INL/JOU-22-68388-Revision-0

SoK: A Framework for and Analysis of Software Bill of Materials Tools

April 2022

Arushi Arora

Changing the World's Energy Future

INL is a U.S. Department of Energy National Laboratory operated by Battelle Energy Alliance, LLC

INL/JOU-22-68388-Revision-0

SoK: A Framework for and Analysis of Software Bill of Materials Tools

Arushi Arora

April 2022

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

SoK: A Framework for and Analysis of Software Bill of Materials Tools

Arushi Arora, Virginia L. Wright, Christina Garman

ABSTRACT

Modern software development has gradually become more complex, leveraging available open-source software and third-party components. This practice has raised questions about the provenance, licensing, versioning and compliance of reused code and its dependencies. Furthermore, it is particularly important to review such code fragments and third-party components for known-vulnerabilities before they are included in a software product. A *Software Bill of Materials (SBoM)* is a mechanism to achieve such an analysis, providing transparency and visibility into a software product to both the software developer and its respective consumer. An SBoM lists information and details about all the elements constituting a piece of software and can, therefore, be used to evaluate associated security risk. While the concept of SBoMs is growing in popularity, it is still fairly new to many organizations, causing them to potentially struggle with producing and processing SBoMs and limiting their widespread adoption.

In this work, we delve into the area of SBoMs and present the state-of-the-art SBoM tools, creating a framework for analysis and categorizing them based on a diverse set of features and functionalities. We are the first to provide a detailed analysis of 83 open-source SBoM tools along with a perspective on how a potential SBoM user can select a tool based on their specific requirements. Our work aims to help promote understanding of this domain, thereby encouraging and furthering its overall adoption. We additionally seek to pave a path for future work in this area by providing recommendations to tool developers and users, researchers, and standardizing organizations.

KEYWORDS

Software Bill of Materials (SBoMs), Software supply chain security, SoK, SBoM tools

1 INTRODUCTION

Software supply chains have become increasingly complex and dynamic over the years. A large portion of the code that constitutes today's modern software has been reused and derived from miscellaneous open-source locations, for example in the form of libraries. This diverse nature of contemporary software products raises the need for an inventory of its subcomponents.

A Software Bill of Material (SBoM), which aims to provide this information, is therefore becoming a significant part of modern software. The concept of an SBoM became much more prevalent after the release of US Executive Order 14028 [30], which emphasizes safety and integrity of the software supply chain and directed the use of SBoMs to enhance US cybersecurity. This Executive Order and motivation partially came in response to the SolarWinds supply chain hack [102], where one of SolarWinds' software products (Orion) was injected with hidden malicious code that allowed hackers to gain unauthorized access to the software's ecosystem. This attack impacted about 18,000 of the company's customers, along with several contractors who were indirectly affected through third parties. Even more recently, the Apache Log4j (a logging library for Java) remote code execution vulnerability [34] allowed attackers to control the affected target system in addition to stealing the stored data, affecting both open-source and proprietary software across the software supply chain. Many organizations (and some even unaware) utilize this library as a subcomponent in their various Java-based software, again demonstrating the need for better awareness of the subcomponents of a software system.

Access to SBoMs is even more important when the corresponding software handles sensitive information or is involved in critical infrastructure. It is therefore important for the user of a piece of software to have a transparent view of the product it is using (or wishes to use). SBoMs can thus aid in verifying software components' compliance (i.e., that the components meet its user's software standards) and monitoring them for vulnerabilities (if not already present). They may also help in reviewing and backtracking the presence of an affected software subcomponent in case of a newly discovered vulnerability or a cyberattack.

In order to reap the various benefits that SBoMs can provide, it is essential to have tools that can accomplish their generation, exchange, and consumption. As the nature of contemporary software continues to become convoluted and elaborate, it is also important to automate these techniques. We believe that one of the large barriers to the widespread adoption of SBoMs is the unavailability of a unified framework and documentation that can aid software developers and their purchasers in both the usage of SBoMs as well as understanding the different features and trade-offs offered

by different tools. In other words, there still exists a wide gap between the availability and the usage of SBoM tools.

Our Contributions. In this work, we aim to provide an in-depth analysis of the state-of-the-art SBoM open-source tools, building a framework and categorizing them based on the SBoM tool taxonomy. We believe this framework will also be helpful as this relatively new space continues to evolve and new tools are developed, allowing for easier and more consistent comparison between them. In addition, based on our analysis, we provide insights and recommendations about what tools might best fit different use cases and needs. We hope to both provide a discussion of current SBoM tools in detail, but also to pave a path for future work by providing recommendations to tool developers, software users, researchers and standardizing organizations. This work is a step forwards to ease and automate SBoM production and management, hopefully leading towards its more widespread adoption and capturing the knowledge that can steer SBoM users to practice vulnerability management and software compliance.

Roadmap. Section 2 presents background and related work. Section 3 provides an in-depth discussion on state-of-the-art SBoM tools, and discusses our framework to categorize them based on various functionalities (Table 2). Section 4 provides a perspective on how to choose a tool for potential tool users. Section 5 provides general insights along with recommendations to tool developers, software users, researchers, and standardizing organizations for future work. Section 6 concludes this work.

2 SBOM BACKGROUND AND RELATED WORK

An SBoM provides a list of underlying components, along with their relationships, dependencies, and additional relevant information for a piece of software. The main objective of an SBoM is to uniquely identify a software product, and it therefore includes baseline attributes such as the SBoM's author and supplier's name, its timestamp, underlying components, and their respective relationships, versions, unique identifiers, and cryptographic hashes [56]. This information can additionally help to indentify components that are approaching (or have approached) their end-of-life along with their compliance or technologies they may support.

In this section, we provide a brief overview of existing SBoM formats [49] followed by an SBoM tool taxonomy [48], which we utilize later on in Table 2 for a functionality-based categorization of SBoM tools.

2.1 SBoM Formats

Software Package Data Exchange (SPDX) [91], an open-source project and an open standard for delivering SBoM information, provides a common format for organizations and constitutes provenance, license, security, and other associated data. The standard aims to reduce redundancy and improve compliance, security, and dependability. SPDX has been accepted as the international open standard for security, license compliance, and other software supply chain artifacts as it can represent all the elements found in traditional software development and deployment. The specification provides this information in various file formats including Resource Description Framework in Attributes (RDFa), Microsoft Excel Open XML Spreadsheet (XLSX), Extensible Markup Language (XML), YAML Ain't a Markup Language (YAML), and JavaScript Object Notation (JSON), thereby efficiently collecting and sharing data and improving accuracy. An SPDX document can be created for any software component (or set of components), an individual file, or even a snippet of code that can be exchanged as part of an SBoM. Table 4, in the Appendix, gives an overview of SPDX Version 2.2.1.

