ENDOR LABS

In SBOMs We Trust:

How Accurate, Complete, and Actionable Are they?

Henrik Plate
Joseph Hejderup

FOSDEM'23

Henrik Plate

Security Researcher

Joseph Hejderup

Researcher

Agenda

1. Why do I need an SBOM?
2. Case Study - Why can the same app have different SBOMs?
3. Why is it helpful to enrich SBOMs with call graph information?
4. Takeaways

Software Bill of Materials


```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.4",
  "serialNumber": "urn:uuid:efa1076f-e57c-434c-ad56-3d5a0f401967",
  "version": 1,
  "metadata": {
    "timestamp": "2023-01-26T18:06:34+01:00",
    "tools": [ {
      "name": "Tool A",
    } ],
    "component": {
      "bom-ref": "6e5f53bb4d29d051",
      "type": "container",
      "name": "eclipse/steady-rest-backend:3.2.5",
      "version": "sha256:0be6a5be5d727608b8f6d9dda73646db794
    }
  },
  "components": [
    {
      "bom-ref": "pkg:maven/org.hdrhistogram/HdrHistogram@2.1.12?package-id=6d5f7d142788ba37",
      "type": "library",
      "group": "org.hdrhistogram",
      "name": "HdrHistogram",
      "version": "2.1.12",
      "cpe": "cpe:2.3:a:HdrHistogram:HdrHistogram:2.1.12:*:*:*:*:*:*:*",
      "purl": "pkg:maven/org.hdrhistogram/HdrHistogram@2.1.12",
      "properties": [ ...
```


SBOM format

Creation timestamp and SBOM generator

Software product

Contained components

Why do you need SBOMs?

Use-cases

- Inventory & visibility of software components
- Tracking licence violations
- Systematic view of security & operational risks
- Insights to current adoption and procurement practices of packages

Emerging regulations will require software vendors to provide SBOMs to their customers

- Candidate EU Cybersecurity Certification Scheme for Cloud Services (Dec 2020)
- Executive Order 14028 (May 2021)
- FDA Draft Guidance (Apr 2022), effective as of June 2023
- H.R. 7900 - National Defense Authorization Act for Fiscal Year 2023 (Aug 2022)
- Etc.

Agenda

1. What are SBOMs and why do I need one?
2. Case Study - Why can the same app have different SBOMs?
3. Why is it helpful to enrich SBOMs with call graph information?
4. Takeaways

Case-study

Idea

- SBOMs can be created at different stages in the software lifecycle, incl. from the software source, at build time, or after build through binary analysis. [2] [1, pp. 6-7]

Sample Software

- Eclipse Steady 3.2.5
- Module “rest-backend”: Service developed with Java/Maven and Spring Boot
- Ground truth: 114 compile, 2 runtime, 41 test deps shown by Maven Dep. Plugin
- Docker image: eclipse/steady-rest-backend:3.2.5

[1] Xia et al.: An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead (2023)
[2] The Minimum Elements For a Software Bill of Materials (SBOM)

Background: Component identifiers

Context-specific component identifiers, e.g.

- Maven: groupId, artifactId, version (GAV) (optional: type, classifier)
  Example: org.dom4j:dom4j:2.1.3
- Common Platform Enumeration (CPE): part, vendor, product, version, …
  Example from CVE-2020-10683: cpe:2.3:a:dom4j_project:dom4j:*:*:*:*:*:*:*:*

Universal component identifiers, e.g.,

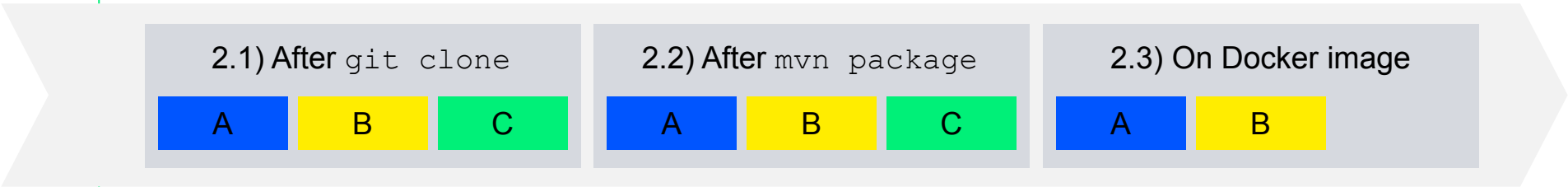
- Package URL (PURL): Type, namespace, name, version, qualifiers

