INL/JOU-22-66300-Revision-1

Strengthening the Security of Operational Technology: Understanding Contemporary Bill of Materials

July 2022

Arushi Arora, Virginia L Wright, Christina Garman

Changing the World's Energy Future

Idaho National Laboratory

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

INL/JOU-22-66300-Revision-1

Strengthening the Security of Operational Technology: Understanding Contemporary Bill of Materials

Arushi Arora, Virginia L Wright, Christina Garman

July 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

Strengthening the Security of Operational Technology: Understanding Contemporary Bill of Materials

Abstract

The evolution of cyber-physical infrastructure has made its security more challenging. The last few years have witnessed a convergence of hardware and software segments in various domains, including operational technology (OT) which is responsible for carrying out critical tasks such as monitoring and controlling power grids, nuclear plants, transportation, and emergency services. Both hardware and software encapsulate numerous open source and proprietary subcomponents, making it crucial for end-users to understand the composition of the products they are using. For example, wind turbines incorporate thousands of lines of code (software) used for the turbine's design, planning, operation, and analytics in addition to the numerous hardware subcomponents that construct it. Due to the highly complex nature of software and hardware, knowledge of the components and subcomponents is required to mitigate cyber vulnerabilities and defend against cyberattacks.

There has also been a transformation from a traditional linear supply chain into a global, dynamic, diverse, and interconnected system. The digitization of the supply chain makes it easier to find and exploit vulnerabilities. Critical infrastructures (e.g., power grids, oil, natural gas, water, and wastewater) rely on OT to function, and if the OT is compromised, equipment damage and potential interruption of services could result. A significant security measure to protect OT systems from disruption is to develop a supply chain bill of materials (BoM) corresponding to the software and hardware used in OT, along with attestations amongst vendors and asset owners. A supply chain BoM is a proactive way to understand the inherent vulnerabilities in the system and mitigate them in advance of being exploited. BoMs bolster the

trust placed in the digital infrastructure and enhance software supply chain security by sustaining the management of component obsolescence and compliance, along with the seclusion of unsafe segments of a specific product.

Adopting BoM tools is becoming increasingly important across various government sectors, as evidenced by the recent U.S. executive order on cybersecurity (NIST 2021). This paper aims to classify BoMs based on structure, functionality, component type, and architecture. The work also discusses case studies to further highlight the benefits of BoMs. In addition, it identifies missing pieces in existing BoM implementations so that future research may identify bounds on where it could expect to make improvements and directly enable researchers to identify promising areas for exploration. Further, the authors provide valuable recommendations to tool developers, researchers, and standardizing organizations (policymakers), additionally benefitting critical infrastructure owners and government executives. This aids in paving a path for future work, thereby, providing suggestions to determine a tool for consumers that best suit their needs.

Keywords

Software Bill of Materials (SBoM), SBoM tools, supply chain, operational technology, critical infrastructure security

Introduction

The 20th century brought the Digital Revolution, which led to the adoption and proliferation of digital computers and digital record-keeping. The past few years have also witnessed the introduction of online shopping, connected vehicles, smart homes, and technologies that have completely disrupted the conventional monolithic supply chain. This has also inflated consumers' demands and expectations, for instance in the form of fast delivery. The escalating

infiltration of digital technologies such as artificial intelligence (AI), blockchain, and automation has also generated vast opportunities for organizations and supply chain practices.

Traditional supply chains, which were meant to be linear, have transformed into a dynamic, diverse and interconnected system. Most major organizations are now scattered so far and wide that they have supply chains that exist in multiple countries. These vast supply chain operations can range from anything like customer service obligations to critical infrastructure: therefore, the circumstances disrupting them can prove catastrophic.

A bill of materials (BoM), which defines the components that are needed to manufacture a product, can help prevent bottlenecking the supply chains, ensure faster time to market, guarantee greater operational efficiency and may lessen the risk of errors and rework. BoMs can better represent the supply chain and associated documents and payloads contained at each stage. The contemporary digitalized BoMs seek to improve supply chain efficiency by facilitating sharing of BoM-relevant information, including data licenses, software descriptions, attestations and versions, hardware components, and lists of staff or other human resources, among supply chain partners.

BoMs support the management of component obsolescence and compliance and the isolation of unsafe segments of a specific product, strengthening the trust placed in the digital infrastructure and enhancing software supply chain security. It also provides an autonomous way to apply government regulations with contractors to protect against the use of counterfeit devices and compromised software in critical infrastructure. To operate correctly, critical infrastructure further relies on operational technology (OT) solutions, which aim to regulate industrial equipment, building management systems, fire control systems, and physical access control mechanisms. Implementing a mechanism to produce, consume, and transport BoMs is especially important to safely monitor OT components that are used in multiple areas like nuclear power

plants, wind farms, and power grids, (Parnas, Asmis, and Madey 1991; Slootweg et al. 2003; Shiroudi et al. 2012; Singhal and Saxena 2012).

OT components are usually not replaced at the same rate as consumer technology infrastructure, and therefore lack modern cyberattack preventive measures. This raises concern about cybersecurity vulnerabilities existing in components or devices supporting critical infrastructure, as cyberattacks can cause equipment damage, interruption of services, and even loss of life. Itis crucial to have a mechanism to share BoMs, corresponding to the software and hardware used in respective components, along with attestations amongst vendors and asset owners. BoMs can ensure all parties involved in the digital supply chain network—including suppliers, manufacturers, and customers—are aware of changes concerning their product of interest through automated synchronization.

In this paper, the authors classify BoMs based on structure, functionality, component type, and architecture. They further provide a deep insight into BoMs that will help readers identify tools that best suit their requirements based on the tool's utility. This paper also specifies broad use cases of employing BoMs and highlights the benefits. In addition, the authors identify missing pieces in existing BoM implementations so that future research may identify bounds on possible improvements and directly enable researchers to identify promising areas for future exploration. The authors also argue that this domain still lacks research thereby paving a path forward for future work.