The **SPDX Lite** [87] format supports a subset of the SPDX specification and provides a software's basic information including the mandatory fields from the document creation and package information segments.

Software Identification (SWID) [52] tags provide descriptive information about a particular release of a software product in XML format, thereby allowing organizations to trace the software installed on their machines. The tag lists and provides cryptographic hashes for the elements that constitute the software and reports the author of the software as well as the tag. The SWID tag is generated during the software's installation process and deleted when the software is uninstalled. A corpus tag is created during a pre-installation stage of an installable software that identifies and defines the product. After the software product is installed on a computing device, a primary tag is created which describes the installed product. Further, a patch tag is created every time incremental changes or updates are made to a software product. Supplemental tags are used to provide any additional information about the software product and to maintain the integrity of primary and patch tags. Table 3 (in the Appendix) provides an overview of an SWID tag.

The **Concise SWID (CoSWID)** [35] tag uses concise binary object representation to reduce the size of a SWID by a notable measure and support domains such as the internet of things (IoT) which utilizes constrained devices.

CycloneDX [23] is another widely accepted SBoM standard specifically intended for use in supply chain component analysis and application security settings. It provides information related to operating systems, containers, firmware, libraries, frameworks, files, services, and optionally, hardware, describing the entire software stack. The format can be represented as XML, JSON, and Protocol Buffers. Table 5

(in the Appendix) provides an overview of CycloneDX specifications. A package URL is a way to uniquely identify and locate a software package using a URL string. This is an effort to standardize and reliably associate a software package utilizing a simplistic and expressive syntax.

Common platform enumeration (CPE) [101] is a data format that identifies applications, operating systems, and hardware devices present among an organization's computing assets, though it does not independently recognize subcomponents.

It is worth noting that SBoM formats allow scope for a software component's modification thus providing more room for customization and backporting. This could be achieved using a link in SWID, pedigree in CycloneDX, or annotation in SPDX. Additionally, it is essential to achieve interoperability of SBoMs across diverse organizations, that may opt to utilize different SBoM standards, to investigate potential vulnerabilities and associated risks and for sharing and exchanging SBoMs.

2.2 SBoM Tool Taxonomy

We now describe a taxonomy for the existing diverse set of SBoM tools as identified by NTIA [48]. This categorization is critical for any *agent* involved in dealing with (generating or managing) SBoMs or is otherwise interested in developing and designing SBoM tools. Table 1 defines these categories as (1) *produce* which defines tools that can generate SBoMs; (2) *consume* which recognizes tools that can aid in understanding, comparing and assessing SBoMs; and (3) *transform* which places tools that can combine SBoMs or convert their format type. This taxonomy and framework is used in Table 2 in Section 3 to categorize SBoM tools.

Table 1: Taxonomy of a BoM tool [48]

Category	Type	Description
Produce	Build	Automatic BoM creation as part of building a software artifact containing information about the build
	Analyze	Analysis of source or binary files generating the BoM by inspection of the artifacts & associated sources
	Edit	Assists manual entry or allows BoM data editing
Consume	Diff	Enables comparison of multiple BoMs
	View	Allows understanding of the contents in human readable form, supporting decision making
	Import	Supports discovering, retrieving, and importing a BoM into the system for further processing and analysis
Transform	Translate	Permits file type conversion while preserving the same information
	Merge	Allows combining multiple sources of BoMs and other data together for analysis and audit purposes
	Support	Support use in other tools by APIs, object models, libraries, transport, or other reference sources

2.3 Related Work

The National Telecommunications and Information Administration (NTIA) has made impressive progress in the domain by releasing introductory guidance on the use of SBoMs[47]. The working group has additionally released documentation on relevant concepts and terminologies [46], use cases [45], tool taxonomy [48] and a survey on standardizations [49]. To ascertain the exploitability of a vulnerability, a separate concept of the *Vulnerability Exploitability eXchange (VEX)* document has been presented [50]. This provides a user with information on whether a specific vulnerability (or a vulnerable component) even affects the corresponding software product. SBoMs may prove to be of great benefit, especially in the case of critical infrastructure wherein a single vulnerability can be catastrophic [9]. One piece of related work examines the use of SBoMs specifically in medical technologies and how it can benefit the healthcare supply chain [5]. We argue that this domain is still developing as well and also lacks work that fills the existing gap between the concept of SBoMs and their adoption, and that our work can be considered domain-agnostic and hence more broadly applicable. Therefore, we aim to help software developers and vendors embrace this idea and broaden the understanding of available means, which we now discuss in the following section.

3 STATE-OF-THE-ART SBOM TOOLS

This section provides a detailed categorization of existing SBoM tools which can assist software designers, developers, and vendors in understanding the tools available and deciding which one may best suit their needs. Table 2 compares numerous SBoM tools based on their features, such as interface type; repository (storage-type), platform, API support, BoM format, and NTIA formats and tooling categorization [48]. We also provide a more detailed discussion, summary, and comparison of a subset of these SBoM open-source tools below based on the taxonomy and categories presented in Section 2.2: *produce* accommodates tools that allow the creation of an SBoM as part of building a software artifact; *consume* supports tools that aid in understanding, comparing, discovering, retrieving, and importing SBoMs;

transform supports translating SBoMs from one file type to another, merging multiple SBoMs, and assisting use in other tools through APIs.

3.1 Produce

Augur [8], a part of Community Health Analytics for Open Source Software (CHAOSS), is a software suite that presents metrics on open source software development focusing on health and sustainability in an interconnected world. Augur facilitates Slack notifications and provides API support, a web interface, and an instance for demonstration. Golang programs kernel-spdx-ids [95] and npm-spdx [96] can scan a Linux kernel generating a license summary report in JSON along with a corresponding tag-value SPDX document only if SPDX-license-identifier tag is present. These tools only provide the functionality to inspect and analyze SPDX documents and support only Linux platforms. Other tools that provide a similar audit functionality include Tern [97], ScanCode toolkit [51], and Longclaw [40] which identify and analyze licenses, copyrights, package manifests direct dependencies, and third-party libraries. Another free and open-source command-line tool, Quartermaster [64], generates reports about the examined product as a component of a software build process, thereby achieving license compliance management. Another open-source framework, in-toto [36, 99], aims to defend the integrity of the software supply chain by providing authorization rights, thereby preventing any tampering that may occur during the software's transit during its development and distribution cycle.

