Understanding Similarities and Differences between Software Composition Analysis Tools

Pranet Sharma
Boston University

Zhenpeng Shi
Boston University

Şevval Şimşek
Boston University

David Starobinski
Boston University

David Sastre Medina
Red Hat

***Abstract*—**
Software Composition Analysis (SCA) plays a key role in ensuring supply chain security, by helping identify known security vulnerabilities in open source libraries. This work reviews several popular SCA tools and compares their key functionalities based on a set of objective criteria.

1. Introduction

Many companies embed third-party open source code into their production software. This code, if not thoroughly vetted, can potentially introduce vulnerabilities and pose security risks. According to the 2023 annual Open Source Security and Risk Analysis report by Synopsys[1], nearly 96% of 1,703 codebases used in key industries contained open source code, and over 84% of them contained at least one known open source vulnerability. Comprehensively addressing open-source vulnerabilities is therefore critical for modern software development.

Software Composition Analysis (SCA) is a security practice that automates the process of open source component management. In order to assist the SCA process, many tools have been designed and developed. SCA tools can detect open source dependencies used by a software application, as well as identify and assess the known vulnerabilities introduced by the dependencies.

SCA tools provide a number of features such as vulnerability detection, vulnerability mitigation, and license compliance. SCA tools typically scan the code base of software applications in order to find known vulnerabilities that potentially exist in the application, such as vulnerabilities listed in the Common Vulnerabilities and Exposures (CVE)[2]. In order to assess and mitigate the vulnerabilities, the tools often provide additional relevant information about the detected vulnera-

[1]https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html

[2]https://cve.mitre.org/

Published by the IEEE Computer Society

bilities, such as their mapped weaknesses from the Common Weakness Enumeration (CWE)[3] and their severity scores under the Common Vulnerability Scoring System (CVSS)[4]. Software developers and security practitioners can leverage these tools to reduce the number of open source vulnerabilities present in their code.

The behavior and outputs of SCA tools differ in many subtle ways, which are currently not obvious to software developers. The purpose of this study is to shed light on key similarities and differences between popular SCA tools. Toward this end, we analyze their workflows and compare their functionalities based on the following objective criteria (further justification about these criteria is provided in Section 5):

1) **Software Development Life Cycle (SDLC) Integration.** Does the tool integrate with commonly used developer tools?
2) **Vulnerability Library.** What libraries are used for vulnerability detection and assessment?
3) **Automatic Remediation.** Does the tool automatically fix vulnerabilities? What kind of vulnerabilities can the tool fix?
4) **Security Report Format.** What format does the tool use to report the findings about the vulnerabilities?
5) **Availability**. Is the tool an open source product or not? Is the tool free to use?
6) **Language Support.** What programming languages can the tool detect vulnerabilities in?
7) **Handling false positives**. Does the tool attempt to reduce the rate of false positives it generates when reporting vulnerabilities?

Our work makes the following contributions in this context:

- **An overview of popular SCA tools**. We discuss the features as well as the workflows of the tools, which can help software developers and security practitioners in selecting the appropriate tool(s) for their use.
- **Demonstration of the workflow analysis of the SCA tools using several open source project examples**. We highlight the similarities and differences between the SCA tools based on the results of this workflow analysis. We further summarize key takeaways (e.g., the tools show discrepancies regarding the scope of dependency packages to be analyzed).
- **Proposal of a set of criteria for the qualitative evaluation of SCA tools.** We use the proposed criteria, adapted from [1], to evaluate and compare the SCA tools. Suggestions on choosing appropriate SCA tools are accordingly given.

Note that our work specifically focuses on the vulnerability detection and mitigation capabilities of SCA tools. Although open source license compliance is also a part of SCA, there exist separate tools specifically designed for this purpose [2]. We stress that the purpose of this paper is not to conduct an exhaustive review of all the SCA tools available on the market, but rather contribute a framework to help compare their functionalities.

The rest of the paper is organized as follows. In Section 2, we discuss related work on SCA tools and open source component management in general. We introduce terminology and overview common SCA tools in Section 3. We conduct the workflow analysis of the tools, and summarize key takeaways from the analysis in Section 4. In Section 5, we evaluate the SCA tools based on our proposed criteria and discuss tool selection. We conclude the paper in Section 6.

2. Related Work

There exists a large body of research on addressing security vulnerabilities introduced by open source components in software applications. This includes (but is not limited to) characterization [3], [4], detection [5], assessment [6], [7], and mitigation [8], [9] of open source vulnerabilities. In particular, the work in [4] conducts an analysis of vulnerabilities in open-source libraries across 450 projects. This work investigates the types, distribution, severity, and persistence of the vulnerabilities. This work also provides recommendations, such as prioritizing the mitigation of Denial-of-Service (DoS) attacks.

In practice, software developers and security practitioners often use vulnerability scanning tools (e.g., SCA tools) to assist in vulnerability detection. As a result, it is valuable to understand and compare the performance and limitation of

[3]https://cwe.mitre.org/
[4]https://www.first.org/cvss/

these tools. Through an empirical study, the work in [10] investigates how different practices of software development can impact the performance of vulnerability scanners. It concludes that none of the vulnerability scanners under study has the ability to identify all four types of modifications across a list of 7,024 Java projects. It focuses on identifying the types of modifications to open-source dependencies such as forking, patching, and re-bundling that affect the performance of the vulnerability scanning tools and thus is a quantitative analysis of the performance of these tools. In comparison, we focus on a general qualitative comparison between the functionalities of SCA tools, in order to understand their similarities and differences.