Overview

Adopting SBoM tools is becoming increasingly important across various government sectors, as evidenced by the recent U.S. executive order on cybersecurity, EO 14028 (NIST 2021). In 2018, the National Telecommunication and Information Administration (NTIA) gathered a cross-sector, industry-led, multi-stakeholder process on software component transparency, to

understand the potential, obligations, and obstacles in the domain (NTIA 2021d). Since its initiation, experts have made valuable progress in defining an SBoM and listing its benefits, identifying existing standards and formats that can be employed to convey information (NTIA 2021e; NTIA 2021g). This section provides a brief overview of the BoM, highlighting its motivation, the process involved, and its use cases, along with an example.

BoM

A BoM is a list of ingredients that make up a particular product. An SBoM identifies and lists software components, whereas a hardware BoM (HBoM) identifies and lists hardware components of a product. With the convergence of software and hardware in modern devices and instruments, their respective BoMs must identify and list both software and hardware components that constitute the device. The amount and type of data included in each BoM may differ and may depend on factors such as the BoM's usage, its industry or sector, and the needs of BoM consumers. NTIA (NTIA 2021a) gives a brief overview of actors in an SBoM system and defines a set of baseline SBoM components as listed in Table 1 and 2 respectively.

Motivation. Modern software often reuses and imports open-source components, such as libraries, from a previous stage, combining it with novel contributions to build a different product. The present digital supply chain is an intricate web of dependencies and any modification in the chain can result in wide-ranging consequences. It is hard to evaluate if the software product meets consumers' standards, obeys licensing regulations, or is free of known vulnerabilities. For example, in addition to hardware components, wind turbines incorporate software that may include third-party or proprietary software as subcomponents, which may be responsible for the turbine's design, planning, operation, and analytics (shown in figure 1). With such a complex and diverse nature of contemporary supply chains, it is hard to mitigate a disruption that occurs somewhere in the chain. Another challenge is to upgrade downstream

component(s), in case of a patch in its corresponding upstream element.

[Figure 1 Near Here]

Process. The first step to creating an SBoM during the software build and packaging process is to record the components that the supplier creates themselves along with its related information and all the reused imported components. The SBoM provides additional information about the software's subcomponents' technical variations, versions, and underlying inherited vulnerabilities. An SBoM is created whenever a component undergoes updates, upgrades, releases, and patches. Next, the SBoM must be made available to the consumer along with the software component. This may assist the consumer in performing an impact and compliance assessment along with the software's validation. The retrieval of the SBoM should be simple and can be carried out by providing an additional file, or a dynamic URL which may be beneficial for constrained devices. This facilitates a persistent assessment of the software and tracking of any updates.

Use Cases. From the perspective of a supplier, an SBoM can help monitor the software's components for vulnerabilities, their end-of-life (EOL), and understand their complex dependencies, thereby providing better transparency into the code resulting in prompt delivery of code updates or patches (NTIA 2021e). It can also support the following: aid in reducing code bloat by providing complete knowledge about relevant and available open source components; familiarize the supplier with the license obligations of the components that are used in a software product; and, aid suppliers in reviewing code and identifying blacklisted components.

From the perspective of a consumer, an SBoM can support verification of the software components' sources, check for compliance with the consumer's policies, and scan for known vulnerabilities by performing software component analysis. It can also help the consumer to

identify requirements for the software product as well as its integration. Further, an SBoM can provide a consumer with additional data, for instance, the software components' EOL, and attest claims that the supplier makes. An SBoM may also help in stimulating independent mitigations and make versed risk-based decisions while operating the software. It also keeps information more organized and readily available, which may aid in effective planning and reducing overall costs through a more contoured and efficient execution.

[Figure 2 Near Here]

Example. Figure 2 presents a conceptual SBoM dependency tree along with a table stating all SBoM elements (as described in Table 2) that belong to Alice's application. As per the SBoM information, this application has four components: Alice's application (*primary*), which reuses Bob's user interface template, Carol's database application, and David's cryptographic libraries. The upstream component (Bob's user interface template) has unknown dependencies. In other words, Bob's code may be the root or if it may have further upstream components. Alice's website also uses Carol's database application which further employs David's cryptographic libraries. In addition, it is known that David's library is the root and does not utilize any upstream components.

[Table 1 Near Here]

[Table 2 Near Here]

Terminology

Various organizations have launched both open source and proprietary BoM tools. Previous works in this domain do not thoroughly investigate and describe existing BoM terminologies, their classifications and formats, and desired properties in the tools. In this section, the authors attempt to elaborate on this aspect of the contemporary BoM.

Classification

In recent years, hardware and software have converged and are dependent on one another such that the two are no longer mutually exclusive. In addition to classifying BoMs based on their structure and functionality (Zhou et al. 2018; Liu, Lai, and Shen 2014), the authors introduce a classification category based on their component type. BoMs can contain sensitive information which may not be public and require restricted access, so the authors also introduce another classification category to accommodate this criterion based on a BoM's availability.

Based on Structure. This type of BoM is suitable when the product does not constitute subcomponents. A single-level BoM is utilized for non-complex products with just one level of subassemblies. On the other hand, a multi-level BoM is used in more complicated assemblies of a final product and is, therefore, composed of multiple levels of subcomponents.

Based on Functionality. Functionality BoMs are broadly classified into eight categories.

1. Configurable BoMs contain a list of all the components required to devise a configurable product, to meet customers' particular requirements.
2. Manufacturing BoMs provide a roadmap for manufacturing any physical product, presenting a structured list of all the components and subcomponents, including their relationships with one another.
3. Engineering BoMs provide the necessary elements and directions for making a particular product from a design standpoint and further helps in composing the manufacturing BoM.
4. Sales BoMs list the final finished product as a sales item and its respective subcomponents as inventory.
5. Assembly BoMs are similar to the sales BoM as they contain the finished product mentioned in the sales document. However, this document does not list the

subcomponents of the final product.

