Comparing Bills of Materials

Lucas Tate
Pacific Northwest National Laboratory
lucas.tate@pnnl.gov

Rebecca Jones
Pacific Northwest National Laboratory

Doug Dennis
Pacific Northwest National Laboratory

Tatyana Benko
Pacific Northwest National Laboratory

Jody Askren
Pacific Northwest National Laboratory

***Abstract*—Bills of materials (BOMs) are quickly becoming an effective tool for managing supply chain risk. As more BOMs enter circulation, the ability to compare them will be crucial to understanding how products differ and in managing BOMs from different tools or sources. This paper will describe some of the challenges of comparing BOMs followed by a discussion of several comparison methods.**

***Index Terms*—bill of materials, BOM, HBOM, SBOM, comparison, graph comparison**

I. Introduction

Modern supply chains are increasingly complex. A supply chain for a single product can include multitudes of suppliers, manufacturers, distributors, and more. Common components that drive efficiency and reduce costs also serve to increase the potential damage of any one compromised component. This complexity poses significant challenges for managing risk because a vulnerability in any one component may have outsized consequences and the knowledge of what's inside any given product may be distributed across different companies or across the globe.

Understanding supply chains risks is more important today than it has ever been. Significant vulnerabilities and breaches continue to highlight the growing risks that supply chains face whether they were introduced maliciously or unintentionally. Well known events such as Log4 Shell [18] or Spectre/Meltdown [9] demonstrated how weaknesses in components could leave millions of products susceptible to attack. Solar winds [29] and the recent XZ Utils backdoor [10] demonstrated how malicious actors could subvert elements of a supply chain in an attempt to exponentially increase their reach. The increasing persistence and sophistication of supply chain threats requires new tools to combat them.

Recently, BOMs have been gaining traction as a tool to increase our supply chain understanding and help respond to this threat. A BOM contains details of the components that are used in building a product. While certainly not a silver bullet, understanding what is inside systems is a first step toward protecting them and responding once protections have failed. Work around software BOMs (SBOMs) far outpaces other proposed BOMs such as hardware (HBOMs) or artificial intelligence (AIBOMs) with regulations such as Executive Order 14028 [12] and the European Union (EU) Cyber Resilience Act (CRA) [6] helping to further global adoption. Encouragingly, interest in BOMs has spawned an abundance of new research dedicated to better understanding how to generate, exchange, store, and operationalize BOMs to improve risk management. As BOMs become ubiquitous, we anticipate a growing need for the ability to compare them which will be the focus of this paper.

Due to the nature of BOMs, comparing them is effectively looking at how the composition of two products differs. There are a variety of cases where that might be useful. It may be important to understand how the composition of a product changed with a version update or patch. Another use case might be evaluating a received BOM against an authoritative reference BOM. Comparisons can also be utilized to understand temporal changes that arise in dynamic systems or variation across a family of products. Beyond these use cases, we also find comparison methods useful for identifying inconsistencies in the creation of BOMs themselves. These inconsistencies are discussed later, with **the irony being that many of these inconsistencies that make comparison difficult are most easily discovered via comparison**.

In this work, we'll start by discussing previous work on comparing BOMs in Section II, followed by some of the barriers to comparing BOMs in Section III. Section IV discusses select methods for comparison followed by several examples that illustrate the comparison of two SBOMs in Section V and two HBOMs in Section VI. Lastly, Section VII will summarize our conclusions.

II. Previous Work

As the application of BOMs continues to evolve, particularly in the context of cybersecurity and supply chain management, significant research has been dedicated to understanding and improving how BOMs are generated, compared, and utilized.

Early research on BOMs predominantly focused on enhancing data management techniques to cope with the complexity of large-scale manufacturing environments, such as implementing a control system to manage BOMs throughout a company [25] and automating the creation of BOMs using an object-oriented model [2]. The object-oriented programming model, similar to the graph method used today, allows for semantic relationships that can create multi-level BOMs with sub-components of components [3]. However, since relational databases like SQL are ubiquitous, using them to create

BOMs became more common [20]. Algorithms were invented to automate the creation of BOMs by determining which products were in a product order and then pulling the required component information from the database of parts [1]. While this method is efficient, flexible and simple, it does not allow for parts explosion or complex computations, especially when analyzing or comparing BOMs [21].

To tackle the growing complexity of intricate products, advancements in BOM structures have been proposed. The multi-level BOM model is particularly effective in managing software, hardware, or system variations, supporting efficient design and planning [31]. Its strengths lies in the ability to handle complex, hierarchical data that can be represented as a graph.

