SIT: An accurate, compliant SBOM generator with incremental construction

Changguo Jia[1 2 3], Nianyu Li[3], Kai Yang[3], Minghui Zhou[1 2†]
[1]School of Computer Science, Peking University, China
[2]Key Laboratory of High Confidence Software Technologies, Ministry of Education, China
[3]Zhongguancun Laboratory, China
jiachangguo@stu.pku.edu.cn, li_nianyu@pku.edu.cn, yangkai@zgclab.edu.cn, zhmh@pku.edu.cn

***Abstract*—SBOM (Software Bill of Materials) is a comprehensive list of components, relationships and metadata associated with software, essential for ensuring software component transparency in the software supply chain. The complexity of SBOM and the massive workload of writing SBOMs call for the assistance of automation. However, existing automated tools excessively rely on parsing dependency manifest and source code without verifying the accuracy of the information. Worse, existing SBOM generators sometimes fail to yield a specification-compliant SBOM. Additionally, existing SBOM generators can not compose a complete SBOM with information that developers know best and entries hidden in the dependencies' metadata in one go. To address the inaccuracy, non-compliance and incompleteness issues of SBOM generation, we propose SIT, an accurate, compliant SBOM generator with incremental construction. Through incremental construction, SIT aggregates manually maintained SBOMs and dependency SBOMs and exports SBOMs for editing, enhancing the correctness and completeness of SBOMs. This capability is built on SBOM IR, a flexible intermediate format that consolidates essential information and acts as a bridge for software representations. By integrating SBOM IR with official SBOM JSON schemas, SIT ensures all generated SBOMs are compliant to SBOM specifications. Additionally, SIT enhances SBOM accuracy with cross-validation, resolving inconsistencies with the real environment. SIT is publicly available at https://github.com/osslab-pku/SIT, and a demonstration video can be found at https://youtu.be/LbzslijVPLc.**

***Index Terms*—Software Maintenance, Software Process, SBOM**

I. Introduction

Modern software applications often depend on numerous third-party components [1], which can introduce security vulnerabilities that affect the final product [2]. SBOM (Software Bill of Materials) addresses this issue by recording detailed information about the software, including its metadata and dependencies, thus enhancing supply chain transparency [3].

To have humans producing SBOMs by hand is an unreasonable, time-consuming, and error-prone task. Yet, the full automation of SBOM production is a process that poses many challenges [4]. Existing SBOM generators suffer from limitations in the following aspects. **Incompleteness**: Some data fields, such as *homepage*, cannot be captured solely by SBOM generation tools. As a result, incomplete SBOMs can hinder software transparency and effective management. Merging well-maintained SBOMs or exporting sub-SBOMs for modification offers a practical solution, yet tools for merging and exporting are currently unavailable. **Non-compliance**: Due to complex SBOM specifications and unreliable third-party libraries, many tools fail to generate compliant SBOMs. Such SBOMs compromise trust of users, even leading to potential legal issues. **Inaccuracy**: Research indicates significant performance variations among different SBOM tools in their dependency detection accuracy in complex supply chains. This inconsistency can lead to inaccurate SBOMs, which undermines the security and transparency of the software supply chain [5].

To address these challenges, we propose SIT, which enhances SBOM generation by incremental construction, SBOM IR (Intermediate Representation) and validation against development environment. SIT focuses on Python ecosystem, one of the top-three languages by most notable metrics. Through incremental construction, SIT leverages well-maintained SBOMs and manually exported corrections to produce the most complete SBOMs. SIT also introduces SBOM IR and makes format conversions through official JSON schemas, ensuring the compliance of produced SBOMs. The accuracy of generated SBOMs is further improved by performing double cross-validation on dependencies and supporting all mainstream package managers.