Reuse [66] is an auditing tool delivering all features of the *produce* functionality in contrast to in-toto and Quartermaster, which only allow SBoM creation. Further, community build tools like SPDX Maven Plugin [88] and SPDX Build Tool [82] support integrations and extensions to automate the production of an SPDX document. In addition, there are several tools supporting the CycloneDX format that aims to generate SBoMs and audit known vulnerabilities in software that use a specific language (as shown in Table 2). While these tools are targeted for a specific language, BoM generation tool CycloneDX Generator [13] is more versatile as it supports Node.js, PHP, Python, Ruby, Rust, Java, .Net, and Go projects. There are also several multi-purpose CycloneDX audit tools to support a wider use case [77]. Some cross-platform SWID tag builds include Swidgen [94] which assists in the manual creation of a tag online, Labs64 [39] maven plugin, and National Institute of Standards and Technology (NIST) SWID for GNU Autotools which utilizes the GNU Autoconf and Automake to create the tag [44].

Out of these tools, ScanCode is a better choice for users preferring cross-platform support. The tool also has API support along with a web interface. On the other hand, Tern might be a better choice if the user is looking for both SPDX and CycloneDX format compatibility.

3.2 Consume

These tools promote understanding and analysis of SBoMs. SParts[79], uses an immutable ledger, Hyperledger Sawtooth in this instance, enabling users to trace open source components, including BoMs, cryptographic data, source code, legal notices, and other compliance artifacts used in a software supply chain [2]. This auditing tool is helpful to users wanting to achieve a decentralized storage architecture. MdBOM [41] is another Python tool that supports CycloneDX (JSON) and creates a markdown of the inventory from the SBoM making the BoM more human-readable. Open Web Application Security Project (OWASP) Defect Dojo [60] provides the ability to import scan reports from several security tools and export findings. The tool also provides security features like user authentication, rate limiting, and notifications regarding modifications on different channels.

3.3 Produce-Consume

Tools that fall into this category are advantageous to users who aim to generate and audit SBoMs. The Open Source Software Review Toolkit (ORT) [59] provides programmatic and command-line usage and strives to support license compliance checks, especially for free and open-source software dependencies. OWASP's Dependency-Track [61] provides component and known vulnerability analysis, license evaluation, and component identification platform. The tool provides support for applications, libraries, frameworks, operating systems, containers, firmware, files, and hardware. It also provides an API client for a secure continuous integration and delivery (CI/CD) pipeline [62]. It is worth noting that dependency-track has support for multiple repositories including PostgreSQL, MySQL, and MS SQL in contrast to ORT. SCANOSS [70] provides similar features but uses a linked-list database (LDB) [69] which is read-only and structures data in the form of linked lists. ORT, dependency-track, and SCANOSS support both CycloneDX and SPDX formats.

3.4 Consume-Transform

SW360 [80] is an open-source software project intended to work with FOSSology [31], providing both a web interface and CouchDB to collect, organize, track, and deliver information about software components, maintain license obligations, and enforce policies. The tool is cross-platform and is advantageous for auditing and merging SBoMs. SPDX Online Tool [89], an online utility tool, is beneficial for users aiming to upload, parse, verify, convert, and compare SPDX documents without worrying about downloading or installing a tool.

Table 2: A detailed comparison of open-source SBoM Tools.

Tools	Features					Produce			Consume			Transform		
	Repository	*Format*	*Interface*	*Platform*	*Others*	*Build*	*Analyze*	*Edit*	*View*	*Diff*	*Import*	*Trans*	*Merge*	*Support*
Augur [32]	★	Ⓢ	$\omega\diamond$	lm	$\alpha\Delta$	✗	✓	✗	✗	✗	✗	✗	✗	✗
FOSSology [31]	★	Ⓢ	$\omega\diamond$	lm	$\alpha\Delta$	✗	✓	✓	✓	✓	✓	✓	✓	✓
in-toto [99]	▲	Ⓢ	◇	l	α	✓	✗	✗	✗	✗	✗	✗	✗	✗
Kernel-spdx-ids [95]	▲	Ⓢ	◇	l		✗	✓	✗	✗	✗	✗	✗	✗	✗
Longclaw [40]	#	Ⓢ	ω	uC_{OS}	α	✗	✓	✗	✗	✗	✗	✗	✗	✗
npm-spdx [96]	▲	Ⓢ	◇	lmW		✗	✓	✗	✗	✗	✗	✗	✗	✗
ORT [59]	▲	ⓈC	$\diamond\omega$	lmW	$\alpha\Delta$	✓	✗	✗	✓	✓	✓	✗	✗	✗
OWASP's DT [61]	★Ʊ#	ⓈC	⧫	lmW	$\alpha\hbar\Delta$	✗	✓	✗	✓	✗	✓	✗	✗	✗
QMSTR [64]	δ	Ⓢ	◇	l	$\alpha\Delta$	✓	✗	✗	✗	✗	✗	✗	✗	✗
REUSE [66]	▲	Ⓢ	◇	l	$\alpha\Delta$	✓	✓	✓	✗	✗	✗	✗	✗	✗
SwiftBOM [7]	▲	ⓈC§	ω	lmW		✗	✗	✓	✓	✗	✗	✓	✗	✗
TERN [97]	★	ⓈC	$\diamond$⧫	l	Δ	✗	✓	✗	✓	✗	✗	✗	✗	✗
ScanCode [51]	★	Ⓢ	$\omega\diamond$	lmW	$\alpha\Delta$	✗	✓	✗	✓	✓	✗	✗	✗	✗
SCANOSS [70]	L	ⓈC	$\diamond$⧫	lmW	α	✓	✓	✗	✓	✗	✗	✗	✗	✗
SPARTS [80]	β	Ⓢ	◇	lW	$\alpha\Delta$	✗	✗	✗	✓	✓	✗	✗	✗	✗
SW360 [79]	c★	Ⓢ	ω	lmW	$\alpha\Delta$	✗	✗	✗	✓	✓	✓	✗	✓	✗
Yocto [98]	★	Ⓢ	ω	lmW		✓	✓	✗	✗	✗	✗	✗	✗	✗
SPDX OT [89]	R	Ⓢ	ω	lmW	$\alpha\Delta$	✗	✗	✗	✓	✓	✓	✓	✗	✓
SPDX Java Lib.[84]	▲	Ⓢ	◇	J		✗	✗	✗	✓	✓	✓	✓	✓	✓
SPDX Python Lib. [90]	▲	Ⓢ	◇	Py		✗	✗	✗	✓	✗	✗	✓	✗	✓
SPDX Golang Lib.[83]	▲	Ⓢ	◇	g		✗	✗	✗	✓	✗	✗	✓	✗	✓
SPDX JS Lib.[85]	▲	Ⓢ	◇	JS		✗	✗	✗	✓	✗	✗	✓	✗	✓
SPDX Maven[88]	▲	Ⓢ	P	lmW		✓	✗	✗	✗	✗	✗	✗	✗	✗
SPDX Build Tool[82]	▲	Ⓢ	P	lmW		✓	✗	✗	✗	✗	✗	✗	✗	✗
AuditJS [55]	▲	C	◇	lmW		✓	✓	✗	✗	✗	✗	✗	✗	✗
Bach [72]	▲	C	◇	lmW	Δ	✓	✓	✗	✗	✗	✗	✗	✗	✗
BOM Repository Server[10]	▲	C	ω	lmW	$\alpha\Delta$	✗	✗	✗	✗	✗	✓	✗	✗	✗
Chelsea [73]	▲	C	◇	lmW	Δ	✓	✓	✗	✗	✗	✗	✗	✗	✗
Bower [6]	▲	C	◇	lmW		✓	✗	✗	✗	✗	✗	✗	✗	✗
CocoaPods [14]	▲	C	◇	lmW		✓	✗	✗	✗	✗	✗	✗	✗	✗
CycloneDX community Build tools [16, 18, 22, 27, 28] [11, 20, 29, 67] [19, 21, 33]	▲	C	◇	lmW		✓	✗	✗	✗	✗	✗	✗	✗	✗
CycloneDX Generator [13]	▲	C	◇	lmW	Δ	✓	✗	✗	✗	✗	✗	✗	✗	✗
Jake [74]	▲	C	◇	lmW	Δ	✓	✗	✗	✗	✗	✗	✗	✗	✗

