The Impact of SBOM Generators on Vulnerability Assessment in Python: A Comparison and a Novel Approach

Giacomo Benedetti, **Serena Cofano**, Alessandro Brighente, Mauro Conti

IMT SCUOLA ALTI STUDI LUCCA

Università di Genova

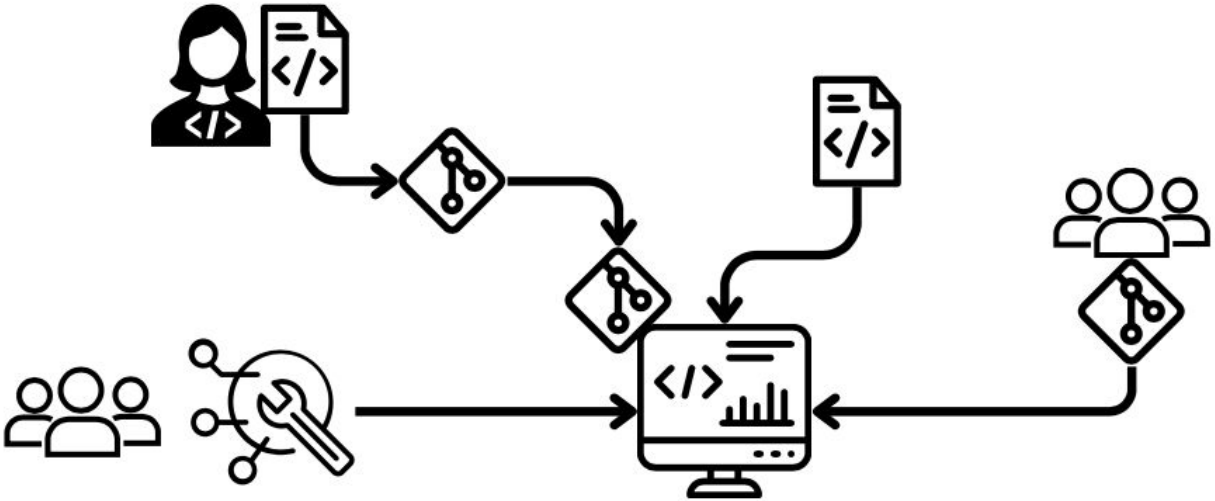
Università degli Studi di Padova

Background

- What is the general context?
    - Software Supply Chain security
- What is an SBOM?
- What is the role of an SBOM in vulnerability assessment?

Background - Software Supply Chain Security

Software Supply Chain refers to the collection of **devices**, **systems**, and **people** involved in creating and delivering final software.



Background - Software Supply Chain Security

The security of the Software Supply Chain is the security of its components.

How do we ensure Software Supply Chain Security?

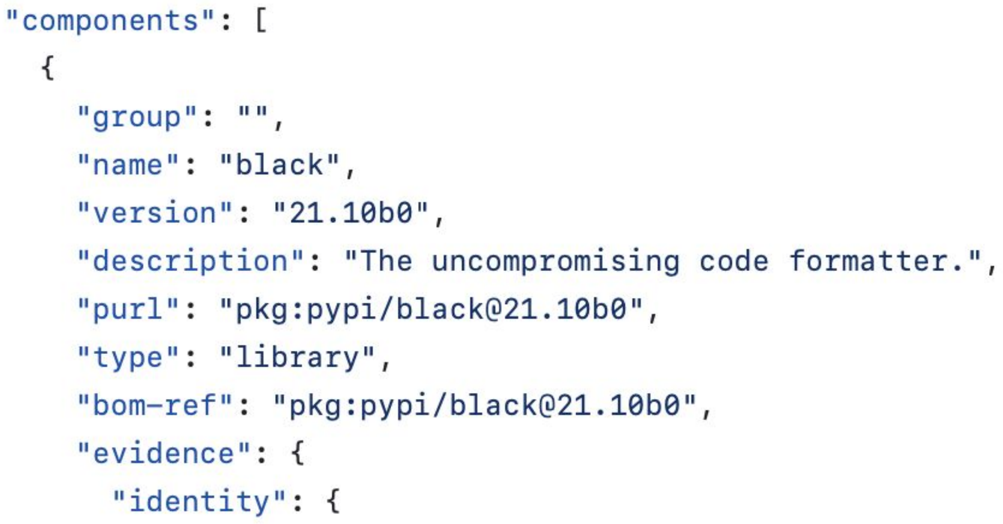
- Transparency → knowledge of the components in the Supply Chain
  → security analysis on them.

