Zephyr : **SPDX 2.2**
- The “server application”
  - Managing sensor data from multiple devices, REST API with a database, in Python: **Python-native tooling**
  - Directly on Linux (venv), could be a container

# A simple embedded product: SBOM management

YGREKY

- The “base station”
  - Running Linux with the Yocto Project : **SPDX 3.0**
- “Sensors”
  - Running Zephyr : **SPDX 2.3**
- The “server application”
  - Managing sensor data from multiple devices, REST API with a database, in Python: **Python-native tooling**
  - Directly on Linux (venv), could be a container
- SBOM management tool and vulnerability scanning: **Dependency Track**
  - Frequently used
  - Interoperability testing CycloneDX<->SPDX

# SBOM Generation: Yocto Project whinlatter (5.3)

YGREKY

- What is the Yocto Project?
  - Distribution building system for embedded Linux
  - A Linux Foundation project
  - Configurable, extensible by so called “layers”
  - Each included package is described in a “recipe” defining how to download and build it
  - Builds from low level software (the kernel, bootloaders) to final applications
  - NOT linked to any standard Linux distribution
- SBOM generation status
  - SPDX 3.0 generated by default at build
  - 3rd party CycloneDX patches (unofficial)

# SBOM Generation: Zephyr

YGREKY

- What is the Zephyr?
  - A real-time operating system
  - A Linux Foundation project
  - Supports various microcontroller boards
  - Multiple examples available
- SBOM generation status
  - SPDX 2.3 can be generated with its build tool ‘west’
    - In official documentation, but need to search for it: https://docs.zephyrproject.org/latest/develop/west/zephyr-cmds.html
    - Video reference: https://www.youtube.com/watch?v=PVqSHqXf4LE “Practical SBOM Management with Zephyr and SPDX” by Benjamin Cabé
  - No CycloneDX support I know of

# SBOM Generation: Zephyr

YGREKY

- What is the Zephyr?
  - A real-time operating system
  - A Linux Foundation project
  - Supports various microcontroller boards
  - Multiple examples available
- SBOM generation status
  - SPDX 2.3 can be generated with its build tool ‘west’
    - In official documentation, but need to search for it: https://docs.zephyrproject.org/latest/develop/west/zephyr-cmds.html
    - Video reference: https://www.youtube.com/watch?v=PVqSHqXf4LE “Practical SBOM Management with Zephyr and SPDX” by Benjamin Cabé
  - No CycloneDX support I know of

```
west spdx --init -d build
west build -b qemu_x86 samples/hello_world -- -DCONFIG_BUILD_OUTPUT_META=y
west spdx -d build
ls -l build/spdx/app.spdx
```

# SBOM Generation: Python Linux application

YGREKY

- What kind of architecture?
  - A Python app with REST APIs
  - Standard build with requirements.txt
- SBOM generation status
  - SPDX haven’t tried it this test
  - CycloneDX with cyclonedx-py
    - There are other ways too

```
pip install pipdeptree cyclonedx-bom
cyclonedx-py requirements requirements.txt > cyclonedx-sbom.json
```

# SBOM Generation: Python Linux application

YGREKY

- What kind of architecture?
  - A Python app with REST APIs
  - Standard build with requirements.txt
- SBOM generation status
  - SPDX haven't tried it this test
  - CycloneDX with cyclonedx-py
    - There are other ways too