Retire.js [65]	▲	C	◇*P*	*lmW*		✗	✓	✗	✗	✗	✗	✗	✗	✗
CycloneDX Core for Java [15]	▲	C	*P*	*J*	*α*	✗	✗	✗	✗	✗	✗	✗	✓	✓
CycloneDX Maven [17]	▲	C	*P*	*lmW*		✗	✗	✗	✓	✓	✗	✓	✓	✗
CycloneDX Web Tool [24]	▲	C	*ω*	*lmW*		✗	✗	✗	✓	✓	✗	✓	✓	✗
CycloneDX CLI [12]	▲	C	◇	*lmW*	Δ	✗	✗	✗	✓	✓	✗	✓	✓	✗
CycloneDX community Transform tools [25, 26]	▲	C	Lib	*lmW*		✗	✗	✗	✗	✗	✗	✗	✗	✓
dtrack-audit [62]	▲	C	◇	*lmW*		✗	✓	✗	✗	✗	✗	✗	✗	✗
DevAudit [77]	#★	C	◇	*lmW*	Δ	✗	✓	✗	✗	✗	✗	✗	✗	✗
Go Sonatypes [78]	▲	C	Lib	*lmW*		✗	✗	✗	✗	✗	✗	✗	✗	✓
Grype [3]	▲	C	◇	*lmW*	Δ	✓	✓	✗	✗	✗	✗	✗	✗	✗
ittosai [37]	▲	C	◇	*lmW*		✗	✓	✗	✗	✗	✗	✗	✗	✗
MdBOM [41]	▲	C	◇	*lmW*		✗	✗	✗	✓	✗	✗	✗	✗	✗
OpenRewrite [57]	▲	C	*P*	*lmW*	*α*	✓	✓	✗	✗	✗	✗	✗	✗	✗
Defect Dojo [60]	★#	C	♦	*lmW*	*α*Δ	✗	✗	✗	✗	✗	✓	✗	✗	✗
ShiftLeft Scan [71]	▲	C	◇	*lmW*	Δ	✓	✓	✗	✓	✗	✗	✗	✗	✗
Syft [4]	▲	ⓈC	◇	*lmW*	Δ	✓	✓	✗	✗	✗	✗	✗	✗	✗
Nancy [75]	▲	C	◇	*lmW*	Δ	✓	✗	✗	✗	✗	✗	✗	✗	✗
VS Code Plugin for Nexus IQ [76]	▲	C	*P*	*lmW*		✗	✗	✗	✗	✗	✗	✗	✗	✓
Swidgen [94]	▲	§	*ω*	*lmW*		✓	✗	✗	✗	✗	✗	✗	✗	✗
StrongSwan [92]	▲	§	◇	*l*		✓	✗	✗	✗	✗	✗	✗	✗	✗
Labs64 SWID Maven Plugin [38]	▲	§	*P*	*lmW*		✓	✗	✗	✗	✗	✗	✗	✗	✗
RPM 2 SWID Tag [93]	▲	§	◇	*l*		✓	✓	✗	✓	✗	✗	✗	✗	✗
NIST SWID for GNU Autotools [44]	▲	§	◇	*lmW*		✓	✗	✗	✗	✗	✗	✗	✗	✗
NIST SWID Builder [42]	▲	§	◇	*J*	*α*	✓	✗	✗	✗	✗	✗	✗	✗	✗
NIST SWID Maven Plugin [43]	▲	§	*P*	*lmW*		✓	✗	✗	✗	✗	✗	✗	✗	✗
NIST SWID Tag Validator [54]	▲	§	◇	*lmW*	*α*	✗	✗	✗	✓	✗	✗	✗	✗	✗
NIST SWID Repo Client [53]	▲	§	◇	*J*	*α*	✗	✗	✗	✓	✗	✗	✗	✗	✗
libswid [58]	▲	§	Lib	*C++*		✗	✗	✗	✗	✗	✗	✗	✗	✓

Repository: ★ PostgreSQL | ▲ File system | *δ* Dgraph | *L* LDB | *β* Blockchain | *c* CouchDB | # MySQL | Ʊ MS SQL | *R* Redis

Format: Ⓢ SPDX | C CycloneDX | § SWID

Interface: ♦ GUI | *P* Plugin |◇ CLI | *ω* Web service | Lib Library

Platform: *l* Linux Distribution | *m* MacOS | *W* Windows | C_{os} CentOS | *u* Ubuntu | *J* Java | *Py* Python | *g* Golang | *JS* JavaScript | C++

Others: *ħ* Hardware support | *α* API support | Δ Docker support

CycloneDX Web Tool [24] is a browser-based tool that supports the conversion of SBoM versions and formats, their validation, and merging into a single BoM. Verification and conversion of SBoM versions, as provided by SPDX Online Tool and CycloneDX Web Tool are significant in promoting interoperability between different tools. In addition, there are also language-specific libraries endowing SPDX format, for developers that intend to build software that facilitates consuming and transforming SBoMs [83–85, 90].