How do we know which components are present in the Supply Chain?

- SBOM

Background - SBOM Definition

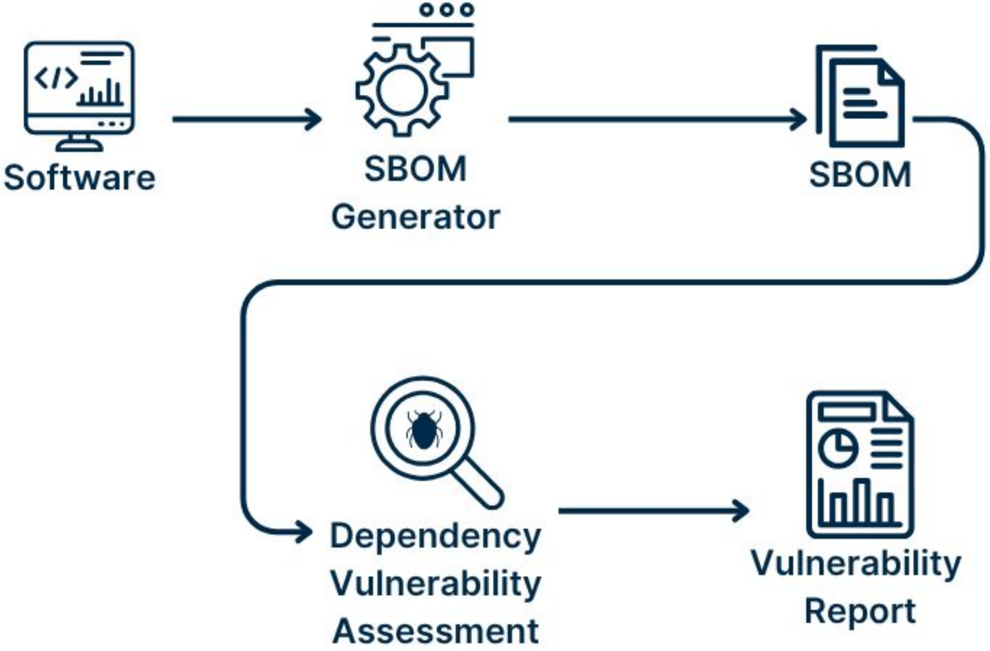
Software Bill of Materials (SBOM) is a detailed inventory of all components, libraries, and dependencies used in a software application, including their versions, sources, and licenses.



```
"components": [
  {
    "group": "",
    "name": "black",
    "version": "21.10b0",
    "description": "The uncompromising code formatter.",
    "purl": "pkg:pypi/black@21.10b0",
    "type": "library",
    "bom-ref": "pkg:pypi/black@21.10b0",
    "evidence": {
      "identity": {
```

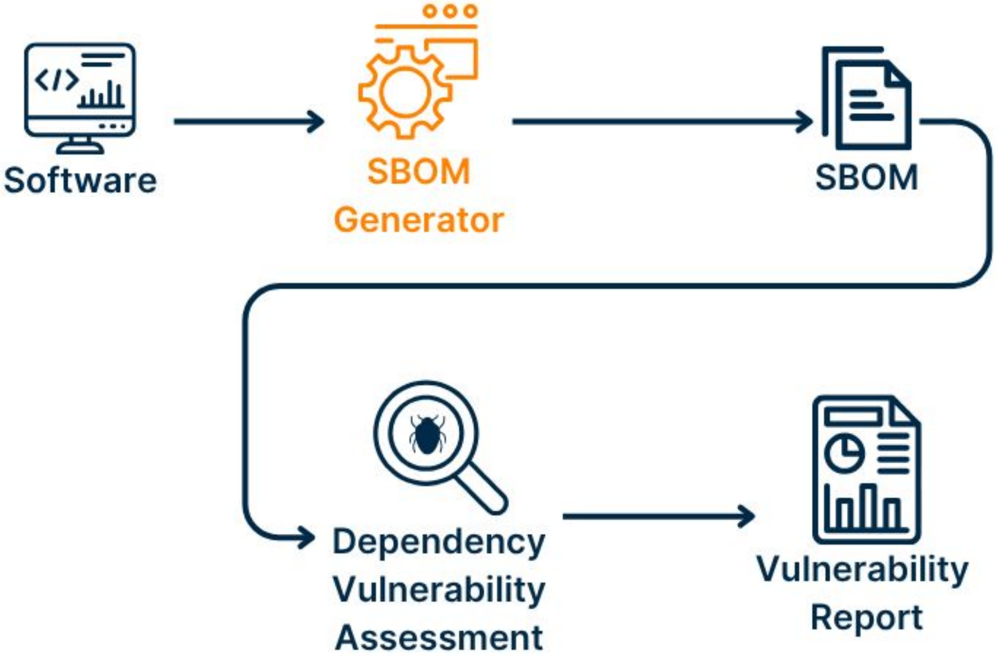
Background - SBOM in Security Assessment

