Software Bill of Materials in Open Source C/C++ Projects: An Exploratory Study

Hangbo Zhang
Advisor: Dr. Anthony Peruma
MS Plan B / ICS 699, Spring 2024

Introduction

In today's software-centric landscape, comprehending the intricate interplay of dependencies within a project stands as a pivotal pillar for ensuring security, compliance, and transparency. As software ecosystems burgeon in complexity, the imperative for robust documentation of these dependencies has never been more pronounced. This is precisely where the Software Bill of Materials (SBOM) assumes its significance.

Essentially, the SBOM functions as a comprehensive ledger, cataloging all components, libraries, frameworks, and software artifacts incorporated within a project. Its utility lies in furnishing invaluable insights into the project's composition, empowering developers, stakeholders, and security professionals to navigate vulnerability management, licensing compliance, and risk mitigation strategies with informed precision

This work serves as an instructional blueprint for conducting an SBOM study on C/C++ project hosted on GitHub. Through meticulous analysis, extraction, and documentation of dependencies and associated metadata, our objective is to address the following research questions:

Research Questions

RQ1: How frequently do developers update project dependencies?
RQ2: What are the common libraries that undergo changes?
RQ3: What are the reasons behind these changes?

Methodology

The methodology employed for establishing the database aimed to ensure systematic organization and comprehensive coverage of SBOM information extracted from GitHub repositories. Through meticulous project selection criteria, data retrieval procedures, and database construction steps as shown in Figure 1, we laid the foundation for a robust dataset conducive to in-depth analysis and inquiry.

Project Selection

The selection of projects for inclusion in the database followed specific criteria to ensure relevance and diversity. We are using the SEART, a GitHub Search Engine to sample repositories to use by using several combinations of selection criteria, to get the projects from GitHub based on the following criteria: C/C++ projects, at least 100 releases, not a fork, last commit within last 6 month.

Data Retrieval

To acquire SBOM information spanning between releases, we employed the GitHub REST API as our primary data collecting method such as releases, tags, dependency reviews.

Database Construction

In order to facilitate systematic analysis and storage of SBOM data extracted from GitHub repositories, we constructed a relational database using SQLite with original 4 tables.

Data Analysis

In order to effectively address our research inquiries, we adhere to a meticulous preprocessing protocol o meticulously filter out extraneous data artifacts. Then pull the data out of the database to conduct the analysis such five number summary, common library finding, n-gram of the message.

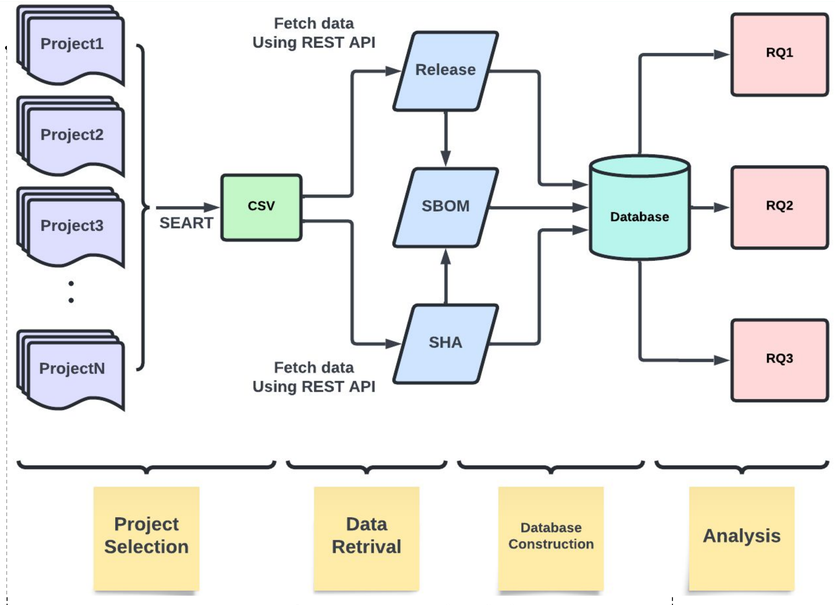

Figure 1: Methodology

Count	Min	Q1	Median	Q3	Max
number of changes for all projects					
314	0	0	0	8	613
number of changes without zero change projects					
144	1	3	9	22.5	613
release gaps					
3002	1	1	2	5	462

Table 1: Five Number Summary

Dependency	Added		Removed	
	Usage	Projects	Usage	Projects
(actions)/checkout	1746	125	947	94
()/upload-artifact	488	82	259	58
()/cache	274	41	111	23
()/setup-python	272	30	147	21
()/download-artifact	188	46	88	26
junit	144	3	98	3

Table 2: Occurrence of Top 6 Common Libraries

Trigram	Count	Percentage
identity integration test	658	0.08%
fix memory leak	618	0.07%
version bump package	554	0.06%
c coverity cid	477	0.06%
fix use free	427	0.05%
fix refcount leak	335	0.04%
publish backend test	320	0.04%
fix error handle	307	0.04%
integration test logger	304	0.04%
fix data race	297	0.03%
others	858364	99.58%

Table 3: Trigram of notes/messages

Results

In Table 1, we observe a right-skewed pattern for both number of releases with changes and release gap, indicating that while the majority of projects undergo minimal changes in each release with short intervals, a subset of projects experiences more significant alterations. This variability underscores the diverse nature of dependency updates across repositories

Table 2 shows widespread use of GitHub Actions workflows for CI/CD, testing, and automation. Additionally, the presence of popular libraries such as "junit" underscores their critical role in software development. Frequent updates in key dependencies reflect dynamic CI/CD pipelines, emphasizing ongoing workflow optimization.

Common unigrams like "fix" and "add" indicate a focus on issue resolution and feature addition, while specific bigrams and trigrams highlight nuanced concerns such as "bluetooth controller" and "fix memory leak" as shown in Table 3. Platform-specific modifications and version management activities underscore the diverse motivations driving development efforts.

Through the exploration of three research questions, we gained insights into the frequency of dependency updates, common libraries undergoing changes, and the reasons behind changes.