6. Service BoMs carry a list of all the components, installation steps, and restoration directions. This BoM is used by service technicians while installing or servicing a product.
7. Production BoMs list subcomponents, costs, specifications, and quantities of a finished product, serving as the groundwork for a production order. The components listed can also be converted into finished items during the production process.
8. Template BoMs allow flexibility to update quantities or replace components of the product and can be used for either production or sales BoMs.

Based on Component Type. SBoM is effectively a nested inventory, and a list of components that make up the software. Similarly, an HBoM provides a nested list of all the elements that make up the hardware.

Based on Availability. A public BoM architecture is completely open and can be accessed by anyone. A private BoM architecture is accessible to only selected members of a group or an organization, thereby serving a private business. A permissioned BoM architecture, which lies between the above-listed categories, allows a verified entity to access the BoM, thereby accommodating different organizations sharing a supply chain.

Storage Mechanism

Finding a suitable database solution for any application is not straightforward, and this challenge holds true for BoMs. Application developers choose between relational, non-relational, and blockchain-based databases based on the type of data handled by the application. Other factors to consider include operation and maintenance costs, service stability, performance, scalability, security, and ease of developing database interface and modifying the database schema (Jatana et al. 2012; Boicea, Radulescu, and Agapin 2012; Aboutorabi et al. 2015; Li and

Manoharan 2013).

Properties

Table 3 gives an overview of the SBoM tool taxonomy, which is significant to the range of use cases in SBoM generation and consumption. Tool consumers may benefit from identifying and understanding availability as per their needs. The information in this table has been derived from categories defined by NTIA (NTIA 2021c) in addition to the transport category, which represents the tools that support a mechanism to share and exchange SBoMs (NTIA 2021b). For instance, an organization that develops and vends software might be interested in SBoM tools that fall under the produce category, whereas the consumers of this software might be interested in SBoM tools that fall under the consume category. Further, the authors provide a list of security and privacy properties in Table 4 that may be desired by the users to support SBoMs containing sensitive information. This may be beneficial for tool developers and consumers while developing and choosing a tool for use.

[Table 3 Near Here]

[Table 4 Near Here]

State-Of-The-Art

Many organizations rely on open-source software for their product development. A conventional software product nowadays often reuses third-party libraries and packages as subcomponents, which are not obvious to detect. SBoM tools can aid in understanding dependencies within more comprehensive and complex software, comply with the licenses, review software's components for vulnerabilities, blacklist components if required, and provide SBoM to consumers. To accomplish this goal, SBoM standards, such as Software Package Data Exchange (SPDX) (Gandhi, Germonprez, and Link 2018), Software Identification (SWID) tags

(Waltermire and Cheikes 2015; SWID Tags, n.d.) and CycloneDX (CycloneDX, n.d.) have been introduced (NTIA 2021c).

Why Use SBoMs?

SBoMs can prove to be beneficial in multiple industries including automotive, software, agricultural, healthcare, and energy sectors. They can decrease unplanned, unscheduled work, in addition to reducing code bloat. Below are some of the benefits that SBoMs may provide.

Vulnerability Management. An SBoM may be beneficial to determine if a subcomponent contains any vulnerabilities and needs a patch. This also helps identify any downstream component that may be at risk due to this vulnerability. Further, an SBoM may provide details of previous patches and updates. Introducing and automating vulnerability assessment and management can curtail time to remediation and promote awareness (Dwyer 2018; Greenberg 2018).

License Management. SBoMs can guarantee that the software and its respective subcomponents comply with the licensing requirements.

Intellectual Property. SBoM information can assist intellectual property applications, as access to such data provides details of subcomponents used, licenses, and history of updates. It is worth noting that both SPDX and SWID can provide license information.

High Assurance. SBoMs can ensure the integrity of subcomponents in the software by providing the source, information about its suppliers, history of modifications, and the chain of custody as components move through the supply chain. It also stimulates the exclusion of low-quality or abandoned software elements. Further, the SBoM also includes information about the technologies supported by the corresponding software and EOL dates which may assist in carrying out timely updates to the software.

To help readers understand how this research can benefit in averting severe cyberattacks that affect critical infrastructure, the authors shed light on the recent attack on SolarWinds and Log4j (CISA 2021) vulnerability along with a discussion on their prevention (Wolf, Growley, and Gruden 2021).

Case Study: The Attack on SolarWinds

The Attack. SolarWinds is an American company that provides software for monitoring network and information technology (IT) infrastructure, serving hundreds of thousands of organizations around the globe. In September 2019, one of SolarWinds' software products, Orion, was infiltrated with malicious code, known as SUNBURST, by hackers who discreetly broke into the company's systems. This malware, which remained hidden when dormant, could create backdoors when active, permitting a malicious user to gain unauthorized access to the software ecosystem. This further let the hackers gain access to the software's host machine and other components in the network. The infected Orion software was available as a legitimate SolarWinds software update which went undetected, impacting about 18,000 SolarWinds' customers, both government and commercial. SolarWinds' supply chain attack has also affected contractors that do not use the company's software. For instance, CrowdStrike and Malwarebytes, which are not SolarWinds' customers, were indirectly affected through third parties.

Prevention. While the usage of SBoMs was not widespread and their benefits were not widely understood at the time (as Executive Order 14028 (NIST 2021) was partly in response to such cyberattacks (CSIS 2022)), the authors argue that such damaging supply chain cyberattacks could potentially be thwarted in the future by enforcing certain preventive measures, including the implementation of SBoMs. If a vendor, like SolarWinds, had provided an SBoM along with the software (or its update) to its customers, the latter could have performed various inspections

that might have prevented this attack. An SBoM could assist customers in verifying the software's integrity. For instance, the hash of the software component in the SBoM can verify that the respective component has not been tampered with. The SBoM can also list hashes computed from multiple algorithms. Therefore, the customer could have further verified the software's authenticity. SBoMs can provide the vendor's digital signature, enabling the customers with non-repudiation capabilities. In addition to allowing the consumer to understand the components and third-party subcomponents of the software, SBoMs can also verify license compliance and check known vulnerabilities existing in the respective software.