Further development in this area focuses on using graph databases, which store node and relationships, instead of relational databases to integrate product development with production planning [13]. BOM management, found at companies such as Neo4j and OpenBOM take this approach of using a graph database.

The main advantage of turning BOMs into graphs is pairing the knowledge of what's inside something with information about how those components relate to each other. Using graphs, multiple BOMs can easily be combined to gain new information on a larger system, especially when combined with graph visualization tools. Even when used on a single BOM, graph analysis techniques can provide new insights into a system [4]. Another contribution to the graph-theoretic approach is where BOMs are converted into generic BOM graphs using data mining techniques [24]. This method leverages graph theory to identify common substructures within BOMs, facilitating the detection of component reuse across different products. While the ability to uncover hidden patterns and relationships within BOMs is useful, its application could be hindered by computational complexity and graph scalability.

There have been a few attempts to leverage graph theory for BOM comparison.

Graph based similarity analysis has been used to highlight the importance of reducing unnecessary production variations [26] and derived graph similarity metrics have been used to describe the similarity of two BOMs to place them into their product families [23, 17, 27]. Tree reconciliation, matching components in one graph to the components of another, has been used in biology to compare phylogenetic trees and extended to BOMs [16]. Similar methods can be applied to BOMs to create new products quickly [15]. This is all of the literature we could find on applying graph comparisons to BOMs.

However, graph comparisons have a rich history that could be explored for comparison of BOMs [7]. Work has been done on comparing and visualizing trees, a specific type of graph with no cycles [11]. This is might be especially applicable to HBOMs since they tend to be more hierarchical while SBOMs have a tendency to create loops making it less suitable. Comparison methods range from similarity metrics, like the ones referenced for BOM graph comparison to unknown node comparisons, which attempt to create a mapping between the nodes of the two graphs [19]. In the latter, some algorithms use attributes while others focus solely on the graph structure. Taking advantage of the attributes is more computationally complex, but is important when comparing BOMs due to the metadata often captured in BOM components. Since there is not a lot of research on efficient and effective ways of comparing BOMs, graph comparison literature may offer promising methods for future application research.

III. Barriers to Comparing BOMs

BOMs today are extremely heterogeneous which makes subsequent comparison very difficult. Before undertaking comparison, it's important to understand some of the sources of variability [28, 32]. While addressing these differences will be outside of the scope of this paper, considering them will likely be a prerequisite to meaningful comparison. This list is not exhaustive but captures some of the variability inherent in HBOMs and SBOMs.

A. BOM Standards and Versions

Currently in the field there are not single authoritative standards describing the structure or contents of an HBOM or an SBOM. For software, the NTIA Minimum elements [5] has been an influential guidance document outlining a set of generally accepted minimum elements. The two leading standards, SPDX [30] and CycloneDX [22], provide detailed schemas that describe a data structure for the capture of SBOM information but the mapping between them can be lossy. Despite the fact that BOMs have existed in manufacturing for decades, development of HBOM conventions has not reached full maturity. In addition to the CycloneDX and SPDX standards, the Information and Communications Technology (ICT) Supply Chain Risk Management (SCRM) Task Force and Department of Homeland Security Cybersecurity & Infrastructure Security Agency (DHS CISA) released a comprehensive HBOM framework that differs from those standards, although it attempts to provide mappings to them as applicable [8]. Even within the same format, major and minor versions describe BOM changes that can impede direct comparison.

B. SBOM Types

Despite some foundational work defining SBOM types, little has been done to formally differentiate them within real world SBOMs. The result is that two SBOMs for the same software can be markedly different. As an example, a *source* SBOM created directly from the source code will include named dependencies that are imported or loaded. This will look very different from a *build* SBOM which will describe a specific release and may include information on the build process and produced files.

C. Naming Conventions

Naming challenges permeate every aspect of BOM generation and despite being a known problem, it is extremely difficult to solve. Software names remain an open challenge.

Efforts such as the common platform enumeration (CPE) and package uniform resource locator (PURL) have helped machine-to-machine readability, but they deviate from how people would colloquially refer to software. Other information such as a vendor is complicated by lack of authoritative conventions. As an example 'MSFT', 'Microsoft Corporation', and 'Microsoft' are all defensible values but the inconsistent recording makes systematically disambiguating them difficult. As a last example, hardware component identifiers have a tendency to describe a family of components. This means that sub-strings of the name can still accurately describe components, but additional characters identify it with increasing specificity. The 'AD7579' from Analog Devices describes a LC^2MOS 10-Bit Sampling A/D Converter, but 'AD7579JN' distinguishes it as having a specific temperature range, integral nonlinearity, and package. Neither name is incorrect, but they utilize different levels of specificity which makes comparison more challenging.