Finding & mapping such identifiers is key for metadata-centric tools

```
"bom-ref": "pkg:maven/dom4j/dom4j@2.1.3?package-id=7be91dd889ff562b",
"type": "library",
"name": "dom4j",
"version": "2.1.3",
"cpe": "cpe:2.3:a:dom4j:dom4j:2.1.3:*:*:*:*:*:*:*",
"purl": "pkg:maven/dom4j/dom4j@2.1.3"
```

Approach

1) Select three open source SBOM generators
   - A + B are generic solutions to scan directories, images, etc.
   - C is a Maven plugin that hooks into Maven's build process
2) Run at different points in time / on different targets



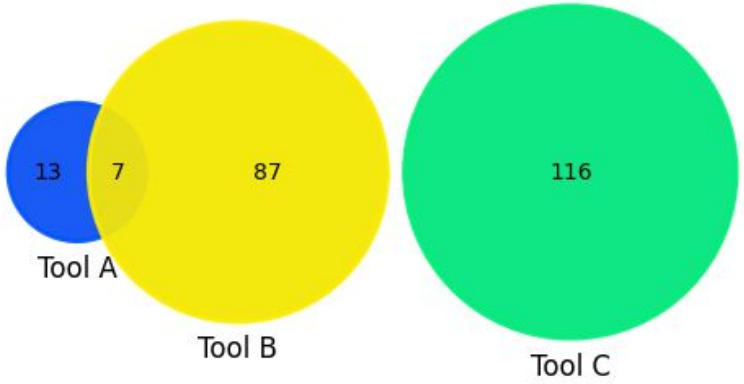
3) Compare SBOM data
   a. Precision and recall of tools (identified components' GAV vs. ground truth)
   b. Venn diagrams show where tools (dis)agree (on the level of PURLs and simple names)
   c. Additional properties

2.1) git clone & PURLs

A

- **precision = 0.5, recall = 0.06**
- Does not resolve the deps declared in the pom.xml
- Few components, incl. test deps, **some w/o version**
- Additional properties: CPE combinations, file

```
"bom-ref": "pkg:maven/org.springframework.boot/spring-boot-starter?p
"type": "library",
"group": "org.springframework.boot",
"name": "spring-boot-starter",
"cpe": "cpe:2.3:a:spring-boot-starter:spring-boot-starter:*:*:*:*:*:
"purl": "pkg:maven/org.springframework.boot/spring-boot-starter",
```



B

- **precision = 0.93, recall = 0.75**
- Resolves dependencies in pom.xml, but some with wrong versions, and some missed altogether
- JARs in filesystem are ignored
- Additional properties: None

C

- **precision = 1.0, recall = 1.0**
- Correctly identifies all 114 compile + 2 runtime deps (PURLs contain additional qualifiers)
- Additional properties: Digests, description, license, external links