Case Study: The Log4j Vulnerability

The Attack. This case study discusses the remote code execution vulnerability in Apache Log4j (a logging library for Java) disclosed on December 10, 2021 (CISA 2021). This library is a subcomponent used by many companies in a variety of Java-based software and web portals. The vulnerability allowed an attacker to execute arbitrary code on devices or execute a DoS (Denial of Service) attack on the targeted servers, remotely. This allows a malicious entity to further steal the data or take control over the affected target system.

Prevention. This case study is a perfect instance to showcase how modern software is composed of multiple third-party and open-source elements making it complex (Bauer et al. 2020). The widespread use of Java across both IT and OT platforms made this vulnerability complicated to patch as many open-source, as well as proprietary applications, also utilized the Log4j library. In addition, Log4j is also used as a JAR file in a third-party application and is also embedded in many custom applications. Therefore, remediating such an attack is complicated along with being resource-intensive.

It is, therefore, feasible to implement measures in order to safeguard against such vulnerabilities beforehand. The authors argue that keeping a record of all components enables

organizations to analyze the impact, identify, and mitigate risks allowing greater transparency. The authors claim that mandating an SBoM along with the software purchase could have helped many organizations deal with such a vulnerability promptly, aiding them to perform a targeted search by scanning just the SBoM. Having an SBoM along with the software would have also made the companies more aware of the subcomponents being reused in the respective software.

Building the Right SBoM

This paper's systemization and categorization of SBoM tools, and their features could have helped SolarWinds (the vendor) build an SBoM to achieve the above. First, observe a few key properties (Produce, Consume, or Transform) and features (SBoM format, repository support, user interface, and platform support) the vendor might want, based on the discussion in the previous section. The vendor could have chosen a produce software, which at least sustains the build feature supporting automatic BoM creation as part of building a software artifact. Additionally, the vendor could combine the build property with consume properties (if desired) which can support understanding, comparing, discovering, retrieving, and importing SBoMs. Transform properties could have been combined which could support translating SBoMs from one file type to another, merging multiple SBoMs, and assisting use in other tools through APIs. Further, the vendor could have considered other suitable features like format (CycloneDX), user interface (CLI), cross-platform, and API support. Verification and analysis tasks on the consumer side could have been achieved by choosing SBoM tools that support the Analyze (Produce) property, which generates the BoM by inspection of the artifacts and associated sources, and the View (Consume) property, which allows understanding of the contents in human-readable form, endowing decision-making. In case the consumer preferred a different SBoM format (say SPDX), they may have also chosen the tool to support the Translate property, to perform SBoM format conversion. Based on the different roles an organization performs, they

should choose a tool that best suits their needs. For instance, the vendor could choose CycloneDX Generator (AppThreat, n.d.) to generate CycloneDX SBoMs and SCANOSS (SCANOSS, n.d.) or ORT (ORT, n.d.) if additional Analyze or Consume properties are desired. Similarly, the consumer of the software could have chosen OWASP's Dependency-Track (OWASP DT, n.d.), which provides component and known vulnerability analysis, license evaluation, and component identification platform.

Future Directions

Executive Order 14028 (NIST 2021) requests enhanced cybersecurity through maintaining privacy, security, and integrity of the software supply chain. NTIA is still in the process of formalizing records, and documentation relevant to SBoMs. In 2022 the National Institute of Standards and Technology (NIST) will be releasing recommendations to enhance software supply chain security along with issuing policies and standards (NIST 2021).

Barriers. Unfortunately, current SBoM usage in this sector is low due to the limited understanding of available tools and lack of standardization. Another barrier to establishing broader SBoM usage is the lack of a common platform to distribute and exchange SBoMs. Moreover, these SBoMs may contain sensitive or proprietary information, strongly necessitating the need of a privacy-preserving SBoM distribution and access platform. Also, for OT devices, there might exist a number of unknown subcomponents of software since the codebase may be old and the owner themself might be unaware of certain libraries or dependencies in the software product. Further, it is important for SBoM tools to handle dynamic and indirect dependencies. Since software nowadays is complex and cross-organizational, an SBoM tool should therefore be scalable before it is put into practice. Clearly, this domain is in a nascent stage requiring attention both from a research and development perspective.

Below the authors aim to pave a path forward, discussing some of the current issues that hinder the wide usage and adoption of SBoMs, highlighting the present needs, and providing recommendations to different actors associated with SBoMs.

Recommendations to Tool Developers, Researchers, and Authors. The authors identify the following open challenges concerning SBoM tools.

- ***Privacy and Security***. Current implementations of tools that enable sharing of SBoMs introduce problems encompassing sharing crucial and sensitive (sometimes confidential) information in a secure and privacy-preserving approach (Unisys, n.d.). One obstacle that a vendor may encounter is with sharing security-relevant information about a proprietary subcomponent produced by a third-party and contained in the vendor's software. Sharing software vulnerabilities in the digital supply chain may provide a malicious entity access to information before the vulnerability is patched. One of the challenges in this domain would be to check for the presence of software components (and subcomponents) or vulnerabilities anonymously (i.e., without revealing the verifier's identity to the BoM owner). A privacy-preserving BoM query would prevent information leaks about components the verifier intends to use. Another possible challenge would be how a party could trust an SBoM or HBoM entry for the BoM's correctness and completeness (BoM attestation).
- ***Transform Tools***. CycloneDX CLI (CLI, n.d.) can convert from CycloneDX 1.2 to SPDX 2.1 and 2.2. For SPDX-based transformation, there exists some language-specific libraries (SPDX Golang, n.d.; SPDX BT, n.d.; SPDX Python n.d.), but the area still lacks a sophisticated tool that provides a wider range of format conversions without loss of information. Such tools are beneficial in instances where participating organizations adopt different SBoM formats.