3.5 Produce-Consume-Transform

FOSSology toolkit [31], one of the most versatile open-source tools available, is a license compliance software that was released in 2007 under the Linux Foundation that lets a user generate an SPDX file and copyrights notices. This facilitates the analysis, storage, and distribution of open-source software and its metadata. Yocto Project [98], which provides tools for embedded developers worldwide to share technologies, software stacks, configurations, and best practices using FOSSology API to generate SPDX documentation during a software build. Similarly, Augur-SPDX [8] uses FOSSology for its license scanning. CERT's SwiftBOM tool [7] has an edge over FOSSology, as the former generates SBoMs in SPDX, CycloneDX, and SWID formats, but it provides limited import capability.

3.6 Sharing & Exchanging SBoMs

A last set of tools facilitates the sharing and exchange of SBoMs. The CycloneDX BoM Repository Server [10] is a service for publishing, managing, distributing SBoMs, and promoting basic authentication and authorization features. The tool converts all BoMs to Protocol Buffer format before storage which might lead to loss of information. Another such tool, Digital BoM (DBoM) [100], aims to streamline sharing attestations through a set of relational, non-relational, and blockchain-based repositories. Unfortunately, the tool lacks security and privacy properties such as encryption at rest and in transit, authentication, and authorization.

4 CHOOSING AN SBOM TOOL

Table 2 provides an extensive view of the state-of-the-art tools and categorizes them based on various properties which can help a prospective tool user to make a choice. For instance, one of the most diverse SBoM tools includes FOSSology [31] which supports all three functionalities. However, for a potential tool user that aims to only generate SBoMs (*produce*), a simpler tool such as REUSE [66] would be meaningful. It is worth noting that this tool only supports generation in SPDX format, therefore a more sophisticated tool such as SCANOSS [70] would be helpful if the user desires SBoM generation in both SPDX and CycloneDX formats. This would enable the user (say a software developer) to share SBoMs based on the format needed by the vendor.

Other criteria that a tool user (say a software developer or consumer) may consider before choosing a tool includes platform (i.e., the operating system and language) support, which should be compatible with the chosen tool. Additionally, SBoM tools support a diverse set of storage or database types such as CouchDB, MySQL, and Redis which can provide more flexibility to a potential tool user. A user can also make a decision based on the interface type they prefer, which can either be a GUI (Graphical User Interface), a plugin, CLI (Command Line Interface), or a Web service. On the other hand, a tool user, say a vendor, who only wishes to read and understand an SBoM may only opt for *consume* functionality tools. If the vendor desires more features such as file-type conversion and auditing, then they may opt for both *consume-transform* functionalities as provided by CycloneDX Web Tool and CLI [12, 24].

Case study. To demonstrate how our framework and taxonomy can help an SBoM find the right tool for their use case, we now walk through a brief example.

Alice, a software developer, hopes to choose a tool to generate SBoMs, as mandated by Bob, who wishes to purchase her software product. It is worth noting that Alice reuses open-source code (just like many other software developers) and therefore requires a tool that can not only generate SBoMs but can also process SBoMs belonging to the reused code. Such a tool will help her to understand the code she intends to reuse. For instance, the tool may aid her in checking the code for any known vulnerability, versions, and if the code meets Bob's software compliance needs. She, therefore, needs a tool that supports both *produce* and *consume* functionality for her to generate SBoMs and provide her with an understanding of them. Additionally, Alice requires that the tool should be able to support both SPDX and CycloneDX standards since she might have to process existing SBoMs from multiple vendors for the code she reuses. She further checks her storage and platform compatibility and finds herself comfortable using CLI-based tools since she is a developer. In such a scenario, a tool like ORT [59] seems a fit for her needs since it supports all her requirements and is cross-platform. Alice can easily determine this using our framework by simply breaking down her needs in each category as described and then easily identifying which tools meet her requirements.

In addition, if Alice needs a tool that also supports the *transform* functionality, since she is outsourcing SBoMs from multiple sources for the reused code and may, therefore, have to perform file type conversion or combine two or more SBoMs. Cyclonedx CLI cy-cli may meet Alice's requirements

in such a situation. It is worth noting that, as per our knowledge, there still does not exist a tool that is broad enough to support all the functionalities, hence, users may have to opt for multiple tools as per their requirements.

On the other hand, Bob, who only wishes to understand SBoMs, may choose a tool that supports the *consume* functionality. For example, if Bob instructed *Alice* to provide her SBoM in SPDX format and desired a GUI, he may opt for OWASP's Dependency-Track [61] which is also cross-platform and supports Postgres, MySQL and MS SQL SBoM storage type. SPDX Online Tool [89] may also be a great option that supports Redis for SBoM storage and is cross-platform and web-based.

5 FUTURE WORK AND DIRECTIONS

We believe this work is one of the first to provide a detailed analysis of state-of-the-art SBoM tools, and we hope to help broaden the understanding of available means for software developers and vendors. Despite the wide range of tools that currently exist, this domain still demands attention from a research and development outlook as it is still in an early phase.

Recommendations to Researchers and Tool Developers. One of the barriers that may hamper widespread adoption of SBoMs is interoperability across organizations. This area still requires an elegant tool that can aid parties opting for different SBoM formats and standards to perform conversions without loss of information. Additionally, there still does not exist a sophisticated tool that is diverse enough, thereby providing all the functionalities in one package (*produce*, *consume* and *transform*). Furthermore, the existing tools do not take into account that SBoMs may contain confidential and sensitive information. This issue may restrict a vendor to share an SBoM for a software product that, say, uses proprietary code. Therefore, there is a need for sophisticated tools that implemented sharing and exchanging of SBoMs keeping the confidentiality of subcomponents and the privacy of their respective owners into consideration.

Additionally, there is a requirement for tools that can attest to or verify the authenticity of SBoMs as well as automatically identify a software subcomponent (through a unique ID [56]). It is worth noting that the domain still struggles with a lack of a universally accepted source for SBoM component identification. Also, there is still not a global access point or a platform available for attested or accepted SBoMs (although SPDX provides a list of accepted licenses [86]). A publish/subscribe system that informs software consumers about reported vulnerabilities, updates, or any modifications would ascertain to be valuable.