D. Hashing Approaches

Well known hashing approaches such as MD5, SHA1, SHA256, SHA512 are extremely useful for providing easily matchable fingerprints of files. Their reproducible and static nature make them much more attractive in certain cases than names. One problem is that they are susceptible to dynamic information such as timestamps that often appear in files. Since hashes don't convey why the files are different, it won't be obvious whether the difference is meaningful in a specific comparison. Furthermore, existing standards offer a lot of flexibility in choosing hashing methods which means a different method might have been used from one SBOM to the next reducing comparability.

E. Structure

Structure in this context describes the relationships between components within a BOM. This structure gives us additional information such as where a dependency is introduced into our software or which board a specific component is mounted on. The problem is that methods for describing these structures are not rigid leading to expected variability in how they are described from one BOM to the next.

F. Scoping

In this context, scoping describes the boundaries of a BOM; what goes inside a particular BOM and what falls outside. This is a surprisingly hard problem. As an illustrative hardware example, we could describe a Raspberry Pi with an HBOM. If that Raspberry Pi is mounted inside a consumer product, should the HBOM for the consumer product include an external reference to the Raspberry Pi HBOM? Should it duplicate the information from the Raspberry Pi? From a software perspective if a software application requires the use of a shared library in the operating system, should that be included in the SBOM? The lack of a clear and accepted answer to these scoping questions result in variability that needs to be considered.

G. Quantities

Quantities are an interesting property that appear in HBOMs. Rather than listing a component *n* times, we can indicate how many of them are present with an integer value. However, if one HBOM opts to list the components individually and another HBOM leverages the quantity field, then you have to rectify these different representations when conducting a comparison.

H. Order

BOMs are unordered. This makes sense because there isn't a correct order to describe components. This property however immediately adds a lot of variability to the files which poses some challenges for simple comparative methods such as tabular comparison especially in conjunction with name variation that will stymie attempts to sort the data.

IV. Comparing BOMs

Unfortunately there is no single method that can be used to compare BOMs. Instead, strategies need to be specifically chosen based on the data available in the BOMs in conjunction with consideration of the questions that need to be addressed. In a simple example, if we want to understand the difference in *licenses* between two BOMs that utilize a well-formed ontology such as the SPDX License List [30], a set comparison using exact match of the values can be employed successfully.

In other cases, understanding quantities can be important. If we consider two HBOMs and want to understand how the components differ, we may opt for a list comparison of *component names* which will tell us if there are different components, as well as whether there were a different number of them used. We know from earlier discussion that *component names* can have a lot of inherent variability, so depending on the consistency of the data, a fuzzy matching technique may be needed. Fuzzy matching allows for some threshold of leniency in matching values that are 'close enough' at the expense of possibly making errant matches.

Direct element comparisons are not the only useful comparisons to be made. Creative use of redundant or complimentary information can be exploited to great effect. This is especially useful when comparing SBOMs where *component names*, *hashes*, *cpes*, or *purls* can be used together to gain additional insights. As an example, matching *hashes* provide some level of guarantee that the contents of a software component match. When compared to *component names* this can identify interesting situations where 1) *component names* are the same, but the contents differ, 2) *component names* are different, but the contents are the same, or 3) offer consensus between *component names* and *hashes*. These comparisons can often uncover unexpected results that are invaluable for assessing quality and consistency of BOMs.

So far, the comparisons that have been discussed implicitly assume a comparison of two similar BOMs, but other comparisons can be useful as well. SBOM practitioners will likely be intimately familiar with the variability of generation tools. Despite the monumental efforts around standardization,

SBOMs tend to vary greatly from one tool to the next. This can happen for many reasons, but some examples include inconsistent assumptions, different methods or levels of technical ability, different opinions on the boundaries of an SBOM, and opinions on whether transitive dependencies should be included. Further exacerbating the issue is the fact that the details driving the variability are often proprietary or black-box. By comparing the lists and sets of elements within the BOMs it is possible to gain insights into design choices and accuracy of various tools.