- ***Recursive Verification***. Another possible area of research could be recursively verifying an SBoM. It is worth noting that modern software usually reuses third-party open-source code as subcomponents, thereby making SBoMs multi-level and complex. Attesting SBoMs, which may list a chain of software subcomponents, entails backtracing them in the complex supply chain. Normalizing this information still needs to be further studied and understood.
- ***Interoperability***. Different organizations, vendors, and asset owners sharing a supply chain may use different tools to generate or consume SBoMs. In such a situation, it is important to automatically identify a software subcomponent among multiple tools, systems, and databases. This can be achieved by using a consistent and unique primary key. A universal method to generate a complete identifier for a software component without the need for any extra information or central authority is necessary. CPE, one of the existing approaches in this domain, is a means of defining and distinguishing software including applications and operating systems, as well as hardware devices (Sanguino and Uetz 2017). The disadvantage of using CPE is that the naming convention is not unique, therefore composing a central dictionary is impractical (Fitzgerald and Foley 2013). Some other efforts in the area include the use of package URL and software heritage ID, which have still not been universally adopted (NTIA 2021f; PURL, n.d.). Another recommendation for organizations like Internet Engineering Task Force (IETF), and Internet Society (ISOC), is to standardize SBoM generation by identifying essential fields across different formats.

Recommendations to Standardizing Organizations. Closing the gap between defining standards and their respective implementations can be a challenging task. In this subsection, the authors discuss future directions for organizations dealing with standardizing SBoMs.

- ***SBoM Formats***. Potential improvements in the existing SBoM formats would be the inclusion of the status of unknown subcomponents that may exist in a particular software product or if no such components exist. Organizations may also identify essential fields to standardize SBoM generation across different formats.
- ***Dependent Links and Subcomponents***. One of the main applications of SBoM tools is software composition analysis (SCA) which aims to identify open source software in a product thereby assessing its security, license compliance, and code quality and providing vulnerability patches. The current SPDX format does not specify the information on any known patched vulnerabilities or updates.
- ***HBoM Integration***. In addition, the authors propose that hardware-specific fields, based on the device type, can be added to the already existing SBoM formats, such as CycloneDX. These potential fields could specify barcode formats (Universal Product Code or European Article Number), media access control (MAC) address, a timestamp, stock-keeping unit (SKU) number, global trade item number (GS1 GTIN, n.d.), global location number (GS1 GLN, n.d.), global individual asset identifier (GS1 GIAI, n.d.), global model number (GS1 GMN, n.d.) and so on. Additionally, based on the type of device, specialized identification field(s) can be supplemented. For instance, a unique device identification (UDI) system can be added to the HBoM to uniquely identify medical devices (Food and Drug Administration 2013).
- ***SBoM Distribution and Access***. The goal is to have a BoM be accessible and available to the right entities at the right time. SBoMs of open-source components should be easily available for a consumer or a developer reusing the component to view through a global access point. Although SPDX provides a list of accepted licenses (SPDX-Licenses, n.d.), a common domain for SBoMs is still not available. DBoM (Unisys, n.d.) aims to improve

attestation and BoM sharing across different entities sharing a supply chain but still requires a way to map different databases supporting relational or non-relational data formats. An SBoM may be distributed to the consumer through a URL using a manufacturer usage description (MUD) extension (NTIA, 2021b; IETF 2019). Another way to achieve this task is to provide a manifest offering licensing specifications and package contents to the downstream consumer of the software. In addition, a publish or subscribe system would prove to be advantageous, allowing consumers to get notified about updates from the supplier through a shared channel. In other words, a consumer can access the SBoM by getting it directly from the supplier, say through an email. In case the SBoM exists on the device executing the software to which it belongs, it can be fetched using protocols like HTTPS. An SBoM can also be made available on a shared repository.

Recommendations to Users: Choosing a Tool. Based on various properties of an SBoM tool, such as repository and format supported, user interface and most importantly its functionalities (i.e., produce, consume, and transform) a choice can be made. For instance, FOSSology (Gobeille 2008) and ORT (ORT, n.d.) are two of the most diverse open source SBoM tools available. Below the authors provide additional insight which may help users to select a tool as per their requirements.

- ***Insight on SBoM Format***. It is worth noting that SWID tags currently support only the XML format which is perceived to have poor performance characteristics concerning its parsing (Nicola and John 2003). JSON files, on the other hand, are easier to consume. Using the lightweight CoSWID may therefore be beneficial in certain scenarios like usage in constrained devices. The authors propose the use of open standard- CycloneDX which can provide vulnerability details through the fields `cpe`, `purl` and `swid`. In

addition, this lightweight format was specifically designed to improve application security and facilitate supply chain component analysis and therefore incorporates characteristics of SWID and SPDX formats. The format also has an active community and support.

- ***Insight on Tool Repositories***. Software used across the globe handles sensitive information and carries out critical tasks which raise the importance of having clarity into the software's composition. The concept of building and exchanging BoMs of these contemporary devices can promise transparency into its components, defending their networks from known vulnerabilities, therefore, this information must be available to those who require it. It is worth noting that the BoM data is inherently relational as shown in Figure 2. This is due to the diverse nature of the contemporary supply chain architecture as well as code reuse, which is an integral part of modern software. In general, a relational database would be most appropriate for a BoM application as this would preserve the logical connections (i.e., relations between the data) (Goggins, Germonprez, and Lumbard 2021; Gobeille 2008; OWASP DT, n.d.; Tern, n.d.). A blockchain-based architecture that uses a relational database would also serve the purpose in case properties, such as decentralization and immutability, provided by blockchain technology are desired (OWASP DT, n.d.). A choice between various relational database tools can be made based on other desired properties like security, performance features, and so on (SParts, n.d.). The authors believe that PostgreSQL (Obe and Hsu 2017) would be an appropriate choice for an SBoM tool and would be beneficial in situations when an existing repository or vendor-native database is non-relational. As per the authors' knowledge, the only shortcoming of PostgreSQL is that it is a centralized database, which implies that the server storing the DBoM database is a single point of failure. To tackle

such a situation, PostgreSQL allows for replica servers, maintaining database availability in case of a failover situation (PostgreSQL, n.d.). Blockchain technology has earned popularity over the past few years, which cannot be denied. But for the purpose of choosing a BoM repository, the authors argue that a blockchain-based database does not hold many benefits. The authors, therefore, propose to use a much simpler database like PostgreSQL.