Threat modeling starts from the early stages of development and relies on a system model to identify potentially vulnerable components in the design. The work in [11] presents a taxonomy of popular threat modeling tools and concludes with tool usage recommendations. Static application security testing (SAST) analyzes the source code of software applications and looks for vulnerabilities, such as injections. In [12], a novel differential benchmark is proposed for automatically evaluating SAST tools. In our work, we investigate and compare popular SCA tools. Among other uses, SAST tools are also used to help the security certification process for open source software. In [13], a delta certification-based tool, DeltAICert uses SAST tools as a first step in data evaluation. In delta certification, analysts leverage third-party tools to extract evidence, which is then assessed against established requirements. In comparison, SCA tools focus on open source dependencies used by software applications, and determine potentially vulnerable dependencies using knowledge from vulnerability databases.

Open source dependencies, such as packages from Maven and NPM, contribute to a large portion of vulnerabilities in software applications. The work in [14] conducts an extensive qualitative study about the decision-making of software developers in managing dependencies. The study shows possible trade-off between functionality and security, and suggests using high-level metrics for selecting mature and secure dependencies. In addition, it points out that SCA tools should alert only on dependencies relevant to vulnerabilities, and be careful not to break a project when suggesting vulnerability mitigation. This brings the challenges of handling false positive vulnerabilities, which we consider as a criterion for evaluating the SCA tools.

The work in [15] compares the analysis reports of nine industry-leading SCA tools on a large web application, OpenMRS. The criteria consist of the number of vulnerabilities reported, runtime, and ability to track unique dependency, package and vulnerabilities. Via a manual analysis of the vulnerability reports, the authors conclude that the accuracy of the vulnerability database is the key difference between the SCA tools. While the objectives of our study are broadly similar, we offer and evaluate complementary criteria for drawing comparisons between the different tools (see Table 2, in particular).

3. Overview of Software Composition Analysis

In this section, we introduce terminology and provide an overview of common Software Composition Analysis (SCA) tools. These tools are further analyzed and compared in subsequent sections of the paper.

3.1. Common Terminology

- **Dependency**: A dependency refers to a relationship between different software components within a project, where one component relies on the functionality or services provided by another.
- **Direct Dependency**: A direct dependency refers to a software component that is explicitly declared and directly used within a project.
- **Transitive Dependency**: A transitive dependency refers to a software component that is indirectly introduced into a project as a result of its reliance on other direct dependencies. In other words, if component A is a direct dependency of a project and component A, in turn, relies on component B, then B becomes a transitive dependency for the project.
- **Vulnerability Database**: A vulnerability database is a centralized repository systematically cataloging known software vulnerabilities. It provides crucial details for security professionals, developers, and

organizations, including the nature of vulnerabilities, affected components, severity ratings, and recommended remediation. In Section 3.2, we detail the different vulnerability databases utilized by each tool.

- **CI/CD**: CI/CD, or Continuous Integration/-Continuous Deployment, is a software development practice that emphasizes frequent and automated integration of code changes into a shared repository (Continuous Integration) and the automated deployment of applications to production environments (Continuous Deployment).
- **Manifest File**: A manifest file is a structured document that contains metadata and essential information about a project. It serves as a guide for the system to understand the project's structure, dependencies, configurations, and other relevant details.
- **Lock File**: A lock file is a file within a project that specifies the precise versions of dependencies used in that project. While the manifest file does list the direct dependencies for the project, the lock file is generated to list all direct and transitive dependencies, along with the specific versions of each.
- **SBOM**: An SBOM, or Software Bill of Materials, is a comprehensive inventory that details all the components, libraries, modules, and dependencies involved in the development and deployment of a software application. This includes both open-source and proprietary elements. An SBOM typically includes information about the version, origin, and licensing of each component, and may also highlight any known vulnerabilities.

3.2. Software Composition Analysis Tools

We now overview some popular SCA tools. We selected 5 prominent SCA tools which easily integrate into software development, support the Maven package manager and the Java programming language, and are either open source or are free versions of commercially available tools. The Snyk Open-Source CLI[5], GitHub Dependabot Core[6], OSV-Scanner[7] and OWASP Dependency-Check[8] Github repositories are popular with developers, featuring 4.8k stars, 4.4k stars, 5.9k stars and 5.9k stars respectively. Finally, RedHat CodeReady Dependency Analytics' VS Code IDE extension features nearly 2.2 million installs[9].

1) **Snyk Open Source**[10] is a tool developed by Snyk. It utilizes the Snyk Intel Vulnerability Database, a publicly accessible database that contains entries analyzed and verified by the Snyk security team. This team also engages in proprietary research to discover new vulnerabilities. The tool scans the manifest file of a project to create a hierarchical tree, which includes both direct and indirect dependencies, as well as points at which different packages were introduced.
2) **GitHub Dependencies Graph and Dependabot**[11] is a dependency analysis tool developed by GitHub for maintaining supply chain security. It utilizes the GitHub Advisory Database, which is a publicly accessible database that adds advisories from sources such as the NVD, and NPM Security Advisories Database. It automatically reads the dependencies explicitly declared in the manifest and lockfiles to generate both direct and transitive dependencies. The tool integrates with GitHub repositories and utilizes Dependabot, a tool developed by GitHub to monitor vulnerabilities in dependencies used in projects and to keep dependencies up to date by informing the user of any security vulnerabilities.
3) **OSV-Scanner**[12] is an open source vulnerability scanning tool developed by Google. The scanner obtains vulnerability information from the OSV database, a publicly accessible database that aggregates vulnerability information from a number of public databases including GitHub Advisory Database, PyPI Advisory Database, and Global Security Database. The tool parses lock files, SBOMs, and git directories to determine open source dependencies used by the software application.
4) **Red Hat CodeReady Dependency Analyt-**