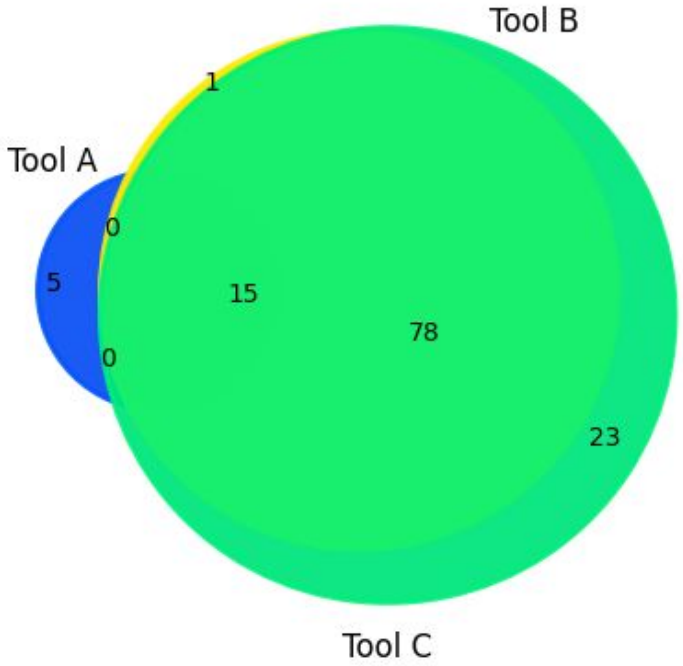
2.1) git clone & names

Intersections increase due to ignoring

- A's lack of version identifiers
- B's wrong version identifiers
- C's additional qualifiers

B

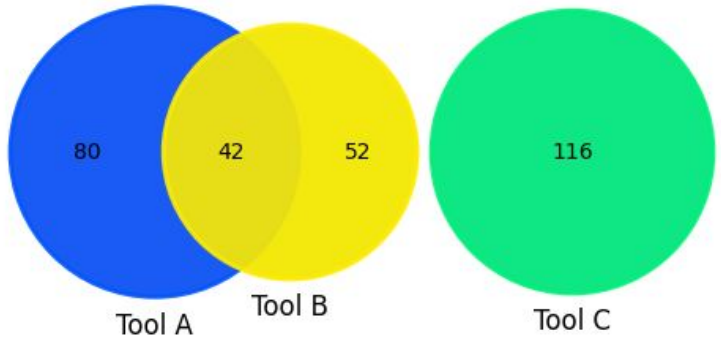
- Identifies the pom.xml itself as a component of type "application"
- 23 missing components in comparison to C are transitive deps



2.2) mvn package & PURLs

A

- **precision = 0.56, recall = 0.44**
- Improves due to processing the build result (self-contained Spring Boot app)
- Some groupIds are wrong, e.g., `"purl": "pkg:maven/dom4j/dom4j@2.1.3"`
- Test dependencies are included and partly redundant (from pom.xml, src/test/resources, target/…)



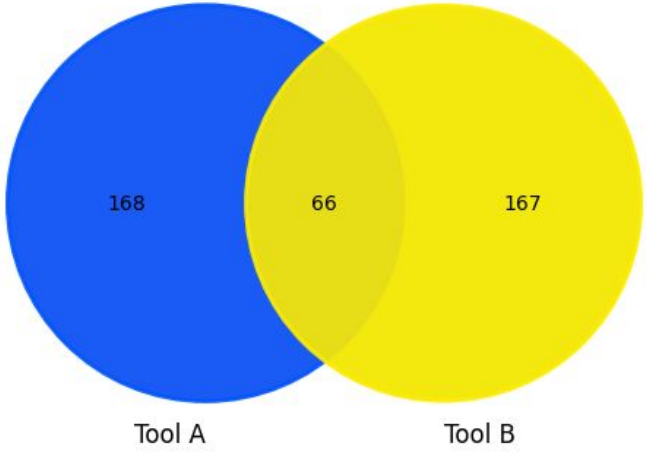
B

- Same as before (precision = 0.93, recall = 0.75)
- Example PURL: `"purl": "pkg:maven/org.dom4j/dom4j@2.1.3"`

C

- Same as before (precision = 1.0, recall = 1.0)
- Example PURL: `"purl" : "pkg:maven/org.dom4j/dom4j@2.1.3?type=jar"`

2.3) Docker image & PURLs



OS-level components introduced through Docker image

A

- **precision = 0.59, recall = 0.55 (Maven components only)**
- `"purl": "pkg:maven/org.objectweb.asm/asm@9.1"`
- `"purl": "pkg:maven/com.sun/activation@1.1"`
- `"purl": "pkg:deb/ubuntu/dash@0.5.11+git20210903+057cd650a4ed-3build1?arch=amd64&distro=ubuntu-22.04"`