Recommendations to Standardization Bodies. There are a number of SBoM formats available such as SPDX, CycloneDX, SWID, SPDX Lite, CoSWID and CPE [23, 35, 52, 87, 91, 101]. We believe it is therefore important for standardization bodies to provide advice and guidance on a specific SBoM format (or set of formats with interoperable APIs) which can aid in streamlining the process of producing and consuming SBoMs. Since the hardware and software sectors have gradually converged, it would be helpful if the SBoM formats additionally include a way to uniquely identify the hardware utilized in the concerned product. This can be achieved by specifying the hardware's barcode formats, MAC (Media Access Control) address, a timestamp, stock-keeping unit (SKU) number, Global Individual Asset Identifier and Global Model Number and so on [1].

Recommendations to Tool Users. We recommend that tool users first identify the functionalities (*produce*, *consume, transform*) they need in an SBoM tool, along with other desired features such as user interface, storage type, SBoM format and platform support as specified in Table 2. The case study provided in Section 4 is an instance of how users should proceed when choosing a tool. Users should also keep in mind that most of these tools do not support data encryption, therefore, they should handle SBoMs containing sensitive data appropriately.

6 CONCLUSION

Modern software is acquiring a more involved and intricate architecture, thereby making its components less perceptible. This poses a concern when it comes to assessing a software's security from underlying potential vulnerabilities and its (and subcomponents') compliance. SBoMs provide a way to combat with this problem, allowing software users to gain more transparency into the product. While an SBoM is not a silver bullet that can prevent all cyberattacks, it can help strengthen an organization from known-vulnerabilities and may later aid in identifying affected (or vulnerable) software components. It is worth noting that their widespread adoption is only achievable if software developers and users have access to tools that can aid in generation and management SBoMs, thereby automating the process, propelling them to make better security decisions. In this work, we presented and categorized the state-of-the-art SBoM tools, helping different agents involved in dealing with SBoMs identify tools that would best suit their need. We believe this work is an important step that can contribute towards wide adoption, growth and further expansion of SBoMs, by delivering a coordinated knowledge about how to produce and manage them efficiently.

REFERENCES

[1] Haris Aftab, Komal Gilani, JiEun Lee, Lewis Nkenyereye, SeungMyeong Jeong, and JaeSeung Song. 2020. Analysis of identifiers in IoT platforms. *Digital Communications and Networks* 6, 3 (2020), 333–340.

[2] Benjamin Ampel, Mark Patton, and Hsinchun Chen. 2019. Performance modeling of hyperledger sawtooth blockchain. In *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 59–61.

[3] Anchore. n.d. Grype. https://github.com/anchore/grype.

[4] Anchore. n.d. Syft. https://github.com/anchore/syft.

[5] Seth Carmody, Andrea Coravos, Ginny Fahs, Audra Hatch, Janine Medina, Beau Woods, and Joshua Corman. 2021. Building resilient medical technology supply chains with a software bill of materials. *NPJ Digital Medicine* 4, 1 (2021), 1–6.

[6] CDX Bower. n.d. CycloneDX Bower managed dependencies generator. https://github.com/hanstdam/cdx-bower-bom.

[7] CERT. n.d. SwiftBOM Demo. https://democert.org/sbom/.

[8] CHAOSS. n.d. Augur License. https://github.com/chaoss/augur-license.

[9] Cybersecurity and Infrastructure Security Agency (CISA). n.d. Healthcare and Public Health Sector. https://www.cisa.gov/stopransomware/healthcare-and-public-health-sector.

[10] CycloneDX. n.d. BOM Repository Server. https://github.com/CycloneDX/cyclonedx-bom-repo-server.

[11] CycloneDX. n.d. CycloneDX-Buildroot. https://github.com/alvinchchen/cyclonedx-buildroot.

[12] CycloneDX. n.d. CycloneDX CLI. https://github.com/CycloneDX/cyclonedx-cli.

[13] CycloneDX. n.d. CycloneDX CocoaPods. https://github.com/AppThreat/cdxgen.

[14] CycloneDX. n.d. CycloneDX CocoaPods. https://github.com/CycloneDX/cyclonedx-cocoapods.

[15] CycloneDX. n.d. CycloneDX Core for Java. https://github.com/CycloneDX/cyclonedx-core-java.

[16] CycloneDX. n.d. CycloneDX for Go. https://github.com/ozonru/cyclonedx-go.

[17] CycloneDX. n.d. CycloneDX for Maven. https://github.com/CycloneDX/cyclonedx-maven-plugin.

[18] CycloneDX. n.d. CycloneDX for Node.js. https://github.com/CycloneDX/gh-node-module-generatebom.

[19] CycloneDX. n.d. CycloneDX for NPM. https://github.com/CycloneDX/cyclonedx-node-module.

[20] CycloneDX. n.d. CycloneDX for PHP Composer. https://packagist.org/packages/cyclonedx/cyclonedx-php-composer.

[21] CycloneDX. n.d. CycloneDX for Rust. https://crates.io/crates/cyclonedx-bom.

[22] CycloneDX. n.d. CycloneDX module for .NET. https://www.nuget.org/packages/CycloneDX/.

[23] CycloneDX. n.d. CycloneDX Specifications. https://cyclonedx.org/specification/overview/.

[24] CycloneDX. n.d. CycloneDX Web Tool. https://cyclonedx.github.io/cyclonedx-web-tool.

[25] CycloneDX. n.d. Go Library for CycloneDX to Encode and Decode BoMs. https://github.com/CycloneDX/cyclonedx-node-module.

[26] CycloneDX. n.d. Rust Library for CycloneDX to Encode and Decode BoMs. https://github.com/CycloneDX/cyclonedx-node-module.

[27] CycloneDX. n.d. SBT BOM. https://github.com/siculo/sbt-bom.

[28] CycloneDX for Erlang. n.d. Elixir Mix. https://github.com/voltone/sbom.

[29] CycloneDX for Python. n.d. CycloneDX Python SBOM Generation Tool. https://pypi.org/project/cyclonedx-bom/.

[30] Executive Office of the President. 2021. Improving the Nation's Cybersecurity. https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity.

[31] Robert Gobeille. 2008. The fossology project. In *Proceedings of the 2008 international working conference on Mining software repositories*. 47–50.

[32] Sean P Goggins, Matt Germonprez, and Kevin Lumbard. 2021. Making Open Source Project Health Transparent. *Computer* 54, 08 (2021), 104–111.

[33] Gradle. n.d. CycloneDX for Gradle. https://plugins.gradle.org/plugin/org.cyclonedx.bom.