The **SBOM** is used as input for tools that search for vulnerabilities.



Background - SBOM in Security Assessment

The **SBOM** is used as input for tools that search for vulnerabilities.



What is the problem?

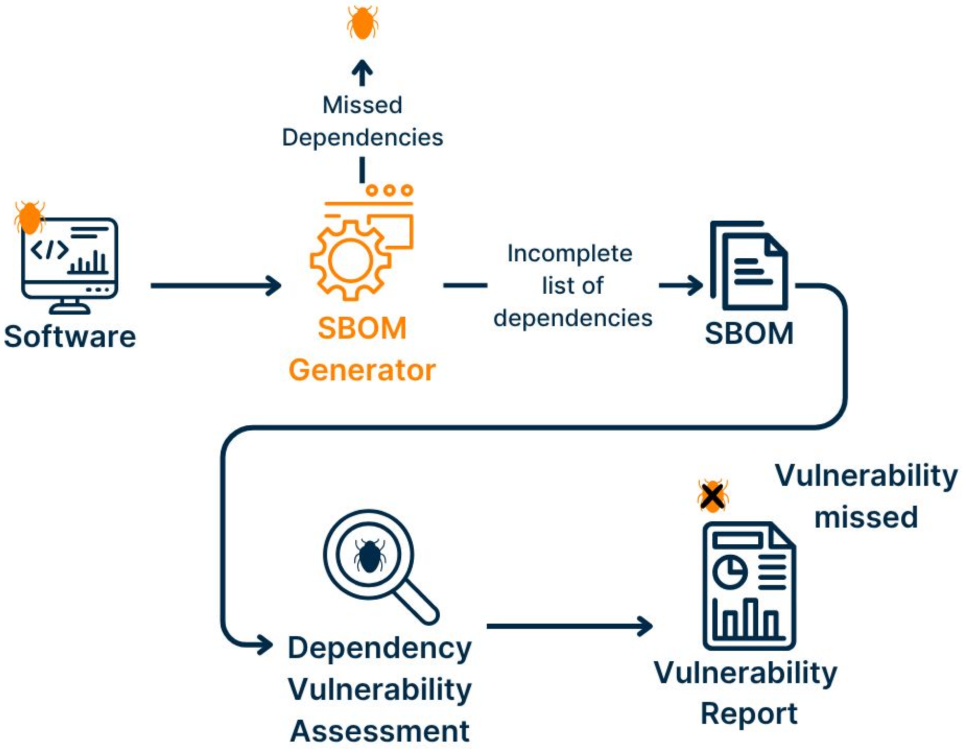
Problem

- SBOM generation has issues
  - Lack of completeness
  - Lack of correctness

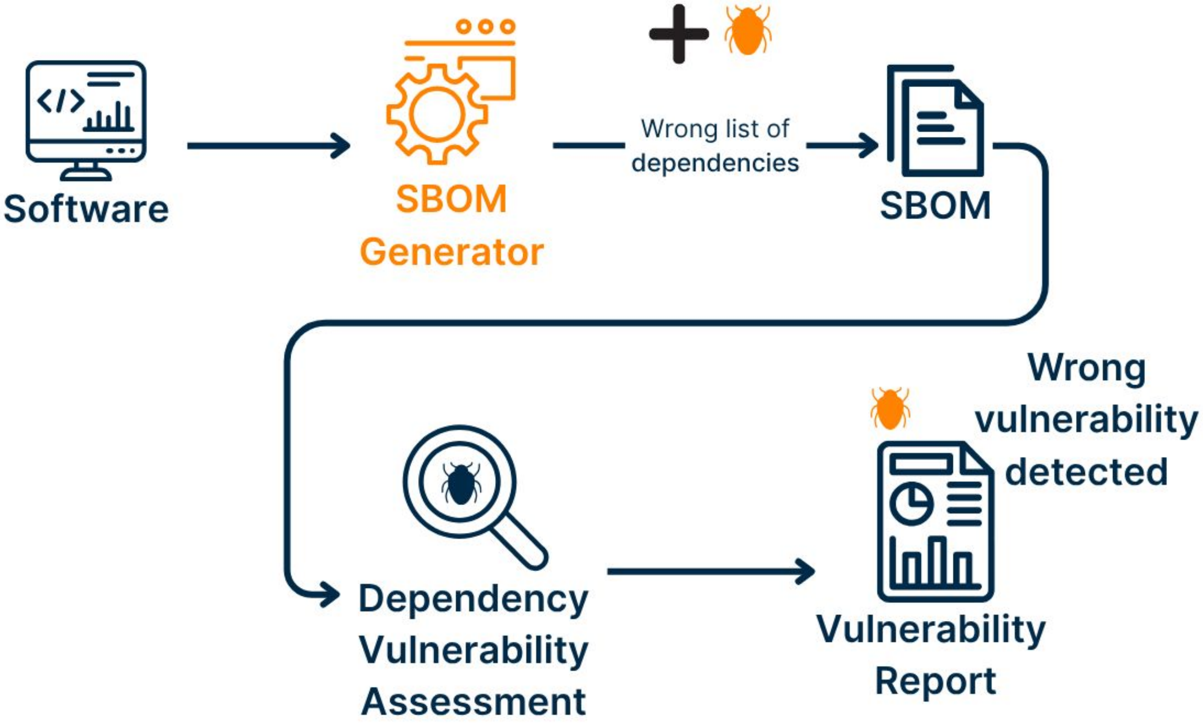
↓

- Dependencies vulnerability assessment is affected by:
  - False negatives → vulnerabilities present but not found
  - False positives → vulnerabilities found but not present

Problem - False negative



Problem - False positive


Software
SBOM Generator
Wrong list of dependencies
SBOM
Wrong vulnerability detected
Dependency Vulnerability Assessment
Vulnerability Report

Contributions

Comparison (RQ1)

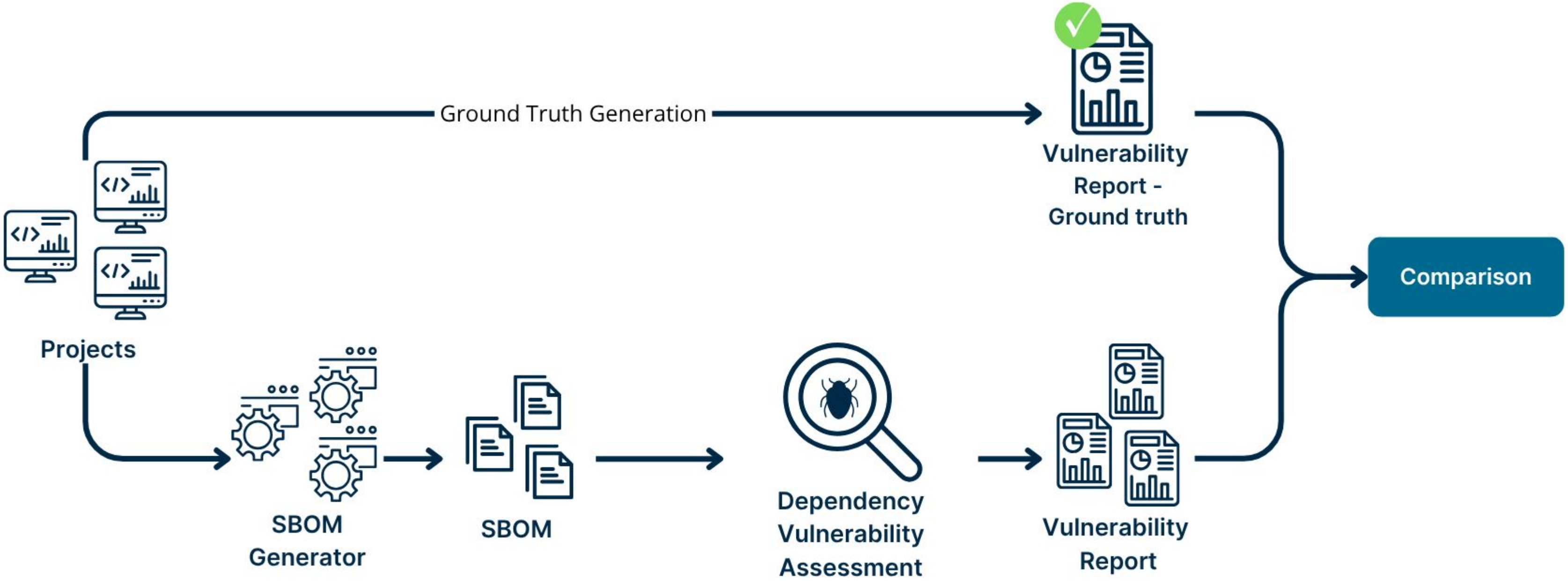
1. Investigate to what extent the SBOM generation affects the vulnerability assessment of dependencies.