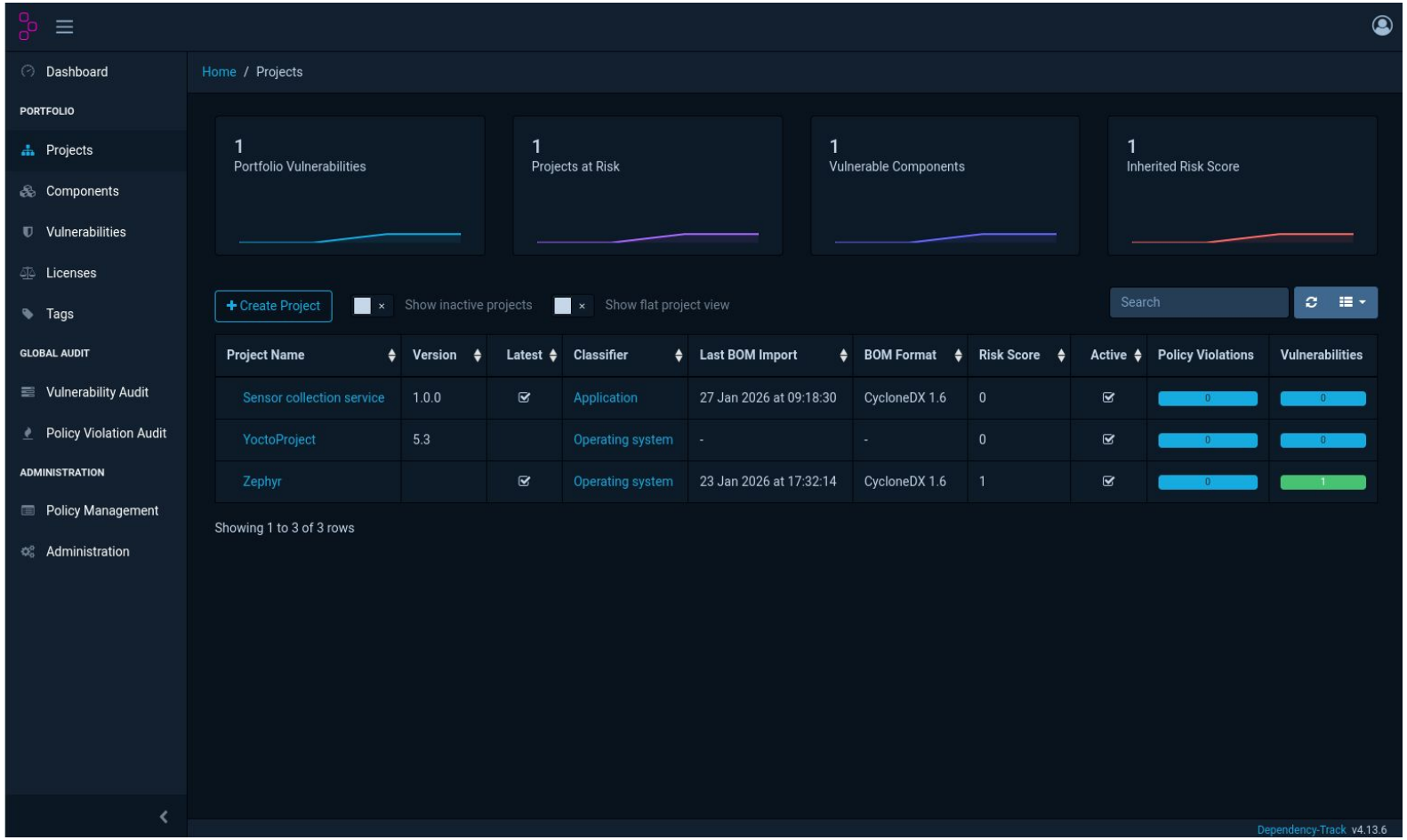
```
Better:
pip install pipdeptree cyclonedx-bom
pipdeptree --freeze > requirements.txt
cyclonedx-py requirements requirements.txt > cyclonedx-sbom.json
```

# SBOM Management: DependencyTrack

## One the screenshot: one of our intermediary states

YGREKY

- Deployed as a container
- Worked rapidly

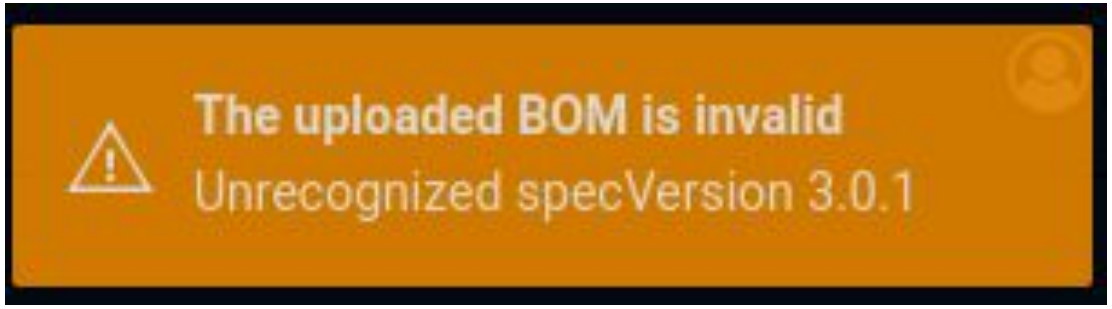


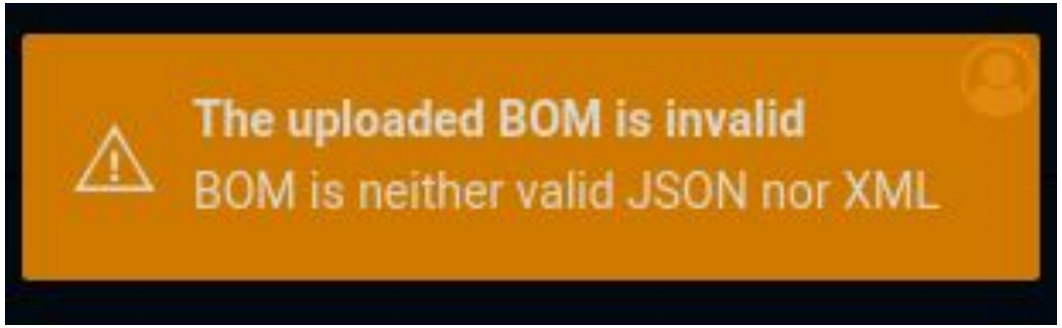
# Let’s go!
## Importing our SBOMs into DependencyTrack