[34] Raphael Hiesgen, Marcin Nawrocki, Thomas C Schmidt, and Matthias Wählisch. 2022. The Race to the Vulnerable: Measuring the Log4j Shell Incident. *arXiv preprint arXiv:2205.02544* (2022).

[35] IETF. 2021. Concise Software Identification Tags. https://www.ietf.org/id/draft-ietf-sacm-coswid-19.html.

[36] in-toto. n.d. in-toto Command Line Tools. https://in-toto.readthedocs.io/en/latest/command-line-tools/index.html.

[37] ittosai. n.d. ittosai. https://github.com/devops-kung-fu/ittosai.

[38] Labs64. n.d. SoftWare IDentification (SWID) Tags Generator (Java Library). https://github.com/Labs64/swid-generator.

[39] Labs64. n.d. SWID Generator. https://github.com/Labs64/swid-generator.

[40] LLNL. n.d. Longclaw. http://rosecompiler.org/.

[41] MdBOM. n.d. Markdown SBOM. https://github.com/HaRo87/mdbom.

[42] National Institute of Standards and Technology (NIST).. n.d. NIST SWID Builder. https://pages.nist.gov/swid-tools/swidval/.

[43] National Institute of Standards and Technology (NIST). n.d. NIST SWID for Maven Plugin. https://pages.nist.gov/swid-tools/swid-maven-plugin/.

[44] National Institute of Standards and Technology (NIST). n.d. SWID Autotools. https://github.com/usnistgov/swid-autotools#swid-for-gnu-autotools.

[45] National Telecommunications and Information Administration (NTIA). 2019. Roles and Benefits for SBOM Across the Supply Chain. https://www.ntia.gov/files/ntia/publications/ntia_sbom_use_cases_roles_benefits-nov2019.pdf.

[46] National Telecommunications and Information Administration (NTIA). 2021. Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM). https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf.

[47] National Telecommunications and Information Administration (NTIA). 2021. NTIA SBoM Summary. https://www.ntia.gov/files/ntia/publications/sbom_overview_20200818.pdf.

[48] National Telecommunications and Information Administration (NTIA). 2021. SBoM Tool Classification Taxonomy. https://www.ntia.gov/files/ntia/publications/\ntia_sbom_tooling_taxonomy-2021mar30.pdf.

[49] National Telecommunications and Information Administration (NTIA). 2021. Survey of Existing SBOM Formats and Standards. https://www.ntia.gov/files/ntia/publications/ntia_sbom_formats_and_standards_whitepaper_-_version_20191025.pdf.

[50] National Telecommunications and Information Administration (NTIA). 2021. Vulnerability-Exploitability eXchange (VEX) – An Overview. https://www.ntia.gov/files/ntia/publications/vex_one-page_summary.pdf.

[51] nexB. n.d. ScanCode Toolkit. https://github.com/nexB/scancode-toolkit.

[52] NIST. n.d. Guidelines for the Creation of Interoperable Software Identification (SWID) Tags. https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf.
[53] NIST. n.d. NIST SWID Tag Repo Client. https://pages.nist.gov/swid-tools/swid-repo-client/.
[54] NIST. n.d. NIST SWID Tag Validator. https://pages.nist.gov/swid-tools/swidval/.
[55] NPMJS. n.d. AuditJS. https://www.npmjs.com/package/auditjs.
[56] NTIA. 2021. Software Identification Challenges and Guidance. https://www.ntia.gov/files/ntia/publications/ntia_sbom_software_identity-2021mar30.pdf.
[57] OpenRewrite. n.d. Rewrite. https://github.com/openrewrite/rewrite.
[58] OpenSCAP. n.d. libswid. https://github.com/OpenSCAP/libswid.
[59] OSS Rewiew Toolkit. n.d. ORT. https://github.com/oss-review-toolkit.
[60] OWASP. n.d. Defect Dojo. https://github.com/DefectDojo/django-DefectDojo.
[61] OWASP. n.d. OWASP's Dependency-Track. https://docs.dependencytrack.org.
[62] Ozonru. n.d. dtrack-audit. https://github.com/ozonru/dtrack-audit.
[63] Package URL. n.d. Package URL Specifications. https://github.com/package-url/purl-spec.
[64] Quartermaster. n.d. QMSTR. https://qmstr.org.
[65] Retire.js. n.d. Retire.js. https://retirejs.github.io/retire.js/.
[66] REUSE. n.d. REUSE Specifications. https://reuse.software/spec/.
[67] Ruby Gems. n.d. CycloneDX for Ruby Gems. https://rubygems.org/gems/cyclonedx-ruby.
[68] Luis Alberto Benthin Sanguino and Rafael Uetz. 2017. Software vulnerability analysis using CPE and CVE. *arXiv preprint arXiv:1705.05347* (2017).
[69] SCANOSS. n.d. ScanCode LDB. https://github.com/scanoss/ldb.
[70] SCANOSS. n.d. SCANOSS. https://scanoss.com.
[71] Shiftleft. n.d. Shiftleft Scan. https://www.shiftleft.io/scan/.
[72] Sonatype Nexus Community. n.d. Bach. https://github.com/sonatype-nexus-community/bach.
[73] Sonatype Nexus Community. n.d. Chelsea. https://github.com/sonatype-nexus-community/chelsea.
[74] Sonatype Nexus Community. n.d. Jake. https://github.com/sonatype-nexus-community/jake.
[75] Sonatype Nexus Community. n.d. Nancy. https://github.com/sonatype-nexus-community/nancy.
[76] Sonatype Nexus Community. n.d. Sonatype Nexus IQ Plugin for VS Code. https://github.com/sonatype-nexus-community/vscode-iq-plugin.
[77] Sonatypes Nexus Community. n.d. DevAudit. https://github.com/sonatype-nexus-community/DevAudit.
[78] Sonatypes Nexus Community. n.d. Go Sonatypes. https://github.com/sonatype-nexus-community/go-sona-types.
[79] SPARTS. n.d. SW360. https://sparts.readthedocs.io/en/latest/web/intro.html.
[80] SPARTS. n.d. SW360. https://www.eclipse.org/sw360/.
[81] SPDX. n.d. Composition of an SPDX Document. https://spdx.github.io/spdx-spec/composition-of-an-SPDX-document/.
[82] SPDX. n.d. SPDX Build Tool. https://github.com/spdx/spdx-build-tool.
[83] SPDX n.d. SPDX Golang Library. https://github.com/spdx/tools-golang.
[84] SPDX. n.d. SPDX Java Libraries & Tools. https://github.com/spdx/tools-java.
[85] SPDX. n.d. SPDX JavaScript Library. https://github.com/spdx/spdx-tools-js.
[86] SPDX. n.d. SPDX-Licenses. https://spdx.org/licenses/.
[87] SPDX. n.d. SPDX LITE Specification. https://spdx.github.io/spdx-spec/SPDX-Lite/.
[88] SPDX. n.d. SPDX Maven Plugin. https://github.com/spdx/spdx-maven-plugin.
[89] SPDX. n.d. SPDX Online Tool. https://tools.spdx.org/app/.
[90] SPDX. n.d. SPDX Python Library. https://github.com/spdx/tools-python.
[91] SPDX Dev. n.d. SPDX Specification. https://spdx.dev/specifications/.
[92] StrongSwan. n.d. SWID Generator. https://github.com/strongswan/swidGenerator.
[93] SWID Tags. n.d. RPM 2 SWID Tag. https://github.com/swidtags/rpm2swidtag.
[94] Swidgen. n.d. Swidgen. https://pgodowski.github.io.
[95] Swinslow. n.d. Kernel SPDX IDs. https://github.com/swinslow/kernel-spdx-ids.
[96] Swinslow. n.d. NPM SPDX. https://github.com/swinslow/npm-spdx.
[97] Tern Tools. n.d. Tern. https://github.com/tern-tools/tern.
[98] The Yocto Project. n.d. Yocto. https://www.yoctoproject.org/.
[99] Santiago Torres-Arias. 2020. *In-toto: Practical Software Supply Chain Security.* Ph. D. Dissertation. New York University Tandon School of Engineering.
[100] Unisys. n.d. Digital Bill of Materials. https://dbom-project.readthedocs.io/en/latest/what-dbom.html.
[101] David Waltermire and Brant Cheikes. 2015. *Forming common platform enumeration (CPE) names from software identification (SWID) tags.* Technical Report. National Institute of Standards and Technology.
[102] Evan D Wolff, KM Growley, MG Gruden, et al. 2021. Navigating the solarwinds supply chain attack. *The Procurement Lawyer* 56, 2 (2021).