Conclusion

In this paper, the authors work to thoroughly describe BoM terminologies, classifications, and desired properties in SBoM tools. They further specify broad use cases of employing BoMs and discuss case studies, highlighting its benefits. In addition, the paper successfully investigates the advancement in the area of SBoMs, thereby helping the readers identify tools that best suit their requirements. The authors identify missing pieces in existing BoM implementations and provide insightful recommendations to tool developers, users, researchers, and standardizing organizations, thereby, leading the pathway for future research.

Acronyms and Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BoM	Bill of Materials
CPE	Common Platform Enumeration
DBoM	Digital Bill of Materials
DoS	Denial of Service
EOL	End-of-life
HBoM	Hardware Bills of Materials
IETF	Internet Engineering Task Force

ISOC	Internet Society
IT	Information Technology
JSON	JavaScript Object Notation
MAC	Media Access Control
MUD	Manufacturer Usage Description
NIST	National Institute of Standards and Technology
NTIA	National Telecommunication and Information Administration
OT	Operational Technology
OWASP	Open Web Application Security Project
SBoM	Software Bill of Materials
SCA	Software Composition Analysis
SKU	Stock-Keeping Unit
SPDX	Software Package Data Exchange
SWID	Software Identification
UDI	Unique Device Identification
XML	Extensible Markup Language

Author Capsule Bios

References

Aboutorabi, Seyyed Hamid, Mehdi Rezapour, Milad Moradi, and Nasser Ghadiri. 2015. "Performance evaluation of SQL and MongoDB databases for big e-commerce data." *In 2015 International Symposium on Computer Science and Software Engineering (CSSE), 1–7*. IEEE. https://doi.org/10.1109/CSICSSE.2015.7369245.

AppThreat. n.d. "CycloneDX Generator." Accessed Jan 18, 2022. https://github.com/AppThreat/cdxgen.

Bauer, Andreas, Nikolay Harutyunyan, Dirk Riehle, and Georg-Daniel Schwarz. 2020. "Challenges of tracking and documenting open source dependencies in products: A case study." *Open Source Systems* 582, 25-35. https://dx.doi.org/10.1007%2F978-3-030-47240-5_3.

Boicea, Alexandru, Florin Radulescu, and Laura Ioana Agapin. 2012. "MongoDB vs Oracle–database comparison." *In 2012 third international conference on emerging intelligent data and web technologies*, 330–335. IEEE. https://doi.org/10.1109/EIDWT.2012.32.

CISA. 2021. "Mitigating Log4Shell and Other Log4j-Related Vulnerabilities". Accessed March 31, 2022. https://www.cisa.gov/uscert/ncas/alerts/aa21-356a.

CLI. n.d. "CycloneDX CLI." Accessed Jan 18, 2022. https://github.com/CycloneDX/cyclonedx-cli.

CSIS. 2022. "Executive Order 14028 and Federal Acquisition Regulation (FAR): 10 Months Later". Accessed June 29, 2022. https://www.csis.org/blogs/strategic-technologies-blog/executive-order-14028-and-federal-acquisition-regulation-far-10.

CycloneDX. n.d. "CycloneDX Specifications." Accessed Jan 18, 2022. https://cyclonedx.org/specification/overview/.

Dwyer, AC. 2018. "The NHS cyber-attack: A look at the complex environmental conditions of WannaCry." *RAD Magazine 44*.

Fitzgerald, William M, and Simon N Foley. 2013. "Avoiding inconsistencies in the security content automation protocol." *In 2013 IEEE Conference on Communications and Network Security (CNS), 454–461*. IEEE. https://doi.org/10.1109/CNS.2013.6682760.

Food and Drug Administration, HHS. 2013. "Unique device identification system. Final rule." *Federal register* 78, no. 185 (2013): 58785-58828. https://www.federalregister.gov/d/2013-23059.

Gandhi, Robin, Matt Germonprez, and Georg JP Link. 2018. "Open data standards for open source software risk management routines: an examination of SPDX." *In Proceedings of the 2018 ACM Conference on Supporting Groupwork*, 219–229. https://doi.org/10.1145/3148330.3148333.

Gobeille, Robert. 2008. "The fossology project." In Proceedings of the 2008 international working conference on Mining software repositories, 47–50. https://doi.org/10.1145/1370750.1370763.

Goggins, Sean P, Matt Germonprez, and Kevin Lumbard. 2021. "Making Open Source Project Health Transparent." *Computer* 54 (08): 104–111. https://doi.ieeecomputersociety.org/10.1109/MC.2021.3084015.

Greenberg, Andy. 2018. "The untold story of NotPetya, the most devastating cyberattack in history." *Wired, August 22*. https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/.

GS1 GIAI. n.d. "Global Individual Asset Identifier (GIAI)." Accessed Jan 19, 2022. https://www.gs1.org/standards/id-keys/global-individual-asset-identifier-giai.

GS1 GLN. n.d. "Global Location Number (GLN)." Accessed Jan 19, 2022. https://www.gs1us.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=158&language=en-US&PortalId=0&TabId=134.

GS1 GMN. n.d. "Global Model Number (GMN)." Accessed Jan 19, 2022. https://www.gs1au.org/what-we-do/standards/global-model-number-gmn.

GS1 GTIN. n.d. “Global Trade Item Number (GTIN).” Accessed Jan 19, 2022. https://www.gs1.org/standards/id-keys/gtin.

IETF. 2019. “Manufacturer Usage Description Specification.” https://datatracker.ietf.org/doc/html/rfc8520.

Jatana, Nishtha, Sahil Puri, Mehak Ahuja, Ishita Kathuria, and Dishant Gosain. 2012. “A survey and comparison of relational and non-relational database.” *International Journal of Engineering Research & Technology* 1 (6): 1–5.