In particular, based on automatically generated SBOMs, SIT incrementally constructs more complete SBOMs by merging well-maintained SBOMs that provide additional verified information and exporting sub-SBOMs for modifications and enhancements. In order to merge and export SBOMs, SIT constructs SBOM relationship graphs based on directionality and transitivity of relationships, which can then be modified and recombined to form a new SBOM. In this way, SIT can iteratively make full use of existing detailed SBOMs to produce the most complete SBOMs.

SIT's SBOM IR is built on the widely adopted SPDX and CycloneDX specifications, enabling developers to fill out entries without comprehensive knowledge of SBOM specifications. By integrating SBOM IR with official SBOM JSON schemas, SIT successfully establishes a standardized process for format conversion and data integration with schema validation, ensuring that no non-compliant SBOM documents are produced.

To mitigate the accuracy limitation of existing SBOM gen-

† Corresponding Author

erators, SIT integrates all the information, including package managers, source code, development environments, and local pip caches. In addition, SIT performs cross-validation on dependencies to ensure SBOM accuracy.

To check the completeness, compliance and accuracy of SIT's SBOM production, we perform a series of automatic and manual evaluations on the generated SBOMs.

In summary, the main contributions of this paper are as follows:

- We propose SIT, which incrementally constructs complete SBOMs by merging or exporting them, addressing data fields that are difficult to analyze. By integrating SBOM IR with official SBOM JSON schema validation, we unify and simplify SBOM generation process and ensure the compliance of resulting SBOMs. Additionally, we enhance SBOM generation accuracy by performing double cross-validation on dependencies.
- We've made SIT open source and packaged it in Docker for user convenience. Additionally, we design an experiment to demonstrate the accuracy of SBOMs generated by SIT, and manually verify the completeness and compliance of SBOMs after incremental construction.

II. Background and Related work

In the landscape of SBOM tools, SBOM generation is the most commonly supported feature [6]. Our research focuses on tools that generate SBOMs for Python packages. Despite Python's prominence, existing SBOM generation tools for Python face the following challenges:

- **Incompleteness**: Certain data fields are difficult for SBOM generation tools to capture, causing further incompleteness.
- **Non-compliance**: SBOMs generated do not comply with the target SBOM specification.
- **Inaccuracy**: SBOMs generated by these tools contain dependencies that are inconsistent with the dependencies installed in the real environment.

Since dependency management is one of the main use cases for SBOMs [7], it's important that dependencies recorded in SBOM are accurate. Unfortunately, SBOMs generated by current tools often deviate significantly from the actual dependencies, with accuracy of less than 75%. A detailed examination reveals the following issues with existing SBOM tools:

- Generators often confuse test, build, and development dependencies with runtime dependencies, as seen in tools like *cdxgen*.
- Packages not present in the development environment frequently appear in the SBOM, particularly in SBOMs generated by *cdxgen*.
- The *requirements.txt* file may list either all dependencies or only direct ones, but some generators, like *sbom-tool*, treat all entries indiscriminately as direct dependencies.

All these issues stem from the fact that current tools primarily rely on package managers and source code without validating the accuracy of all these information.

Additionally, we find that existing tools fail to analyze all possible package managers in Python. For example, PDM, the third most popular Python package manager on GitHub by star count, is not supported by any tool. This oversight can potentially cause inconvenience to developers.

Even worse, SBOMs generated by these tools do not comply with SBOM specifications. For instance, in CycloneDX SBOMs generated by *Scancode*, *value* in *properties* may sometimes be a list, whereas the CycloneDX specification mandates it should be a *string*. Such discrepancies introduce significant security risks during the use and distribution of SBOMs.

After a thorough investigation, we find that when it comes to integrating information into required SBOM formats, these tools primarily rely on two methods:

1) Developers constructing custom SBOM libraries themselves.
2) Using third-party SBOM libraries mainly from SBOM tool center.

Due to developers' limited knowledge of SBOM specifications, the first method often leads to errors, as seen with *Scancode*'s issues. Contrary to expectations, the second method also introduces errors. For instance, *cyclonedx-python-lib* fails to support *components* and *services* in *metadata.tools*.