[5]https://github.com/snyk/cli
[6]https://github.com/dependabot/dependabot-core
[7]https://github.com/google/osv-scanner
[8]https://github.com/jeremylong/DependencyCheck
[9]https://marketplace.visualstudio.com/items?itemName=redhat.fabric8-analytics
[10]https://docs.snyk.io/scan-application-code/snyk-open-source
[11]https://docs.github.com/en/code-security/dependabot/
[12]https://google.github.io/osv-scanner/

ics[13] is a dependency analytics tool developed by Red Hat. It is available as an IDE plugin for Visual Studio Code (VS Code), Eclipse Che, Red Hat CodeReady Workspaces, and IntelliJ-based IDEs. The tool utilizes the Snyk Intel Vulnerability Database in order to identify vulnerabilities. The tool works by automatically scanning manifest files when they are opened in the IDE.

5) **OWASP Dependency Check**[14] is an SCA tool that is developed by the OWASP foundation. The tool aims to detect publicly disclosed vulnerabilities in the project that is being scanned, using the NVD data feeds. The tool scans the project for dependencies, linking these dependencies to specific CPEs. These CPEs are thenused to list associated CVEs. The tool collects information about the vendor, product and version utilizing Analyzers which scan the files in the project. Dependency-check has a command line interface, a Maven plugin, an Ant task, and a Jenkins plugin.

4. Workflow Analysis of Software Composition Analysis Tools

In this section, we analyze the workflow of each tool and report results from that workflow analysis. Based on these results, we provide several takeaways regarding the functioning of the tools.

4.1. Selection of Open Source Projects

The analysis is conducted on five open source Java projects: Zerocode[15], Superword[16], Find-Sec-Bugs[17], Java Speech API[18], and PiTest[19].

All of the chosen projects utilize Maven to manage dependencies and the projects are developed using Java programming language. We have selected projects that show an adequate level of popularity on GitHub, based on the number of times the projects have been starred or forked. Moreover, we omitted any project that has no reported vulnerability among any of the tools. Two of the projects have higher number of vulnerabilities reported, while other projects are simpler projects that have smaller number of vulnerabilities, but are useful to show discrepancies between the selected tools (i.e. particular tool reports vulnerabilities others cannot, or useful to show similarities between tools).

[13] https://developers.redhat.com/blog/2020/08/28/vulnerability-analysis-with-red-hat-codeready-dependency-analytics-and-snyk
[14] https://owasp.org/www-project-dependency-check/
[15] https://github.com/authorjapps/zerocode
[16] https://github.com/ysc/superword
[17] https://github.com/find-sec-bugs/find-sec-bugs
[18] https://github.com/lkuza2/java-speech-api
[19] https://github.com/hcoles/pitest

4.2. Workflows

In this subsection, we detail the workflow for each tool. We choose the free version of the SCA tools for subsequent workflow analysis. All the tools introduced in Section 3 can be used for free, though Snyk Open Source has limitations on the free tier.

1) **Snyk Open Source.** The GitHub integration is utilized in order to conduct the analysis. After logging into the Snyk account and connecting a GitHub account with Snyk, Snyk Open Source automatically scans projects it had been given access to. The number of vulnerabilities in each project is displayed on the dashboard with the option to view a project in more detail. The vulnerability report for the core project of Zerocode is shown in Fig. 1. Snyk Open Source categorizes the vulnerabilities for each project based on their CVSS scores into four major categories: critical, high, medium, and low severity. Snyk Open Source specifies the vulnerability type, the package it is introduced in, the package it is fixed in, and exploit maturity. It also offers a detailed overview of the project's dependencies and vulnerabilities, and additional details about the path through which the vulnerability was introduced. The CVSS scores from multiple sources are shown, including Snyk, NVD, and open source vendors (e.g., Red Hat, SUSE), though the Snyk CVSS scores are preferred when it determines the severity of vulnerabilities. Snyk Open Source also provides links to the corresponding CVE, CWE, and Snyk Vulnerability Database entries. Finally, the option to open a pull request to fix the issue is present.
2) **GitHub Dependencies Graph and Dependabot.** GitHub Dependencies Graph and Dependabot are features that can only be enabled when using GitHub to host the application repositories. After enabling these fea-

tures, GitHub Dependabot raises alerts under the Security tab on the GitHub repository webpage. Fig. 1 shows the Dependabot alerts for Zerocode. The Dependabot alerts on vulnerabilities include the severity level of the vulnerability, the vulnerable dependencies, the package manager, and the manifest/lock file containing the vulnerability. When a specific vulnerability is selected, additional details can be viewed, including its CVSS score, CWE, CVE ID, GitHub Security Advisories Database (GHSA) ID, as well as the affected and patched versions of the affected dependency. It also provides the option to open a Dependabot security update which opens a pull request to fix the issue.