Li, Yishan, and Sathiamoorthy Manoharan. 2013. “A performance comparison of SQL and NoSQL databases.” *In 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 15–19. IEEE. https://doi.org/10.1109/PACRIM.2013.6625441.

Liu, Min, Jianbo Lai, and Weiming Shen. 2014. “A method for transformation of engineering bill of materials to maintenance bill of materials.” *Robotics and Computer-Integrated Manufacturing* 30 (2): 142–149. https://doi.org/10.1016/j.rcim.2013.09.008.

Nicola, Matthias, and Jasmi John. 2003. “Xml parsing: a threat to database performance.” *In Proceedings of the twelfth international conference on Information and knowledge management*, 175–178. http://dx.doi.org/10.1145/956863.956898.

NIST. 2021. “Improving the Nation’s Cybersecurity: NIST’s Responsibilities under the May 2021 Executive Order.” Accessed Jan 18, 2022. https://www.nist.gov/itl/executive-order-improving-nations-cybersecurity.

NTIA. 2021a. “Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM).” https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf.

NTIA. 2021b. “Sharing and Exchanging SBOMs.” https://www.ntia.gov/files/ntia/publications/ntia_sbom_sharing_exchanging_sboms-10feb2021.pdf.

NTIA. 2021c. “Taxonomy, SBoM Tool Classification.” https://www.ntia.gov/files/ntia/publications/ntia_sbom_tooling_taxonomy-2021mar30.pdf.

NTIA. 2021d. “Software Bill of Materials.” https://www.ntia.gov/SBOM.

NTIA. 2021e. “Roles and Benefits for SBOM Across the Supply Chain.” https://www.ntia.gov/files/ntia/publications/ntia_sbom_use_cases_roles_benefits-nov2019.pdf.

NTIA. 2021f. “Software Identification, and Guidance Challenges.” https://www.ntia.gov/files/ntia/ publications/ntia_sbom_software_identity-2021mar30.pdf.

NTIA. 2021g. “The Minimum Elements For a Software Bill of Materials (SBoM).” https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf.

Obe, Regina O, and Leo S Hsu. 2017. “PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database”. *O’Reilly Media, Inc*.

ORT. n.d. “OSS Rewiew Toolkit”. Accessed Jan 18, 2022. https://github.com/oss-review-toolkit.

OWASP DT. n.d. “OWASP’s Dependency-Track”. Accessed Jan 18, 2022. https://docs.dependencytrack.org.

Parnas, David Lorge, GJK Asmis, and Jan Madey. 1991. “Assessment of safety-critical software in nuclear power plants.” *Nuclear safety 32 (2): 189–198*.

PostgreSQL. n.d. “PostgreSQL Replication & Automatic Failover Tutorial”. Accessed Jan 18, 2022. https://www.enterprisedb.com/postgres-tutorials/postgresql-replication-and-automatic-failover-tutorial#replication-use.

PURL. n.d. “Package URL Specifications.” Accessed Jan 18, 2022. https://github.com/package-url/purl-spec.

Sanguino, Luis Alberto Benthin, and Rafael Uetz. 2017. “Software vulnerability analysis using CPE and CVE.” *arXiv preprint arXiv:*1705.05347.

SCANOSS. n.d. “SCANOSS.” Accessed Jan 18, 2022. https://scanoss.com.

Shiroudi, A, R Rashidi, GB Gharehpetian, SA Mousavifar, and A Akbari Foroud. 2012. “Case study: Simulation and optimization of photovoltaic-wind-battery hybrid energy system in Taleghan-Iran using homer software.” *Journal of Renewable and Sustainable Energy* 4 (5): 053111. http://dx.doi.org/10.1063/1.4754440.

Singhal, Anjali, and RP Saxena. 2012. “Software models for smart grid.” *In 2012 First International Workshop on Software Engineering Challenges for the Smart Grid (SE-SmartGrids),* 42–45. IEEE. https://doi.org/10.1109/SE4SG.2012.6225717.

Slootweg, JG, SWH De Haan, H Polinder, and WL Kling. 2003. “General model for representing variable speed wind turbines in power system dynamics simulations.” *IEEE Transactions on power systems* 18 (1): 144–151. https://doi.org/10.1109/TPWRS.2002.807113.

SParts. n.d. “Sparts Documentation.” Accessed Jan 18, 2022. https://sparts.readthedocs.io/en/latest/web/intro.html.

SPDX BT. n.d. “SPDX Build Tool.” Accessed Jan 18, 2022. https://github.com/spdx/spdx-build-tool.

SPDX Golang. n.d. “SPDX Golang Library.” Accessed Jan 18, 2022. https://github.com/spdx/tools-golang.

SPDX Python. n.d. “SPDX Python Library.” n.d. Accessed Jan 18, 2022. https://github.com/spdx/tools-python.

SWID Tags. n.d. “Guidelines for the Creation of Interoperable Software Identification (SWID) Tags.” Accessed Jan 18, 2022. https://nvlpubs.nist.gov/nistpub.

Unisys. n.d. “Digital Bill of Materials.” Accessed Jan 18, 2022. https://dbom-project.readthedocs.io/en/latest/what-dbom.html.

Waltermire, David, and Brant Cheikes. 2015. “Forming common platform enumeration (CPE) names from software identification (SWID) tags.” *Technical report. National Institute of Standards and Technology.* https://csrc.nist.gov/publications/detail/nistir/8085/draft.

Wolff, Evan D., K. M. Growley, and M. G. Gruden. 2021. “Navigating the solarwinds supply chain attack.” *The Procurement Lawyer* 56, no. 2. https://www.crowell.com/files/20210325-Navigating-the-SolarWinds-Supply-Chain-Attack%20.pdf.

Zhou, Chunliu, Xiaobing Liu, Fanghong Xue, Hongguang Bo, and Kai Li. 2018. “Research on static service BOM transformation for complex products.” *Advanced Engineering Informatics* 36:146–162. https://doi.org/10.1016/j.aei.2018.02.008.