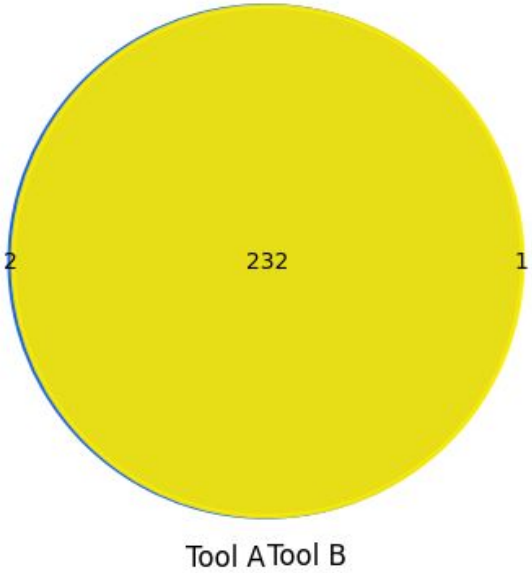
B

- **precision = 0.96, recall = 0.91 (Maven components only)**
- `"purl": "pkg:maven/org.ow2.asm/asm@9.1"`
- `"purl": "pkg:maven/javax.activation/activation@1.1"`
- `"purl": "pkg:deb/ubuntu/dash@0.5.11+git20210903+057cd650a4ed-3build1?distro=ubuntu-22.04"`

Differences due to

- Maven groupIds: “org.objectweb.asm” vs. “org.ow2.asm”
- Qualifiers: “arch=amd64&distro=ubuntu-22.04” vs. “distro=ubuntu-22.04”

2.3) Docker image & names



A
- Unique components (true-positive) found in /opt/java/openjdk
  `"purl": "pkg:generic/java@11.0.16.1+1"`
  `"purl": "pkg:maven/jrt-fs/jrt-fs@11.0.16.1"`

B
- Unique component (true-positive)
  `"purl": "pkg:maven/net.bytebuddy/byte-buddy@1.10.22"`

Comments
- The Java runtime is a big miss of B
- A and B have false-positive `"purl": "pkg:maven/net.bytebuddy/byte-buddy-dep@1.10.22"` (maybe due to the nested JAR containing multiple pom.xml)

Again: Attention
- Overlap on **smallest denominator**
- False-negatives and false-positives when considering the complete identifier (GAV)
- Name itself is not sufficient for vuln. search

Case-study: Lessons learned

Reasons for getting different SBOMs

- Integrated vs. generic tools
- Production vs. test components (different defaults; scope is not reflected)
- Execution during different life cycle phases (recommended field [1], but not present)

Standard data format but…

- Different fields (license, digests, CPEs, …)
- Same fields populated differently (PURLs)

Not discussed here:

- Time of dependency resolution (in case of version ranges)
- Platform

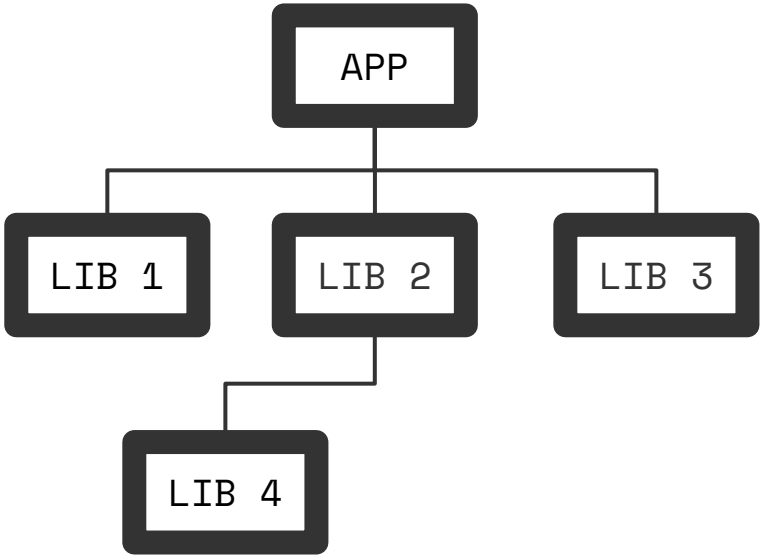
[1] The Minimum Elements For a Software Bill of Materials (SBOM)