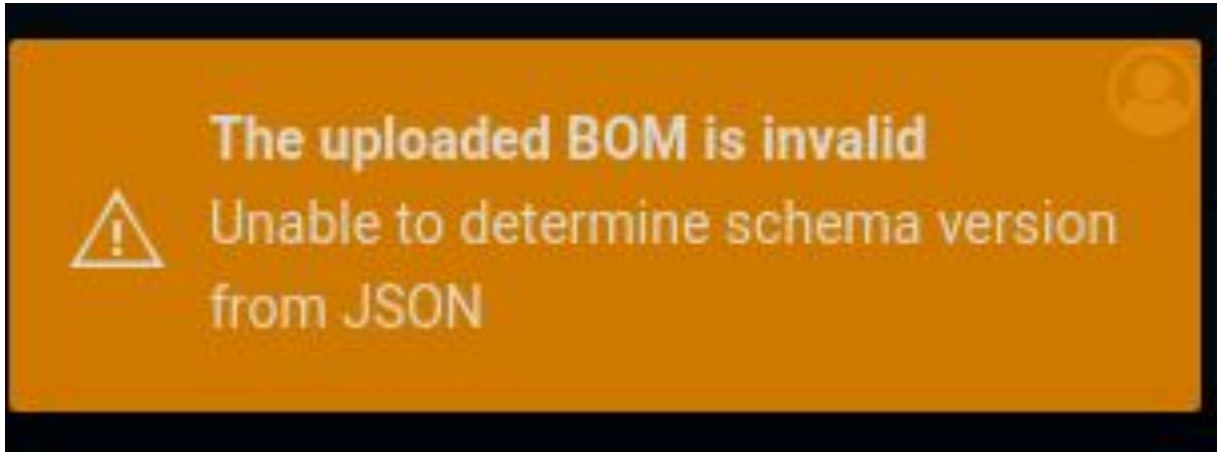
YGREKY

# Let’s go!
## Importing our SBOMs into DependencyTrack

YGREKY



# Let’s Go: What happened?

YGREKY

- CycloneDX from our Python service
  - Worked
- Importing SPDX didn’t work
  - SPDX 3.0 from YP - nope
  - SPDX 2.3 from Zephyr - in tag: value format, not XML or JSON
    - Tag:value not supported, tried conversion
    - No luck with SPDX in XML nor JSON

```
Conversion of SPDX 2.3 into various formats:
pyspdxtools --infile app.spdx --outfile app.spdx.json
pyspdxtools --infile app.spdx --outfile app.spdx.xml
```

# Let’s Go: Trying conversion
## Zephyr

YGREKY

- SPDX 2.3 from Zephyr
  - Conversion SPDX 2.3 to CycloneDX via Syft -> experimental (?) but worked
  - Import to DependencyTrack worked!

```
SPDX 2.3 to CDX:
syft convert app.spdx -o cyclonedx-json=app.cdx.json
```

# Let’s Go: Trying conversion
## Yocto Project

YGREKY

- SPDX 3.0 from the Yocto Project
  - No working conversion tools from SPDX 3.0 to CycloneDX
  - -> Change generation to SPDX 2.2

# Special case: Yocto Project SPDX 2.2 to CycloneDX

YGREKY

- SPDX 2.2 in YP has a specific format
  - One file per recipe, with an index file
  - No conversion tool handles that ;(