Essentially, both approaches translate the SBOM specification into a second-hand tool before use, which inherently increases the risk of errors in the resulting SBOM documents.

As a supplement, certain fields, such as the *homepage*, are indeed challenging for SBOM generation tools to analyze. According to our research, none of the SBOM generation tools we test are able to successfully identify the *homepage* for all packages.

III. Tool Features

In order to address the incompleteness, non-compliance and inaccuracy of generated SBOMs mentioned before, SIT has made optimizations targeting each of these three aspects. Incremental construction is performed by SIT to enhance the completeness of SBOMs. The combination of SBOM IR and official SBOM schema validation ensures the production of compliant SBOMs. Double cross-validation of dependencies improves SBOM accuracy.

A. Incremental Construction

SIT achieves incremental construction by exporting sub-SBOMs for modification and merging well-maintained SBOMs. When exporting and merging, SIT first analyzes given SBOM documents by constructing an internal set of component relationship graphs for each document. In this way, issues of SBOM merging and exporting are translated into operations such as modification and deletion of relationship graphs.

Due to the complexity of relationship fields in SBOMs (e.g. SPDX has 45 types of relationships), this task presents a substantial challenge. We begin by collecting all potential relationship types found in existing SBOM standards, such as

dependsOn and ancestorOf. Then, we classify these relationships based on their properties as follows:

1) Directionality
   When an SBOM includes a relationship like *A ancestorOf B*, A must be included in B's SBOM, but not vice versa. Such relationships are considered unidirectional, from B to A. Of course, there are also bidirectional relationships requiring both components to be included in each other's SBOMs.
2) Transitivity
   When an SBOM states *A dependsOn B* and *B dependsOn C*, then B and C should be included in A's SBOM. This means that such relationships are transitive, making C a transitively reachable node from A. For transitive relationships, SBOM for root node A should include all of A's transitively reachable nodes.

Based on these properties, SIT constructs a set of relationship graphs for each SBOM. When exporting an SBOM, SIT uses BFS (Breadth First Search) to extract the subgraphs of specified node ID. It then organizes all the information and exports the SBOM. When merging an SBOM, SIT first constructs the set of relationship graphs from the given SBOM document. Then it compares this new set with the original relationship graph set. Through operations such as modifying nodes or changing edges, SIT merges the original graphs with the second graphs. After consolidating the information, it generates the final set of graphs, from which the resulting SBOM can be constructed.

B. SBOM IR and Schema Validation

Considering the inconvenience caused by complex SBOM specifications, we design SBOM IR. SBOM IR is a transitional and informal form that contains all the data fields required by common SBOM formats[1]. Based on SPDX and CycloneDX, the most widely used SBOM specifications [8], we construct a one-to-one mapping between SBOM IR and SBOM specifications to streamline compatibility and integration.

To ensure the accuracy of the IR, we carefully compare all the data fields across SBOM specifications. For those that are common fields in all specifications, we will keep them as reserved IR data fields. When it comes to similar data fields across specifications, we will make careful distinctions within the IR. For example, *cdx2spdx* tool incorrectly maps CycloneDX's *supplier* to SPDX's *creators*, despite their different meanings. SBOM IR introduces *creator* with detailed classifications of data types to accurately map to SPDX's *creators* and CycloneDX's *manufacturer*, *authors*, and *tools*, ensuring precise correspondence and strict compliance.

When encapsulating the mapping between IR, SPDX and CycloneDX into classes, we use the SPDX and CycloneDX JSON schemas provided by SBOM standards developers. This approach allows us to validate whether a given SBOM document complies with the required specifications, thus preventing the creation of invalid SBOM documents. In addition, we provide a Pydantic model and JSON schema for SBOM IR to help developers verify whether they have filled in data fields accurately.