Agenda

1. What are SBOMs and why do I need one?
2. Case Study - Why can the same app have different SBOMs?
3. Why is it helpful to enrich SBOMs with call graph information?
4. Takeaways

The standard view

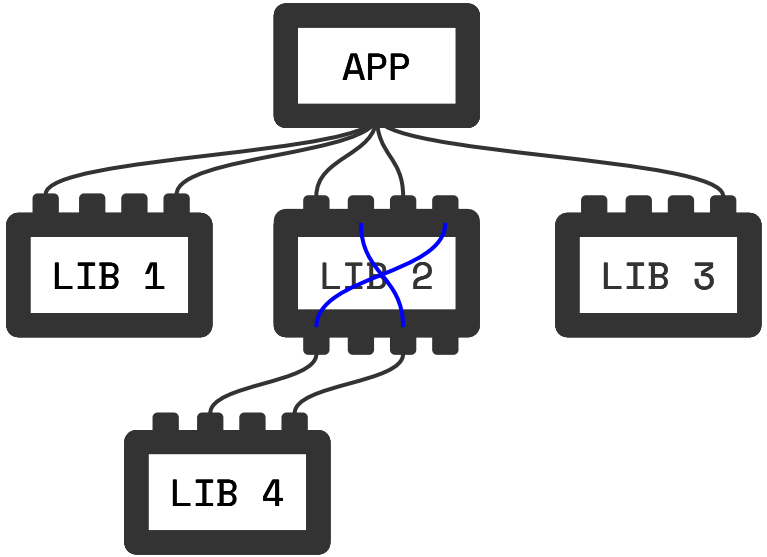
- Imported Components
- Relationship between Components
- Reachability & Impact analysis



The stadard view augmented with call graphs

- Utilized Code functionality
- Relationship between Code entities
- Pinpoint exactly how components are utilized

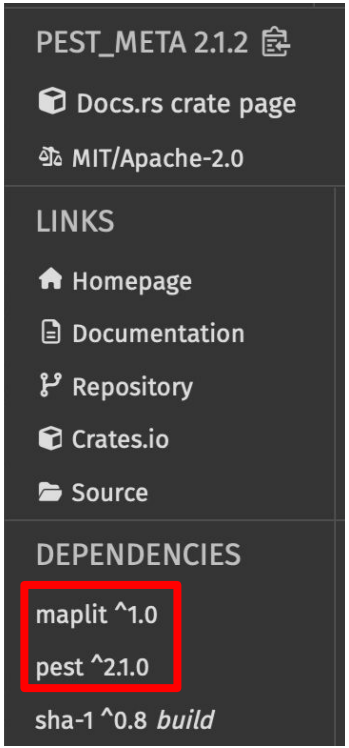
Example: App does not actually use LIB 4, should we include it in our SBOM?