Novel approach (RQ2)

2. Improve the performance of vulnerability assessment with a different approach to SBOM generation.

Comparison (RQ1)

Evaluation RQ1 - methodology



Evaluation RQ1 - Experimental setup

- Projects dataset: **1,000 Python** projects
- SBOM generation tools: **cdxgen**, **Gh-SBOM**, **ort**, **syft**, **trivy**.
- Ground truth: **pip-audit**.
- Metrics evaluation: **precision**, **recall** between the two vulnerability sets.

Evaluation RQ1 - dataset

Build tool	# of Packages
Poetry	65
Pdm	15
Hatch	144
Pipenv	12
Setuptools	764
Tot	**1000**

Evaluation RQ1 - SBOM generation tools

CDXGEN

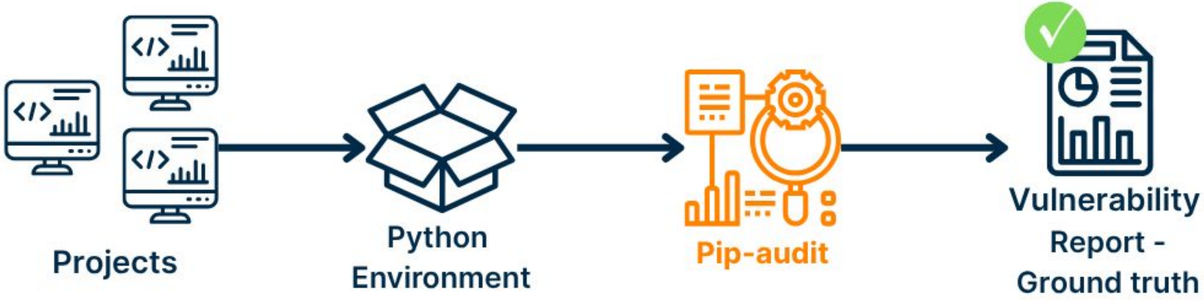