BOMs of different size can also be compared. This can suggest that the components of one BOM are a subset or contained within another BOM. It is also possible to explore subsets of a BOM; in a system with built-in redundancy it may be useful to look at how duplicated modules or sub-assemblies compare. Much of the previous work done on comparing BOMs has been used to cluster or identify product families that contain similar components in a similar structure. Comparing products within a family can lead to quicker generation of new products, as well as streamlining supply chain processes.

A. List and Set Comparisons

A straightforward approach to comparing two BOMs is by simply comparing lists. Due to the popularity of JSON and XML file types for use in BOMs, this will often require parsing and/or flattening of the data to obtain the unordered lists. An example this could be comparing all the *component names* in one BOM to the *component names* in another BOM. List comparisons aid in understanding differences and quantities of components which can be particularly useful if multiples of a single component are present. It should be noted that there are cases where comparing multiple elements simultaneously is necessary. For example, two manufacturers could use the same name for a particular component in which case it may be more useful to compare the manufacturer and name at the same time so as to differentiate one component from the other.

Beyond simple lists it can also be useful to only consider the unique values or sets. This representation sacrifices information about the frequency of values, but can greatly reduce the burden of comparison. Set comparison might be useful when looking at something like *licenses* where knowing that a license appears in *n* dependencies is probably less important than just having the list of unique licenses.

B. Graph Comparisons

Graph matching techniques can provide useful insight into the comparison of two BOMs. The list comparison approach does not take advantage of the relationships which are present between elements in a BOM. For example, if there are duplicates of a chip on a piece of hardware, set comparisons will not capture that information, while a graph comparison will. Importantly, it will also show where the chips are physically in the hardware. This can be done through text like the list comparison or crucially, visualization techniques that are intuitively easy to understand.

In order to take advantage of this visualization ability, the BOMs are converted to a graph by making the components into nodes and the relationships between two elements as edges. Information about each element can be recorded in the graph by associating node attributes, and relationship types can be given by edge attributes. Then a node mapping where one set of nodes is mapped to the other is created by using the edges and the node attributes.

V. SBOM Example

To illustrate the SBOM comparison methods, we used Trivy to generate SBOMs from two versions (3.6.4 and 3.7.0) of Thingsboard, an open-source IoT platform for data collection, processing, visualization, and device management. Thingsboard is largely written in Java and uses Maven to manage the project. Several modifications were made to the SBOMs. First, duplicate software components that shared the same purl and metadata were collapsed to a single component. If relationships existed to the duplicates that were removed, they were tied to the remaining copy. While the exact nature of the duplication in these SBOMs wasn't clear, it should be noted that removing them could hinder certain analyses such as finding multiple copies of a dependency. Next we opted to remove all the npm front-end dependencies. This was only done to make the SBOMs a little smaller for illustration.

The comparison of the resulting SBOMs (results in Table I) showed that out of the 230 components, there were 214 unique names detected in version 3.6.4 and 218 unique names from the 234 components in version 3.7.0. Interestingly, the number of unique purls was also four apart: 170 to 174. We note that the difference between the number of components and purls was due to 60 components that did not contain purls in the SBOM. The number of unique purls is identical to the total purls which is expected after the deduplication, meaning that each purl appears only once. There were 10 component names that were duplicated; each one had a different unique purl, with a total of 16 duplicates. Looking at the comparisons of the unique names, there were 201 names that appeared in both SBOMs. Finally, doing a Jaro-Winkler string comparison with a threshold of greater than 0.85 on the node names between the two sets resulted in a total of 887 matches.

TABLE I
COMPARISON RESULTS FOR SBOM WHERE 3.6.4 (ONLY) INDICATES THE DIFFERENCE BETWEEN 3.6.4 AND 3.7.0

	3.6.4	3.7.0	3.6.4 (Only)	3.7.0 (Only)
Name	230	234	13	17
Unique Names	214	218	13	17
Purls	170	174	158	162
Unique Purls	170	174	158	162

Comparing two SBOMs does not have to be constrained to differences in components. We also considered whether the licenses reported were different between the generated SBOMs. Because the primary interest is whether there are any different licenses to consider, using set comparison (comparing the unique values) of recorded license would seem the obvious

choice. Version 3.7.0 contains only a single unique license: Apache-2.0. However, version 3.6.4 contains two unique licenses: Apache-2.0 and MIT. While this may look like version 3.6.4 is more complete, it is also important to know that only six dependencies in the version 3.6.4 SBOM had a license recovered by the tool. Version 3.7.0 had four dependencies with recovered licenses. This largely indicates that neither SBOM has a complete picture of the licensing exposure in Thingsboard.