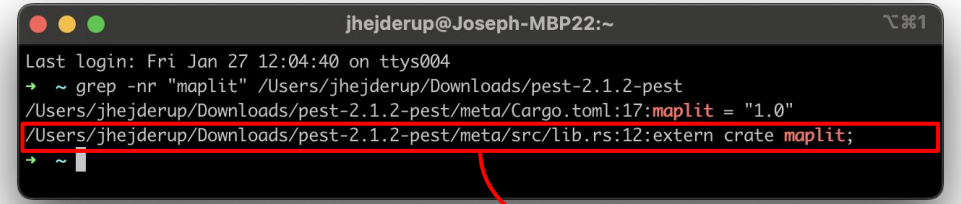


API Level

■ = package functionality

Example: Package `pest_meta 2.12`





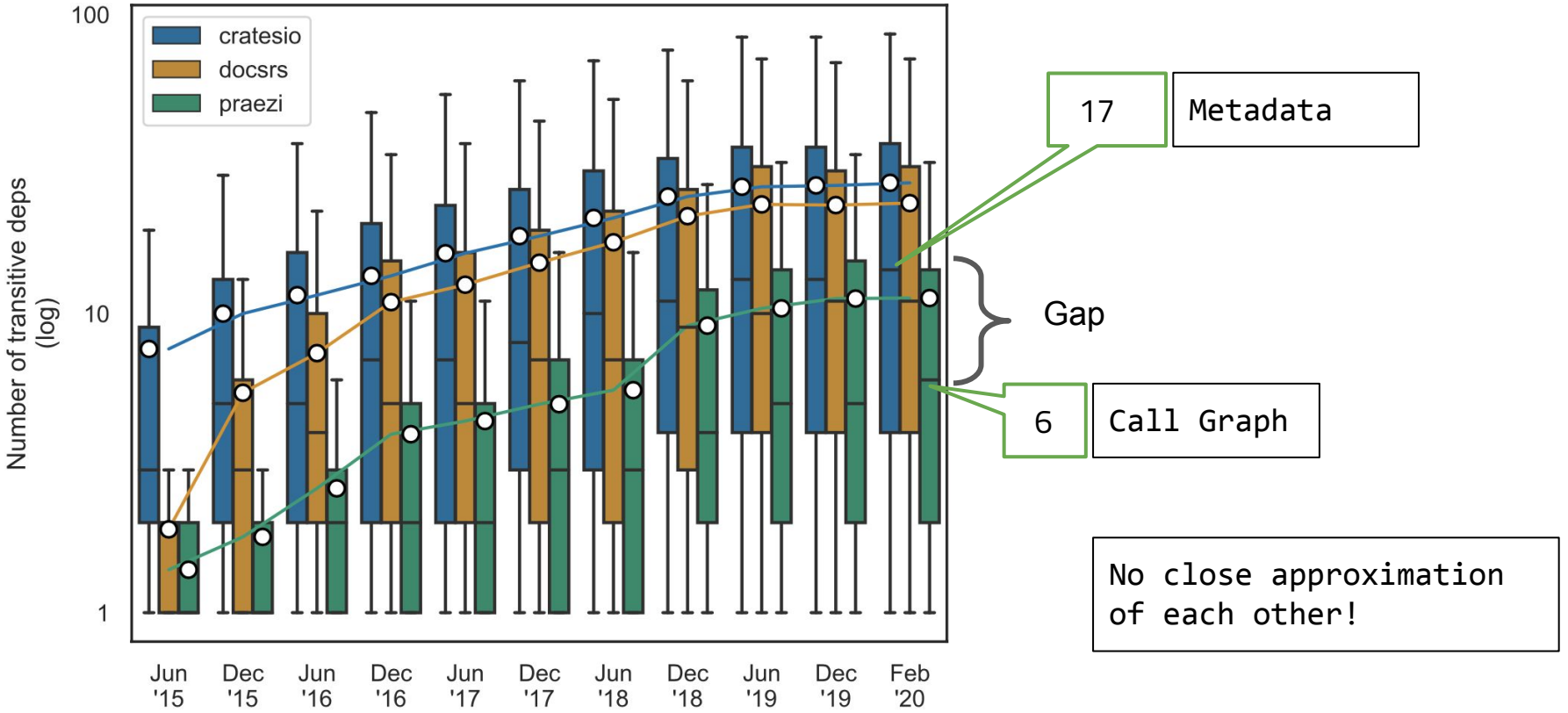
One line of import but not other usage!

Empirical Study - Rust's Crates.io!

RQ: What is the difference in the number of reported dependencies between traditional metadata-based approaches vs program analysis approaches?