aqua trivy

GitHub

syft

OSS Review Toolkit

Evaluation RQ1 - Ground Truth

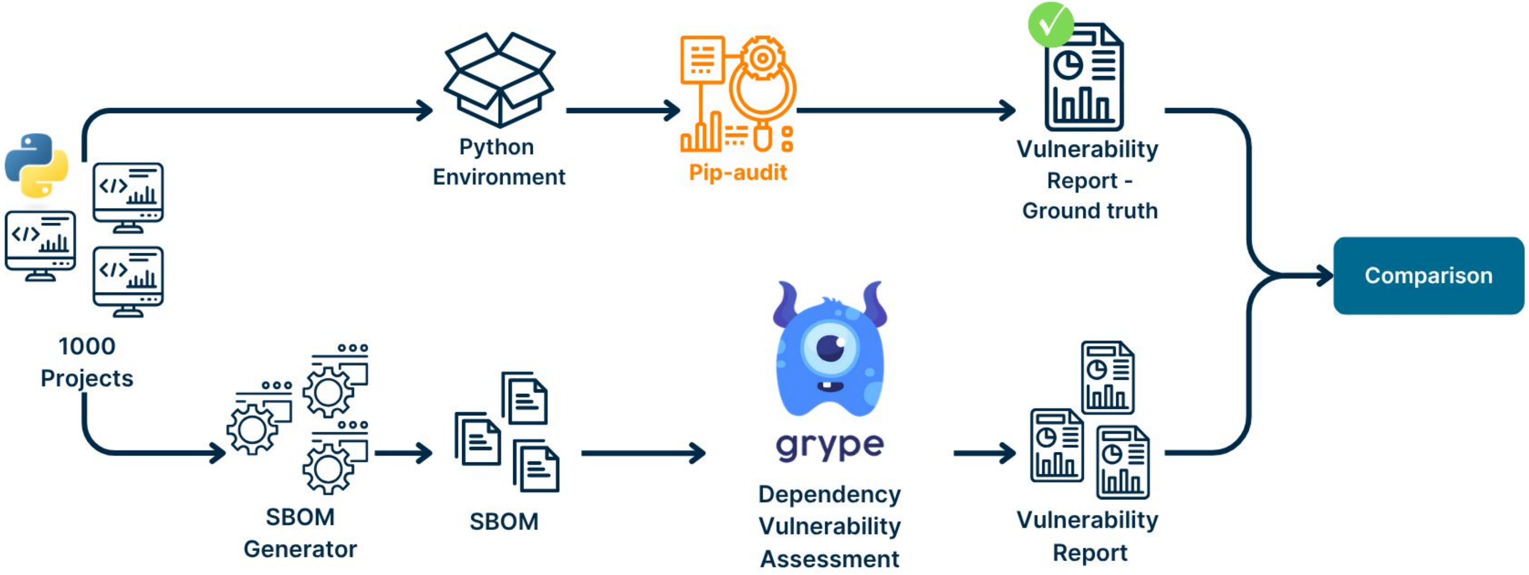


Evaluation RQ1 - Metrics

- **Precision**: The proportion of correctly identified vulnerabilities out of all the vulnerabilities that were identified.
  - **High Precision** means that when the system says "vulnerability found", it's usually right; fewer false alarms.

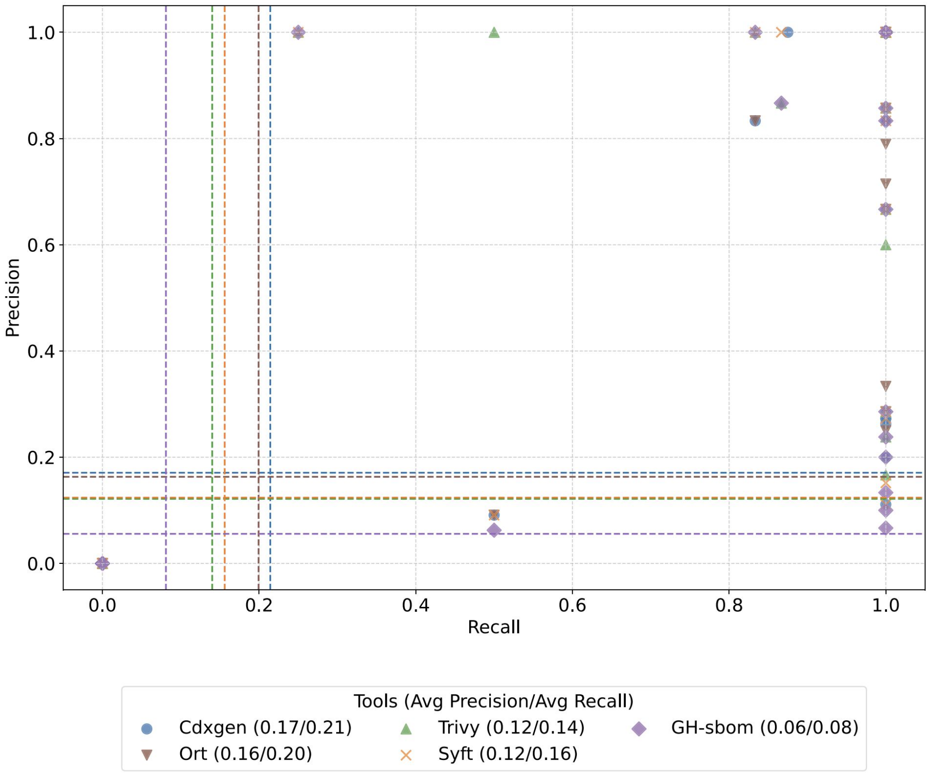
- **Recall**: The proportion of correctly identified vulnerabilities to all actual vulnerabilities in the ground truth.
  - **High Recall** means that the system is catching most of the vulnerabilities that are present; missing very few.