[1]We focus on SPDX and CycloneDX

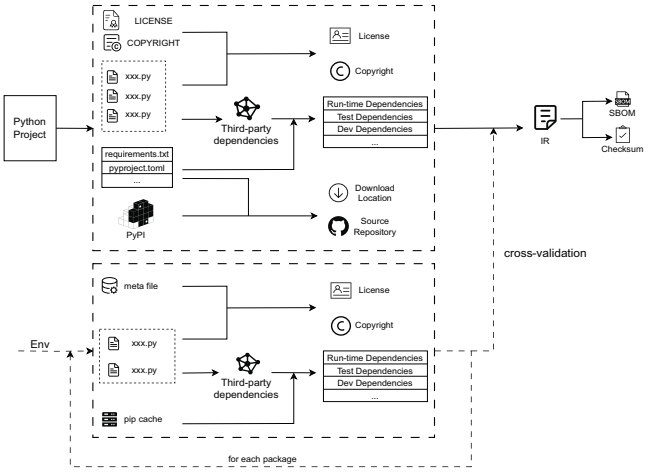

Fig. 1. Overview of SIT's SBOM generation pipeline.

SBOM IR serves as the basis of various tasks, including SBOM generation, conversion, and mergence. For example, SBOM IR enables conversion between different SBOM formats. When converting SBOM formats, SIT first converts the given SBOM into SBOM IR, then converts the IR into targeted SBOM format.

C. Double Cross-Validation of Dependencies

SIT's SBOM generation pipeline is represented as a flow chart in Figure 1. Information from package managers, source code, development environment and pip caches is collected altogether and then cross-validated twice, resulting in accurate and reliable SBOMs.

When analyzing Python packages, SIT starts by statically scanning Python source code to identify dependencies, which are then cross-validated with package managers. Considering the lag of package managers [9], we prioritize dependencies from the code scan. Based on naming conventions and information provided by package managers, dependencies are categorized into types like runtime dependencies and test dependencies.

After completing the preliminary data collection, we proceed to cross-validate these details against the actual development environment, which involves identifying and excluding any software packages that do not exist in the development environment. Through cross-validation, we ensure that the project dependency information is fully aligned with the real-world development settings.

IV. Evaluation

To demonstrate that SBOMs generated by SIT are accurate and SBOMs incrementally constructed by SIT are compliant and more complete than before, we evaluate them seperately. Datasets, evaluation code and results are open source on Zenodo[2]

[2]https://zenodo.org/records/13882428

A. Automatically Generated SBOMs

To evaluate the accuracy of dependencies in generated SBOMs, we first compile a list of SBOM generation tools for comparison. Initiating our search on GitHub using the topic "SBOM", we filter the tools based on the following criterias:

1) Exceed a thousand stars.
2) Capable of generating an SBOM that includes project dependencies.
3) Support analysis of Python projects.
4) Be an open-source project.
5) Can be run as a command-line tool.
6) Can generate SBOMs independently without relying on the API of other tools.

This process yields 3 SBOM tools: *Syft*[3], *sbom-tool*[4], and *cve-bin-tool*[5]. Additionally, we select two tools that works with Python from the official SPDX and CycloneDX tool centers: *cdxgen*[6] (the one with the most stars) and *cyclonedx-python*[7] (the only tool focused on the Python ecosystem). The latest stable releases as of August 2024 are used for the evaluation.

Eight GitHub projects were selected as the subjects for tool analysis: five projects that depend on more than 60 packages and three projects with fewer dependencies. By including both large and small projects, we aim to comprehensively test the accuracy of dependency analysis across different SBOM tools.

To compare dependencies in SBOMs generated by those tools and SIT, we use `pipdeptree`, a highly recommended tool for analyzing Python environment dependencies according to Stack Overflow. After getting the results analyzed by `pipdeptree`, we also manually review and correct dependencies analyzed by `pipdeptree` to minimize errors. Using these dependencies as ground truth, we compare them with the dependencies included in the SBOMs generated by selected SBOM tools.