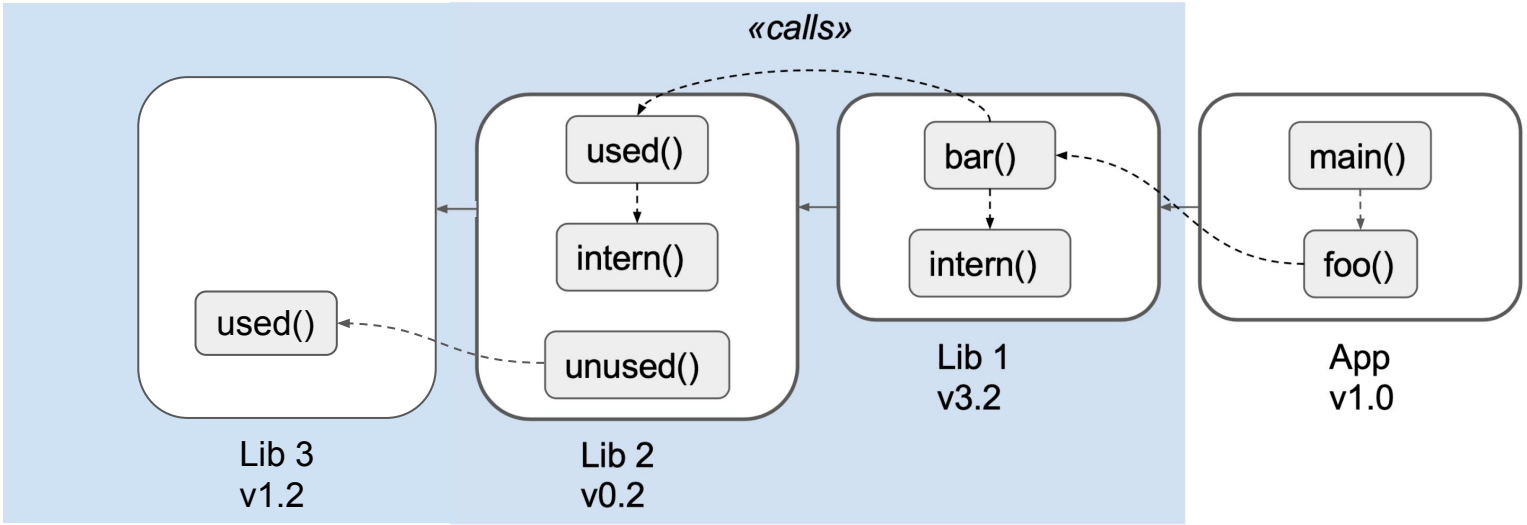
Hejderup, J., Beller, M., Triantafyllou, K., & Gousios, G. (2022). Präzi: from package-based to call-based dependency networks. *Empirical Software Engineering*, *27*(5), 1-42.

SBOM: Metadata vs Call Graph



Hejderup, J., Beller, M., Triantafyllou, K., & Gousios, G. (2022). Präzi: from package-based to call-based dependency networks. *Empirical Software Engineering*, *27*(5), 1-42.

Why is there such a huge difference?



We make the general assumption that we use

- **ALL APIs** of all direct dependencies,
- and **then ALL APIs** of transitive dependencies

Call Graphs: Lessons learned

Security & Operational Risks are challenging to quantify with an SBOM

- Embedding function call information can help:
  - Pinpoint & Prioritize DevSecOps risks
  - Understand reuse of components: **What are my top critical and non critical dependencies?**
- Package-level information is not enough: **we need more layers of information that serve different users of SBOMs**
  - Developers **debug function call traces and not package relationships!**
  - Security folks need holistic views not **code examples!**

Agenda

1. What are SBOMs and why do I need one?
2. Case Study - Why can the same app have different SBOMs?
3. Why is it helpful to enrich SBOMs with call graph information?
4. Takeaways

Takeaways

- Standardized SBOM formats are necessary but not sufficient [2].
- More context is needed for actionable insights, e.g., call graph information.
- Consumers can hardly verify the correctness of SBOMs (and associated VEX documents [1]), especially for proprietary software .
- The evaluation and comparison of SBOM generators requires a benchmark (comparable to the DaCapo benchmark for Java).

[1] Xia et al.: An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead (2023)
[2] The Minimum Elements For a Software Bill of Materials (SBOM)

Joseph Hejderup

joseph@endor.ai

Henrik Plate

henrik@endor.ai