Evaluation RQ1 - Final setup



RQ1 results

- **Low precision**: Most of the identified vulnerabilities are not actually present in the software's dependencies.
- **Low recall**: Most of the vulnerabilities in the software's dependencies are not found.



RQ1 - Takeaway

Current state-of-the-art SBOM generation tools do not properly generate SBOMs for vulnerability assessment of Python projects.

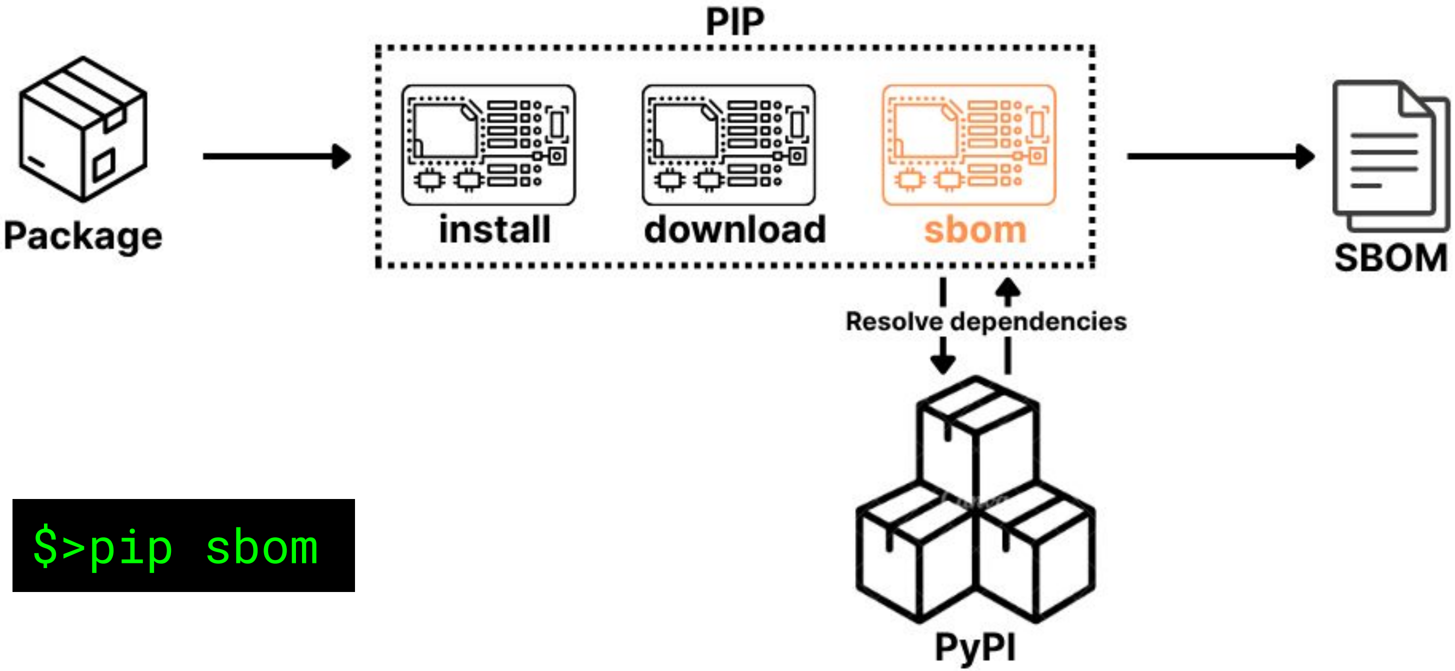
Novel Approach (RQ2)

RQ2 - PIP-SBOM description of the implementation

- Python projects have an heterogeneous structure
  - Multiple metadata files
- SBOM generators cannot keep up with such variety
- We want to bypass this issue by including the SBOM generation during the packaging
- PIP is the Python default package manager and supports the majority build tools