3) **OSV-Scanner.** The command line version of OSV-Scanner is utilized for the purpose of testing. It allows for specifying a particular manifest/lock file to scan or to recursively search through the project to automatically search for them and scan them. As shown in Fig. 1, OSV-Scanner displays a link to the OSV Database entry for each detected vulnerability, the ecosystem that it is a part of, the vulnerable dependencies with their versions being used, and the source of the dependency. Following the links provided by the report to the corresponding OSV Database entries, more information can be found about the detected vulnerabilities, including the source of the vulnerability, the corresponding CVE ID, the date the vulnerability was published, and the modification dates. OSV-Scanner also supports outputting the vulnerability information in the JSON format.
4) **Red Hat CodeReady Dependency Analytics.** The VS Code extension of Red Hat CodeReady Dependency Analytics is used for the evaluation. The results of the dependency analysis show up in a separate window in the IDE. The vulnerability report window for Zerocode is shown in Fig. 1. It lists dependencies with security issues by the severity of their related vulnerabilities. The tool prioritizes the vulnerabilities by the Snyk CVSS scores and displays it along with the number of direct and transitive vulnerabilities. When a particular dependency is selected, the tool will display the type of vulnerability, whether an exploit is available, the current package version, and the recommended version. The report window also has other tabs for detailed information on dependency details, licenses, and add-ons.
5) **OWASP Dependency Check** The command line interface for the OWASP Dependency Check tool is used for the purpose of testing the selected open source projects. The path to scan can be provided as a command line argument. The tool is able to output the results in various file formats including HTML, JSON, SARIF and XML. The HTML output format was used for our purposes and is displayed in Fig. 1. The tool displays the highest severity, CVE count, Confidence and Evidence count for each vulnerable dependency. Besides the summary, the tool lists detailed information about each vulnerable dependency including a description of the dependency and a list of all the associated published CVEs, CVSS information and CWE information.

4.3. Results

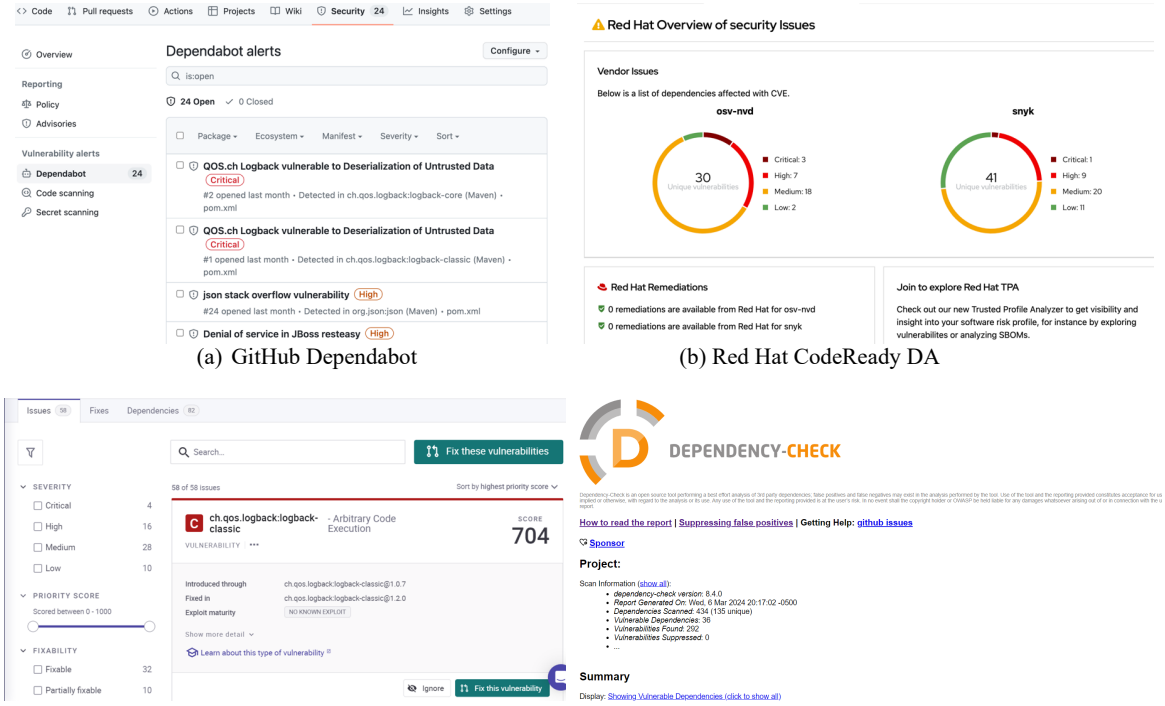
We find that the SCA tools differ in several key aspects. The results and takeaways from the workflow analysis of SCA tools and the case study are summarized in the following:

The SCA tools report different numbers of vulnerabilities, for the same project. As seen in Table 1, the tools differ in number of vulnerabilties reported as well as how many of those vulnerabilities are identical or unique.

The tools show discrepancies when it comes to the vulnerabilities reported, even when the same vulnerability database is utilized. For certain dependencies, the tools report different numbers of vulnerabilities. Moreover, even though the same vulnerability database is used by Snyk Open Source and Red Hat Dependency Analytics, we see that Snyk Open Source reports 3 Snyk exclusive vulnerabilities[20] Red Hat CodeReady Dependency Analytics reports only 2 Snyk exclusive vulnerabilities, for the Zerocode project.

Tools utilizing the same vulnerability database report relatively similar results. For example, Snyk and Red Hat CodeReady Dependency Analytics show the most similarity as

[20]Vulnerabilities exclusive to the Snyk Vulnerability DB do not have any corresponding CVE records.