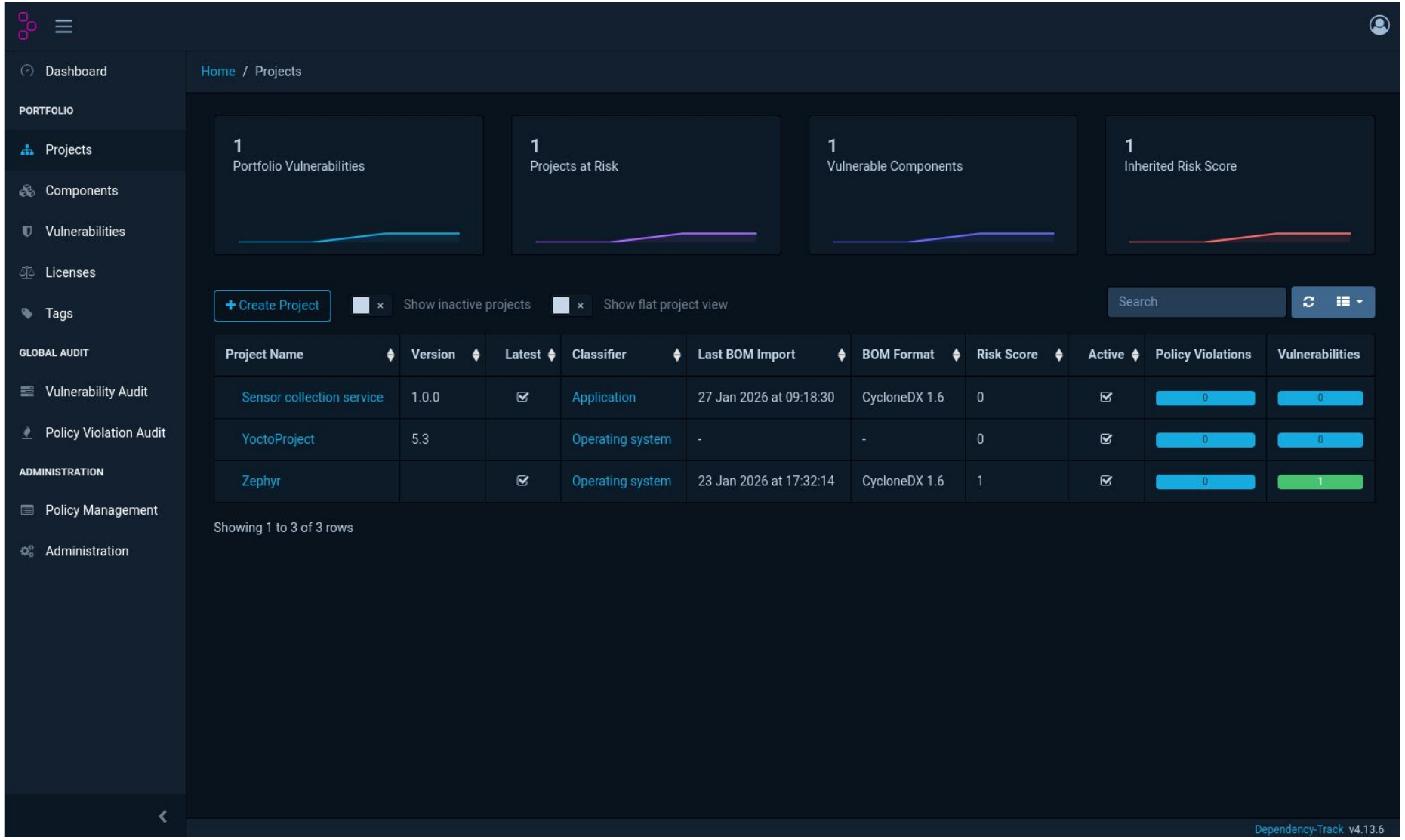
# Special case: Yocto Project SPDX 2.2 to CycloneDX

YGREKY

- SPDX 2.2 in YP has a specific format
  - One file per recipe, with an index file
  - No conversion tool handles that directly ;(
- Conversion path
  - Convert all SPDX 2.2 files to CycloneDX
  - Than merge all CDX
  - Import into DepedencyTrack … worked!

```
YP SPDX2.2 to CDX:
for i in *.spdx.json; do syft convert $i -o cyclonedx-json=$i.cdx.json; done
cyclonedx-linux-x64 merge   --input-files *.cdx.json   --output-file \
merged.cdx.json
cyclonedx-linux-x64 validate   --input-file merged.cdx.json   --fail-on-errors
```

# Import worked: now start vulnerability analysis



# Import worked: now start vulnerability analysis

YGREKY

- Findings from the analysis
  - Detail level not equal
    - At the Python generation, dependency depth matters
  - Conversion losses for the YP SBOM
    - Package name updates, CPEs…
    - For example: the Linux kernel has lost annotations (CPEs)
    - -> consequence: no CVEs reported for the Linux kernel
  - Missing version numbers
    - Especially on the Zephyr side (only some version numbers present)
    - -> limited results
- Work to be done here…

# Take-aways: there is work to be done

YGREKY

- Tools in one ecosystem usually work
  - But less if we merge
- Multiple formats (too many?)
  - I didn’t expect to see tag:value
  - Conversion is complicated
    - Experimental tools, changing formats
    - Ecosystem-specific data is lost
- Cross checking between vulnerability tools
  - Needed next step
  - Requires automation, handling hundreds of packages manually not practical
  - Little information about metadata in DependencyTrack
- More integration and testing work is URGENTLY needed…
  - Real products are MORE complex

# Questions?