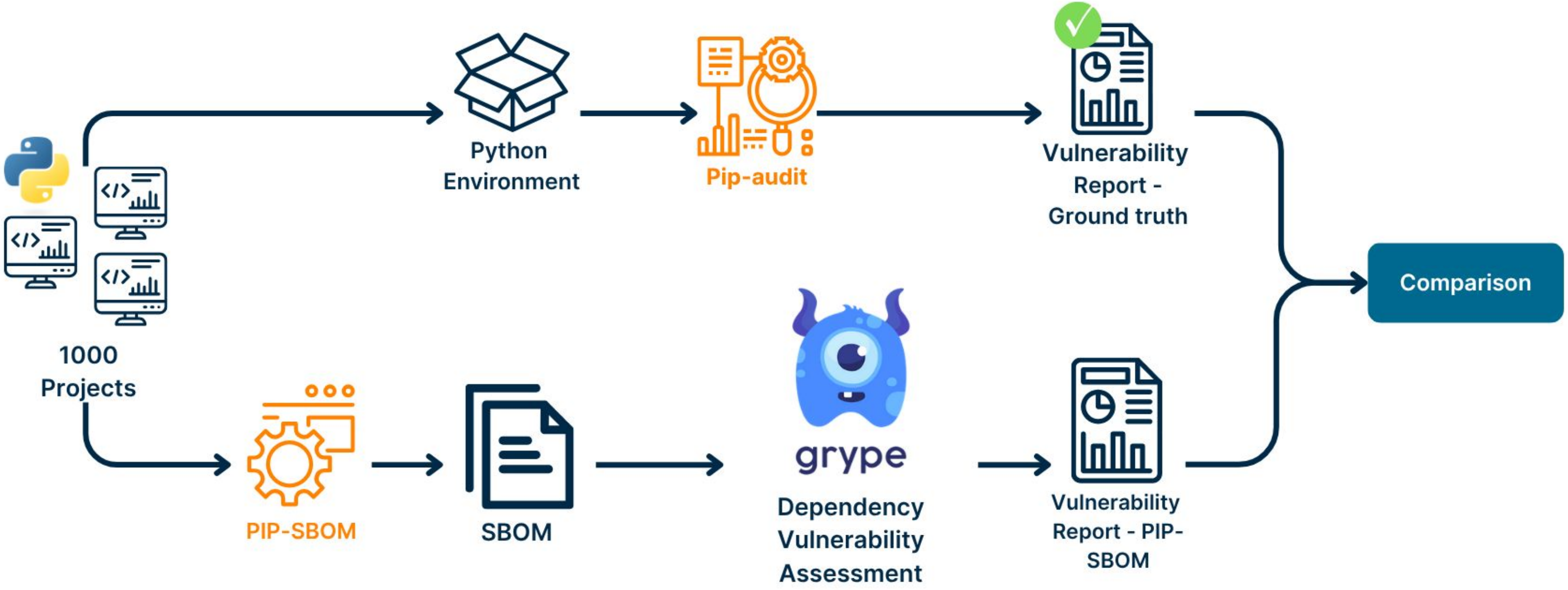
  → **we embed SBOM generation into PIP**

RQ2 - PIP-SBOM



```
$>pip sbom
```

RQ2 -evaluation


1000 Projects
Python Environment
Pip-audit
Vulnerability Report - Ground truth
PIP-SBOM
SBOM
grype
Dependency Vulnerability Assessment
Vulnerability Report - PIP-SBOM
Comparison

RQ2 - results

- Drastically better performance in the vulnerability assessment process
- Lower false positive rate is really important for a good risk management

	cdxgen	ORT	Syft	Trivy	GH-sbom	PIP-SBOM
Avg Precision	17.08%	16.31%	12.39%	12.17%	5.57%	**80.95%**
Avg Recall	21.42%	19.93%	15.61%	14.01%	8.10%	**80.26%**
False Pos	978	449	926	893	2793	**47**
False Neg	5	10	21	21	29	**3**

RQ2 - Takeaway

PIP can be extended by re-using most of its code to generate an SBOM that drastically improves the vulnerability assessment results.

Conclusion and future directions

- The SBOM is necessary for many tasks in software development, vulnerability assessment proves that once again.
- Current SBOM generators need to adapt to the ecosystem’s rules.
- The involvement of the ecosystem’s management in the SBOM generation process is fundamental for better employment of this artifact.
- Can our approach be extended to other ecosystems?

Thank you!

Questions?