(a) GitHub Dependabot

(b) Red Hat CodeReady DA

(c) Snyk Open Source

(d) OWASP Dependency Check

```
Scanned [redacted]/zerocode/pom.xml file and found 35 packages
+-----------------------------------+-----------+-----------------------------------------------+----------+----------+
| OSV URL (ID IN BOLD)              | ECOSYSTEM | PACKAGE                                       | VERSION  | SOURCE   |
+-----------------------------------+-----------+-----------------------------------------------+----------+----------+
| https://osv.dev/GHSA-vmfg-rjjm-rjrj | Maven     | ch.qos.logback:logback-classic              | 1.0.7    | pom.xml |
| https://osv.dev/GHSA-668q-qrv7-99fm | Maven     | ch.qos.logback:logback-core                 | 1.0.7    | pom.xml |
| https://osv.dev/GHSA-vmfg-rjjm-rjrj | Maven     | ch.qos.logback:logback-core                 | 1.0.7    | pom.xml |
| https://osv.dev/GHSA-288c-cq4h-88gg | Maven     | com.fasterxml.jackson.core:jackson-databind | 2.10.0   | pom.xml |
| https://osv.dev/GHSA-3x8x-79m2-3w2w | Maven     | com.fasterxml.jackson.core:jackson-databind | 2.10.0   | pom.xml |
| https://osv.dev/GHSA-57j2-w4cx-62h2 | Maven     | com.fasterxml.jackson.core:jackson-databind | 2.10.0   | pom.xml |
| https://osv.dev/GHSA-jjjh-jjxp-wpff | Maven     | com.fasterxml.jackson.core:jackson-databind | 2.10.0   | pom.xml |
| https://osv.dev/GHSA-rgv9-q543-rqg4 | Maven     | com.fasterxml.jackson.core:jackson-databind | 2.10.0   | pom.xml |
| https://osv.dev/GHSA-4jrv-ppp4-jm57 | Maven     | com.google.code.gson:gson                   | 2.6.2    | pom.xml |
| https://osv.dev/GHSA-5mg8-w23w-74h3 | Maven     | com.google.guava:guava                      | 23.0     | pom.xml |
| https://osv.dev/GHSA-mvr2-9pj6-7w5j | Maven     | com.google.guava:guava                      | 23.0     | pom.xml |
| https://osv.dev/GHSA-4gg5-vx3j-xwc7 | Maven     | com.google.protobuf:protobuf-java           | 3.13.0   | pom.xml |
| https://osv.dev/GHSA-g5ww-5jh7-63cx | Maven     | com.google.protobuf:protobuf-java           | 3.13.0   | pom.xml |
```

(e) OSV-Scanner

Figure 1: User interfaces for the SCA Tools analyzed in our case study.

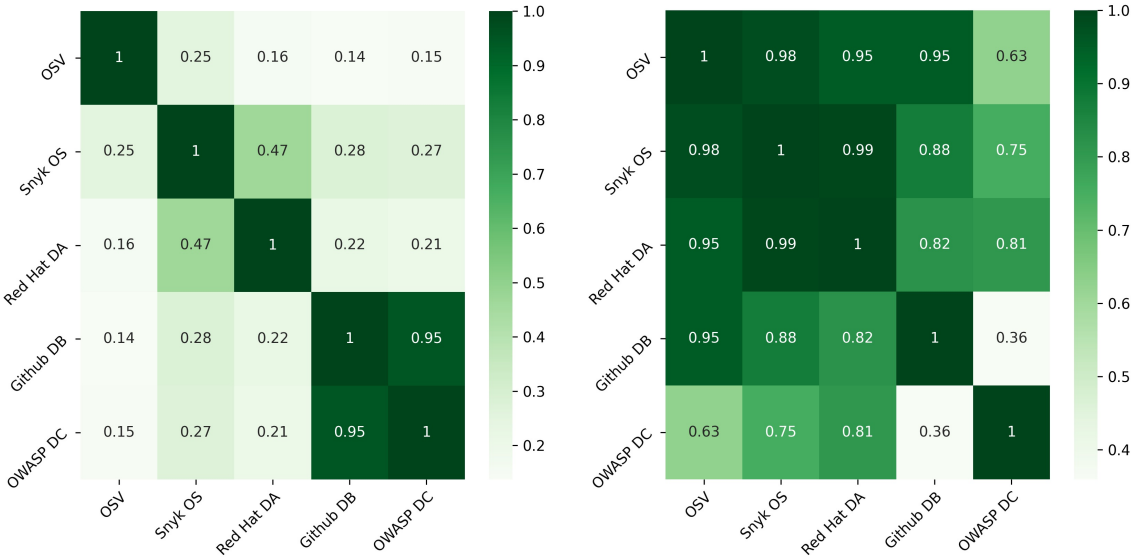
depicted in the correlation matrix in Figure 2b. Similarly, OSV Scanner and GitHub Dependabot report similar number of vulnerabilities. However, when analyze the number of common vulnerabilities reported among the tools, we see that there are major differences even when some tools rely on the same database, as discussed above.

The tools possess distinct approaches in defining the scope of packages being analyzed. For instance, GitHub Dependencies Graph and Dependabot and OSV-Scanner report vulnerability CVE-2022-23221 in the H2 Database Engine (*com.h2database:h2:1.4.19*), which is only a part of the testing scope. Red Hat CodeReady Dependency Analytics and Snyk Open Source do not report this vulnerability at all. GitHub Dependencies Graph and Dependabot identifies the vulnerability with the "Development" tag. An ideal SCA tool would allow developers to modify the scope of the analysis at runtime to determine whether development-only dependencies should be included or not.