Strengthening the Security of Operational Technology: Understanding Contemporary Bill of Materials

Tables and Figures

Table 1. SBoM Actors (NTIA 2021d).

Actors	Description
Element	Ingredient of an SBoM system
Supplier	The entity of the SBoM that creates, defines, and identifies ingredients of a software product and generates associated SBoMs
Component	The unit of a software product. A component may be a library, a file, or larger software like an operating system, a database system, or an office suite that is further comprised of components. It is defined by a supplier when the component is created, packaged, or distributed.
Consumer	An entity that receives SBoMs
Operator	An operator is a leaf entity in the SBoM conceptual tree that is a consumer but not a supplier. (Note that a consumer who reuses components can also be a supplier.)
Author	An author makes claims about components created or incorporated by a separate entity i.e., a supplier
Attribute	Properties and knowledge about a component
SBoM Entry	A row in the SBoM table specifying a component and relevant attribute

Table 2. List of baseline SBoM Elements (NTIA 2021d).

SBoM Element	Description
Author Name	Author of the SBoM entry
Supplier Name	Name or identity of supplier of the component in the SBoM entry
Component Name	One or more component name(s). May include the capability in case of multiple names or aliases
Version String	Version information which aids in identifying a component

Component Hash	Cryptographic hash of the component to uniquely and precisely identify a binary
Unique Identifier	A unique identifier of the component
Relationship	Relationship is inherent in the design of the SBoM. This could be *includes* (downstream) or *included in* (upstream). A *downstream* component is the one towards the consumer whereas an *upstream* component is towards the supplier.
Relationship Assertions	Refers to a component representing its immediate upstream relationships. Four categories cover these assertions— ▪ *Unknown*: This implies that immediate upstream (towards the supplier) components are not currently known and therefore not yet recorded. ▪ *Root*: This indicates that there are no immediate upstream relationships. ▪ *Partial*: There is at least one immediate upstream relationship and others may or may not be known. Known relationships are documented. ▪ *Known*: The entire set of immediate upstream relationships are known and documented.

Table 3. Taxonomy of a BoM tool (NTIA 2021c).

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 generates 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 application programming interface (APIs), object models, libraries, transport, or other reference sources
Transport	Advertise	Specifies how software or a device informs consumers that an SBoM is available
	Discover	Describes how the consumer learns of the location of the SBoM
	Access	Enables the transfer of the SBoM using the method derived from discovery

Table 4. Desired privacy and security properties in a BoM tool.

Category	Property
Encryption	Capability to encrypt data at rest, which is not being actively used and is stored in one location on hard drives, laptops, flash drives, or cloud storage
	Ability to encrypt data in transit, which is data active and flowing between devices and networks
Authentication	Ability to authenticate individual users and groups
Authorization	Ability to authorize individual users and at organizational level Capability to enable role-based access
Accounting	Ability to identify an executed operation and the *user* and its *organization* who executed the respective operation
	Ability to timestamp activities
	Ability to access client metadata
Auditability	Ability to provide transaction-level logging
	Capability to ensure tightly controlled deletions

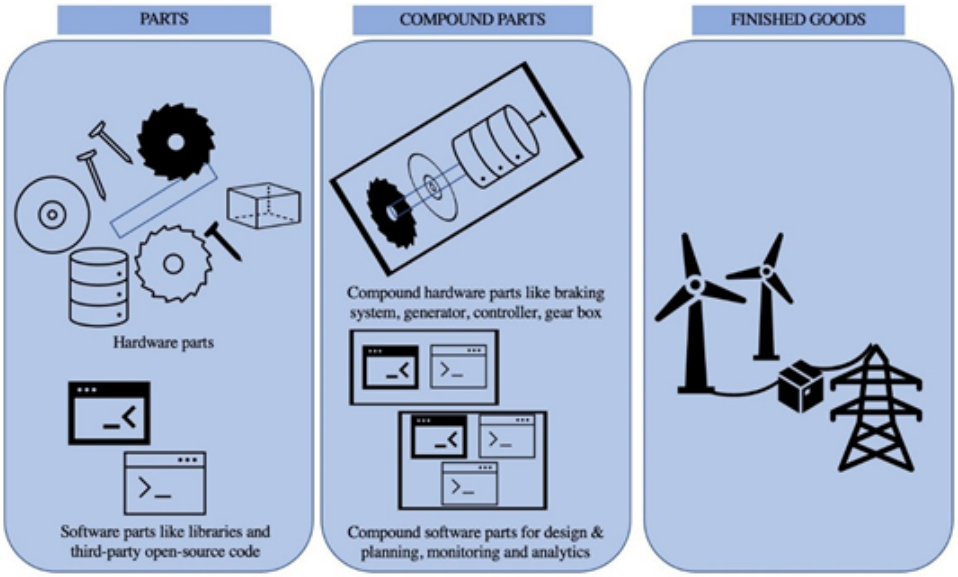
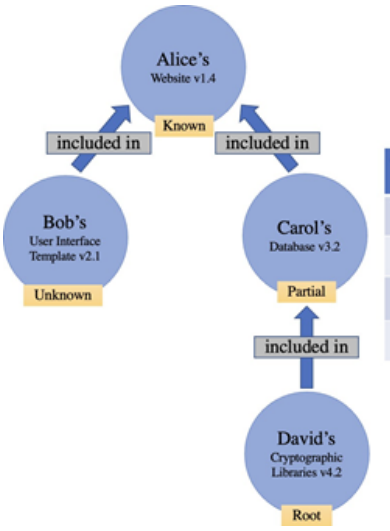


Figure 1. Hardware and software integration in a wind turbine.



Component	Supplier Name	Version String	Author	Hash	UID	Relationship
Website	Alice	1.4	Alice	0x234	280	self
⊣ Database	Carol	3.2	Carol	0x451	613	included in
⊣ Cryptographic Libraries	David	4.2	David	0x823	507	included in
⊣ UI Template	Bob	2.1	Bob	0x519	189	included in

Figure 2. Conceptual SBoM tree and table with upstream relationship assertions.