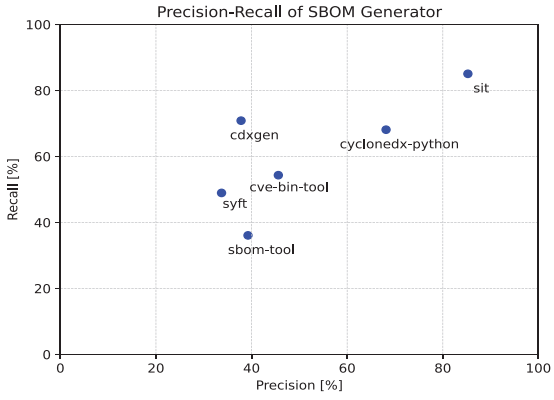

Fig. 2. Precision and recall of SBOM generators.

[3]https://github.com/anchore/syft
[4]https://github.com/microsoft/sbom-tool
[5]https://github.com/intel/cve-bin-tool
[6]https://github.com/CycloneDX/cdxgen
[7]https://github.com/CycloneDX/cyclonedx-python

The PR graph is presented in Figure 2. It clearly demonstrates that SIT achieves significantly higher accuracy than all current tools, outperforming them by approximately 10%.

B. Incrementally Constructed SBOMs

To assess the compliance and completeness of SBOMs generated through incremental construction, we first collect 10 SBOMs from the official SPDX repository and 16 from CycloneDX, taking them as samples to test SIT. Then we perform merging and exporting seperately on these 26 SBOMs, getting 650 merged SBOMs and 26 exported SBOMs. Two authors individually check and validate the resulting SBOMs and the original SBOMs. All resulting SBOMs are compliant to specified SBOM specification and contain necessary relationships, components and metadata. No issues are found.

V. Conclusion

We design SIT which enhances SBOM generation by incremental construction, SBOM IR and double cross-validation. SIT greatly improves the completeness, compliance and accuracy of produced SBOMs. Looking ahead, we aim to support more programming languages in SIT and continuously update SBOM IR to support additional SBOM specification versions.

VI. Acknowledgement

This work is sponsored by the National Natural Science Foundation of China 62332001 and the SBOM Platform Project for Open Source Community.

References

[1] R. Cox, "Surviving software dependencies," *Communications of the ACM*, vol. 62, no. 9, pp. 36–43, 2019.
[2] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*. Springer, 2020, pp. 23–43.
[3] T. N. Telecommunications and I. Administration, "The minimum elements for a software bill of materials (sbom)," 2021. [Online]. Available: https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
[4] M. Balliu, B. Baudry, S. Bobadilla, M. Ekstedt, M. Monperrus, J. Ron, A. Sharma, G. Skoglund, C. Soto-Valero, and M. Wittlinger, "Challenges of producing software bill of materials for java," *IEEE Security & Privacy*, 2023.
[5] M. F. Rabbi, A. I. Champa, C. Nachuma, and M. F. Zibran, "Sbom generation tools under microscope: A focus on the npm ecosystem," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, 2024, pp. 1233–1241.
[6] M. Mirakhorli, D. Garcia, S. Dillon, K. Laporte, M. Morrison, H. Lu, V. Koscinski, and C. Enoch, "A landscape study of open source and proprietary tools for software bill of materials (sbom)," *arXiv preprint arXiv:2402.11151*, 2024.
[7] T. Stalnaker, N. Wintersgill, O. Chaparro, M. Di Penta, D. M. German, and D. Poshyvanyk, "Boms away! inside the minds of stakeholders: A comprehensive study of bills of materials for software systems," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.
[8] S. Nocera, S. Romano, M. Di Penta, R. Francese, and G. Scanniello, "Software bill of materials adoption: a mining study from github," in *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2023, pp. 39–49.
[9] Y. Peng, R. Hu, R. Wang, C. Gao, S. Li, and M. R. Lyu, "Less is more? an empirical study on configuration issues in python pypi ecosystem," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.