All the tools fail to detect some transitive vulnerabilities. For instance, according to the Maven repository, the dependencies *commons-collections*, *org.apache.velocity:velocity* and *org.jboss.resteasy:resteasy-jaxrs* all transitively introduce the vulnerability *CVE-2022-23307* with a CVSS score of 8.8. However, none of the tools are able to detect this vulnerability.

Table 1: Number of total vulnerabilities and number of unique vulnerabilities for each project as reported by the tools analyzed in this case study.

Project	Total	OSV Scanner		OWASP DC		GitHub DB		Snyk OS		Red Hat DA	
		Total	Unique	Total	Unique	Total	Unique	Total	Unique	Total	Unique
Zerocode	119	33 (27%)	0	96 (80%)	34	32 (26%)	0	74 (62%)	11	31 (25%)	0
Superword	102	24 (23%)	11	12 (11%)	0	24 (23%)	0	75 (73%)	27	63 (61%)	14
Find-sec-bugs	8	2 (25%)	0	4 (50%)	4	2 (25%)	0	2 (25%)	2	0	0
Java Speech API	2	2 (100%)	0	0	0	2 (100%)	0	2 (100%)	0	2 (100%)	0
PiTest	17	0	0	0	0	2 (11%)	0	15 (88%)	4	13 (76%)	0



(a) Jaccard Index Matrix for the common vulnerabilities reported by each tool

(b) Pearson correlation matrix created using the number of reported vulnerabilities by each tool

Figure 2: Similarity matrices for the SCA tools: OSV Scanner, OWASP Dependency Check, Github Dependabot, Snyk Open Source and Red Hat Dependencies Graph

This is only one example of the vulnerabilities which are stated to be transitive by the Maven repository but do not appear in the results of any of the tools.

For the same vulnerability, the different tools may report different severity scores. For instance, for the same dependency *junit* with vulnerability *CVE-2020-15250*, Snyk Open Source and RedHat CodeReady Dependency Analytics report a low severity with a Snyk CVSS score of 2.9 whereas GitHub Dependabot reports a medium severity with CVSS score of 4.4. GitHub Dependencies Graph and Dependabot report the severity based on the GithHub Advisory Database CVSS score.

For the same dependencies, different SCA tools may report different results on whether the dependencies are vulnerable. For example, GitHub Dependencies Graph, Dependabot and OSV-Scanner uniquely identify dependency *com.h2database:h2:1.4.19* as vulnerable. In comparison, Snyk Open Source and Red Hat CodeReady Dependency Analytics do not identify this dependency as vulnerable.

Even though the same vulnerable packages are detected by some of the tools, this does not necessarily mean that the same vulnerabilities are reported. As seen in Figure 2a, when the Jaccard similarity coefficient is computed using a binary vector representing the tools reporting or not reporting a certain vulnerability, we see that only OWASP Dependency Check and Github Dependabot report consistent results, whereas others report many unique vulnerabilities others do not.

The results of the analysis were responsibly disclosed with the maintainers of each project.

5. Comparison of Software Composition Analysis Tools

In this section, we compare SCA tools based on the criteria introduced in Section 1. We opt for a qualitative approach in order to objectively compare the capabilities offered by these tools instead of their quantitative performance. We first justify our evaluation criteria, then evaluate and compare the five SCA tools introduced in Section 3 based on each criterion.

5.1. Justification of Evaluation Criteria

In this subsection, we discuss the rationale behind the evaluation criteria chosen. In [11], a set of evaluation criteria is proposed in order to evaluate commonly used threat modeling tools. The Linux Foundation[21] provides a guide to the evaluation of Software Composition Analysis tools [1]. We adapt metrics from the report and highlight features that we believe are important for the tools. The report provides a list of criteria such as knowledge base, ease of use, integration capabilities and reporting capabilities which we adapt to highlight important features of the SCA tools.

SDLC Integration. SCA tools are generally used in conjunction with various other software development tools. During typical software development, SCA must be performed regularly to ensure the security of the open-source software supply chain. Thus, interoperability of SCA tools with other tools proves essential. Integration of SCA tools into the CI/CD pipeline, issue-management systems and code version control systems can make them more convenient.

Vulnerability Library. The size, sources, reliability and level of transparency of the vulnerability library used by the tools can influence the choice of a tool. Some of these tools utilize public vulnerability databases and public dashboards to collect vulnerability information whereas some of the other tools contain proprietary vulnerabilities. The differences between the vulnerability libraries are thus important evaluation criteria.

Automatic Remediation. Automation of remediation of certain vulnerabilities is a key feature advertised by some SCA tools (e.g., Snyk Open Source). The extent to which this automatic process proves useful to developers is essential to evaluating these tools. Determining the scope of automation as well as ensuring that the automatic process does not end up breaking the dependencies of the project is essential to determining how much time these tools can potentially save during the software development cycle.

Security Report Format. The different modes through which the tool can output the list of vulnerabilities and other critical information is an important part of the evaluation as it allows developers to pick the tools based on the scope of their project. Tools that can output the projects as JSON may be more useful for developers who want to perform certain automations on the information, whereas developers who are more focused on individual use may desire easy-to-read dashboards with all the information readily available.

Availability. Developers need to consider whether the tool is open-source or commercially available, as well as the price of the tool, when making the decision of which tool to choose.