7 APPENDIX

This section provides an insight into various SBoM format (SWID, SPDX and CycloneDX) specifications.

Table 3: SWID specification [52]

Field	Description	Sub-fields
<SoftwareIdentity>	It is the root of the tag providing a detailed description of a software product	@name, @version, @versionScheme, @tagId, @tagVersion, @supplemental, @patch, @corpus
<Entity>	This sub-element optionally identifies the creator, licensor(s), or distributor(s) of the tag	@name, @regid, @role, @thumbprint
<Evidence>	This sub-element is used in situations when a software product has a missing tag and a third party discovers and reuses the respective untagged software product. <Evidence> may therefore be used to store results from the scan after the discovery tool generates a *Primary Tag* for the discovered untagged product	@date, @deviceId, <Directory>, <File>, <Process>, <Resource>
<Link>	This sub-element is used to connect a *Patch Tag* or a *Supplemental Tag* to a *Primary Tag*. Additionally, it may be used to associate any *source tag* to other arbitrary information elements	@href, @rel
<Meta>	This sub-element may be optionally used to provide additional metadata attributes	@activationStatus, @colloquialVersion, @edition, @product, @revision
<Payload>	This optional sub-element aims to provide details regarding additional elements, for instance, files, folders, license keys that may be installed on a device during the installation of a software product	<Directory>, <File>, <Process> <Resource>

Table 4: SPDX specification v2.2.1 [81]

Field	Cardinality	Description	Sub-fields
Document Creation Information	Mandatory, one	It renders essential knowledge for forward and backward compatibility for processing tools	SPDX Version, Data License, SPDX Identifier, Document Name, SPDX Document, Namespace, External Document References, License List Version, Creator, Created, Creator Comment, Document Comment
Package Information	Optional, one or many	In case SPDX information is being used to describe packages, then one instance of the Package Information exists per package, providing important information about it	Package Name, Package SPDX Identifier, Package Version, Package File Name, Package Supplier, Package Originator, Package Download Location, Files Analyzed, Package Verification Code, Package Checksum, Package Home Page, Source Information, Concluded License, All Licenses Information from Files, Declared License, Comments on License, Copyright Text, Package Summary Description, Package Detailed Description, Package Comment, External Reference, External Reference Comment, Package Attribution Text

File Information	Optional, one or many	For each file in the software package, an instance of the File Information is required which provides knowledge about a given file including licenses and copyright	File Name, File SPDX Identifier, File Type, File Checksum, Concluded License, License Information in File, Comments on License, Copyright Text, File Comment, File Notice, File Contributor, File Attribution Text.
Snippet Information	Optional, one or many	When a file reuses content from another primary source, snippets can be used to provide added information, by indicating the component of a file that was originally created under another license.	Snippet SPDX Identifier, Snippet from File SPDX Identifier, Snippet Byte Range, Snippet Line Range, Snippet Concluded License, License Information in Snippet, Snippet Comments on License, Snippet Copyright Text, Snippet Comment, Snippet Name, Snippet Attribution Text
Other Licensing Information	Optional, one or many	This section lists any identified, declared, or concluded licenses that do not appear on the SPDX License List	License Identifier, Extracted Text, License Name, License Cross Reference, License Comment
Relationships	Optional, one or many	This section renders information about the relationship between two SPDX elements which could be files, packages, or SPDX Documents	Relationship, Relationship Comment
Annotations	Optional, one or many	This field provides information regarding comments on a file, package, or the entire document to validate and clarify obscure SPDX elements	Annotator, Annotation Date, Annotation Type, SPDX Identifier Reference, Annotation Comment

Table 5: CycloneDX specification [23]

Field	Description
BOM Metadata	This field incorporates the essential information related to the BoM including the corresponding software component along with its supplier, manufacturer, and license information and the tool used to create the BOM
Components	Components provide the complete inventory list of the first and third party components and can be represented as coordinates (group, name, version), Package URL [63], Common Platform Enumeration (CPE) [68], SWID [52], or cryptographic hash functions
Services	This field provides information about the external APIs that the software may request along with the flow of data
Dependencies	This section describes the components and their dependencies, both direct and transitive relationships, on other components, which can be expressed in the dependency graph
Compositions	This field describes the constituent parts of the software, and its entirety can be termed as complete, incomplete, incomplete first-party only, incomplete third-party only, or unknown
Extensions	This field enables agile prototyping of new abilities for future specialized and industry-specific use cases, promoting community support and development