Language Support. The extent of package managers and languages supported by a tool can make or break a software developer's decision to use it. Projects with a very large scope may use multiple types of languages or package ecosystems and thus selecting a tool that meets all requirements is essential. For the purpose of our evaluation, we consider the top 10 programming languages (excluding HTML/CSS, Powershell, Bash, and SQL) and the top 5 package managers most used by developers according to the Stack Overflow 2023 Developer Survey[22]. Thus, we consider whether the Javascript, Python, Typescript, Java, C#, C++, C, PHP, Go, and Rust programming languages are supported and whether the NPM, pip, Yarn, Nuget and Maven package managers are supported. Only the languages explicitly listed as supported in the tools' documentation are counted here. In practice, more languages and package managers may be supported by the tools.

[21] https://project.linuxfoundation.org/hubfs/Reports/An-Open-Guide-To-Evaluating-Software-Composition-Analysis-Tools_V2.pdf?hsLang=en

[22] https://survey.stackoverflow.co/2023/

False Positives. False positive reported vulnerabilities are known to waste valuable development time. Due to the qualitative nature of this survey, our discussion of false positives mainly deals with the mitigation strategies utilized by the tools.

5.2. Results

We next report the results of our comparative evaluation.

SDLC Integration. Snyk Open Source provides repository integration with a multitude of version control tools such as GitHub, BitBucket, and GitLab. GitHub Dependencies Graph is available for GitHub hosted repositories. Snyk Open Source also offers CI/CD tools integration. Snyk Open Source and Red Hat CodeReady Dependency Analytics provide IDE integration services with most of the commonly used IDEs such as VS Code, Eclipse, and IntelliJ based IDEs. OWASP Dependency-Check is available as a Maven plugin, Gradle plugin and a Jenkins plugin.

Vulnerability Library. Snyk Open Source and Red Hat CodeReady Dependency Analytics utilize the Snyk Intel Vulnerability Database which covers all CVE records. Snyk also states that 40 percent of the database is proprietary and contains vulnerabilities not included in the CVE. OSV-Scanner aggregates vulnerability data from a number of public vulnerability databases and includes all CVEs. GitHub Dependencies Graph utilizes the GitHub Advisory Database which includes CVE records as well as advisories from a number of public databases and user reports. OWASP Dependency-Check utilizes vulnerability information primarily from the NVD. Other third party data sources such as the NPM Audit API, the OSS Index, RetireJS, and Bundler Audit are utilized for specific technologies.

Automatic Remediation. Snyk Open Source and GitHub Dependencies Graph provide automatic remediation by opening pull requests in order to fix licensing issues or issues that can be fixed by updates. If an upgrade is not available to fix the issue, Snyk Open Source offers patches in order to fix the issue. Red Hat CodeReady Dependency Analytics can advise users as to which version of the package to use. OSV-Scanner and OWASP Dependency-Check do not provide any automatic remediation support.

Security Report Format. Snyk Open Source allows the security report to be exported to JSON or SARIF file formats. OSV-Scanner gives the output in the form of a human-readable table format or the JSON file format. GitHub Dependencies Graph provides output on the GitHub website. Red Hat CodeReady Dependency Analytics outputs the vulnerabilities in the IDE. OWASP Dependency-Check provides output in the HTML, XML, CSV, JSON, JUNIT, SARIF and JENKINS.

Availability. Snyk Open Source is a commercial product that offers a free version with a limited number of monthly tests as well as paid versions. OSV-Scanner and OWASP Dependency-Check are open source tools that are free to use. GitHub Dependencies Graph and Dependabot alerts are commercial products that can be enabled for free. Red Hat CodeReady Dependency Analytics is available for free as IDE extensions. However, a Snyk account needs to be linked for learning details of the vulnerabilities exclusive to Snyk.

Language Support. Snyk Open Source states that it provides support for 8 out of the 10 programming languages most notably missing support for Typescript and Rust. It also provides support for all 5 package manager ecosystems. OSV states that it supports the Python and Go programming languages and it also supports the NPM, Maven and NuGet package manager ecosystems. GitHub Dependabot states that it supports 9 out of the 10 considered languages, missing support for Rust. It states that it supports 4 out of the 5 package managers with no support for the NuGet package manager ecosystem. RedHat CodeReady Dependency Analytics states that it currently supports the Maven and NPM package manager ecosystems. It also states that it supports the Golang and Python programming languages. OWASP Dependency-Check states that it provides support for the Java, Python and Go programming languages and support for 4 out of the 5 package managers with no support for the Yarn package manager.

False Positives. Snyk Open Source, Red Hat CodeReady Dependency Analytics, OWASP Dependency-Check and GitHub Dependencies Graph do not state how they reduce false positive rates.OSV-Scanner reduces false positives

Table 2: Comparison of five SCA tools under different criteria.

		Snyk Open Source	GitHub Dependencies Graph and Dependabot	OSV-Scanner	Red Hat CodeReady Dependency Analytics	OWASP Dependency-Check
SDLC integration		CLI / Web app / repository integration / IDE plugin	Web app	CLI / API	IDE plugin	CLI
Vulnerability library		Snyk Intel Vulnerability Database	GitHub Advisory Database	OSV Database	Snyk Intel Vulnerability Database	National Vulnerability Database
Automatic Remediation		✓	✓	✗	✗	✗
Availability	Cost open source	Paid[1] ✗	Free ✗	Free ✓	Free ✓	Free ✓
Language Support	Languages[2] Package Managers[3]	8/10 5/5	9/10 4/5	2/10 3/5	2/10 2/5	3/10 4/5
False Positive Reduction		Not Stated	Not Stated	Call graph analysis	Not Stated	Not Stated

[1] Snyk Open Source also provides a free plan which limits the number of SCA tests each month.
[2] Languages considered - Javascript, Python, Typescript, Java, C#, C++, C, PHP, Go, and Rust.
[3] Package Managers considered - NPM, pip, Yarn, Nuget and Maven.

through the use of call graph analysis.

5.3. Discussion

The evaluation results of the SCA tools are summarized in Table 2. The results indicate that Snyk Open Source provides relatively more features and cover most of the SCA use cases, though it comes at the cost of higher pricing. For mainstream software development workflows, free tools often suffice for the need of SCA and offer unique advantages. For example, OSV-Scanner provides an intuitive and lightweight CLI option, GitHub Dependencies Graph integrates seamlessly with GitHub, and Red Hat CodeReady Dependency Analytics provides comprehensive reports.

Moreover, we note that an important factor in choosing an SCA tool is its vulnerability library. Each tool tends to leverage its own vulnerability database, which varies subtly in terms of vulnerability assessment due to the data sources. It is arguable which vulnerability database is the best. The capability of automatic remediation for the detected vulnerabilities is also worth considering.

5.4. Limitations & Threats to Validity

Among the selected tools, Snyk Open Source scans through the project that is being tested, and finds each pom.xml file that exists in the sub-projects. Other tools only detect the pom.xml file on the main project, therefore they only scan for the vulnerabilities according to this pom.xml file. Therefore, Snyk Open Source is able to find vulnerabilities for each pom.xml file. We only report the vulnerabilities detected for the main project, and those vulnerabilities are used for quantitative analysis in the paper.

Moreover, for the similarity computations among the tools, we omit the vulnerabilities that only exist in the Snyk Vulnerability DB, since other tools cannot possibly report these vulnerabilities, except for the Red Hat CodeReady Dependency Analytics tool.

6. Conclusion

In this work, we compared popular Software Composition Analysis (SCA) tools through workflow analysis and a qualitative evaluation. We proposed a set of evaluation criteria based on the features provided by the most popular SCA tools. From the workflow analysis, we highlighted key insights into the functioning of these tools, es-

pecially the distinct approaches they take toward vulnerability detection. We further distinguished these tools based on our proposed criteria.

Our analysis revealed critical distinctions between the SCA tools in terms of vulnerability detection, severity reporting, and package scope. Software developers may want to adopt more than one tool. We also found that SCA tools form clusters in the sense that tools relying on the same vulnerability database(s) tend to provide similar results. Hence, it appears unnecessary to adopt several tools belonging to the same cluster. However, despite leveraging similar databases these tools do not provide exact one to one matches in their results which suggests that the tools' distinct approaches to resolving dependencies play a major role in detecting certain vulnerabilities. These insights should provide a useful reference point for developers in selecting the most suitable SCA tool(s) based on their preferences and requirements.

A direction for future work is to quantitatively benchmark the claims made by the SCA tools regarding false positive rates. Developing a standardized framework for quantitative evaluation of false positive rates of SCA would allow for a better overall analysis of the SCA tools available.

Acknowledgements

This work was supported in part by the Boston University Red Hat Collaboratory (award numbers 2023-01-RH17 and 2024-01-RH03).

REFERENCES

1. I. Haddad, "An open guide to evaluating software composition analysis tools," Linux Foundation, November 2020.
2. P. Ombredanne, "Free and open source software license compliance: tools for software composition analysis," *Computer*, vol. 53, no. 10, pp. 105–109, 2020.
3. E. Iannone, R. Guadagni, F. Ferrucci, A. De Lucia, and F. Palomba, "The secret life of software vulnerabilities: A large-scale empirical study," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 44–63, 2022.
4. G. A. A. Prana, A. Sharma, L. K. Shar, D. Foo, A. E. Santosa, A. Sharma, and D. Lo, "Out of sight, out of mind? how vulnerable dependencies affect open-source projects," *Empirical Software Engineering*, vol. 26, pp. 1–34, 2021.
5. Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017, pp. 914–919.
6. H. Plate, S. E. Ponta, and A. Sabetta, "Impact assessment for vulnerabilities in open-source software libraries," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 411–420.
7. I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vulnerable open source dependencies: Counting those that matter," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.
8. V. Piantadosi, S. Scalabrino, and R. Oliveto, "Fixing of security vulnerabilities in open source projects: A case study of apache http server and apache tomcat," in *2019 12th IEEE Conference on software testing, validation and verification (ICST)*. IEEE, 2019, pp. 68–78.
9. G. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: automated collection of vulnerabilities and their fixes from open-source software," in *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2021, pp. 30–39.
10. A. Dann, H. Plate, B. Hermann, S. E. Ponta, and E. Bodden, "Identifying challenges for OSS vulnerability scanners - a study & test suite," *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3613–3625, 2022.
11. Z. Shi, K. Graffi, D. Starobinski, and N. Matyunin, "Threat modeling tools: A taxonomy," *IEEE Security & Privacy*, vol. 20, no. 04, pp. 29–39, 2022.
12. I. Pashchenko, S. Dashevskyi, and F. Massacci, "Delta-bench: Differential benchmark for static analysis security testing tools," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 163–168.
13. A. Milankovich and K. Tuma, "Delta security certification for software supply chains," *IEEE Security & Privacy*, vol. 21, no. 06, pp. 24–33, nov 2023.
14. I. Pashchenko, D.-L. Vu, and F. Massacci, "A qualitative study of dependency management and its security implications," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1513–1531.
15. N. Imtiaz, S. Thorn, and L. Williams, "A comparative study of vulnerability reporting by software composition analysis tools," in *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software*

Engineering and Measurement (ESEM), ser. ESEM '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3475716.3475769