The potential lack of information prompted us to manually review the licenses for the listed dependencies. We discovered that both versions of the software contained several other licenses such as the Eclipse Public License (EPL) and the Lesser GNU Public License (LGPL). These licenses have additional disclosure and representation requirements that may not be satisfied with the same rules as Apache-2.0 or MIT. Additionally, it is not inconceivable that an organization may apply additional scrutiny to licenses from the GNU Public License (GPL) family and would want to know that LGPL code is being used. While set comparison identified a notable difference in recorded licenses, a list comparison would have highlighted how few of the components captured license information.

We also examined the organizations of the dependencies. They were detected by using the first two segments of the package name in the purls. For example, given the purl "pkg:maven/com.example.foo@1.2.3", the organization is "com.example". There were fifty unique organizations in version 3.6.4 and 52 in version 3.7.0. Excluding Java standard library packages, we found that when moving to version 3.7.0 Thingsboard gained four additional external organizations and lost one. This information could be useful for situational awareness or subsequent corporate analysis where some producers may imply an increased/decreased level of assurance.

For the graph comparison, we converted the SBOMs into graphs using components as nodes and dependencies as edges. We then merged them on the name field using a depth-first search matching algorithm [14] with exact matching. A quick calculation shows us that 217 nodes were matched, 13 appeared only in version 3.6.4 and 17 appeared in version 3.7.0, matching the comparison in Table I. The visualization of the compared graphs did not add to the analysis and was therefore not included. To understand if the identified differences between the two SBOMs was due to small variance in the names, we employed fuzzy matching. It should be noted that with a priori knowledge of the name structure, some fuzzy matching approaches may be more successful, but here we naively employed Jaro-Winkler on the node names with an arbitrary threshold of .85 and found that there are 7 similar packages (see Table II). While we note that the graph wasn't particularly useful for visualization, the added constraint of the structure reduced the number of possible matches by 880 because rather than just finding similar names, the structure requires the names to also be in the same place as defined by the relationships in the SBOM.

In comparing these SBOMs, a combination of methods proved useful. List and set comparisons provided useful characterization of the SBOMs and identified some license irregularities. The graph method allowed us to ignore dependencies with similar names and focus on the differences we are more interested in, which were updated dependencies.

TABLE II
SIMILAR NAMES OF PACKAGES IN EACH THINGSBOARD VERSION WITH THE DIFFERENCES HIGHLIGHTED.

3.6.4	3.7.0
bcpkix-jdk15on	bcpkix-jdk18on
bcprov-jdk15on	bcprov-jdk18on
commons-collections	commons-collections4
hypersistence-utils-hibernate-55	hypersistence-utils-hibernate-63
javax.annotation-api	jakarta.annotation-api
swagger-annotations	swagger-annotations-jakarta
springfox-boot-starter	spring-boot-starter-webflux

VI. HBOM EXAMPLE

In this example, two HBOMs were created from two distinct instances of the same hardware product. The identities of the devices and components have been obfuscated, but the real characteristics of the comparison were preserved. Visualizations of the two graphs are shown in Figures 1 and 2.

A list comparison of the component names for the HBOMs immediately conveys differences shown in Table III. Despite being the same product, we see 35 components that only appear in HBOM 1 and 46 components that only appear in HBOM 2. Because the comparison of unique names reflects different counts, we can infer that some of the differences include components that appeared multiple times. In reviewing the differences, the most noteworthy finding was that one of the components was a circuit board which was especially surprising. Fuzzy matching in this case was not particularly useful because it flagged 743 possible matches which is difficult to sift through.

A comparison of the vendor revealed that there were 17 unique vendors in HBOM 1 and 14 unique vendors in HBOM 2. with only 13 vendors shared between the two. We offer no explanation as to why the vendors differ, but interestingly despite the BOMs representing the same product, the component supply chain looks different and may result in varying levels of risk exposure.

TABLE III
COMPARISON RESULTS FOR HBOM EXAMPLE WHERE HBOM 1 (ONLY) INDICATES THE DIFFERENCE BETWEEN HBOM 1 AND HBOM 2

	HBOM 1	HBOM 2	HBOM 1 (Only)	HBOM 2 (Only)
Name	156	169	35	46
Unique Names	99	108	17	26

Repeating the same graph comparison approach as in Section V, we generated the merged graph shown in Figure 3. Blue nodes indicate that the node names matched exactly, while the thick yellow edges indicate the node names matched approximately (Jaro-Winkler with a threshold of .85).

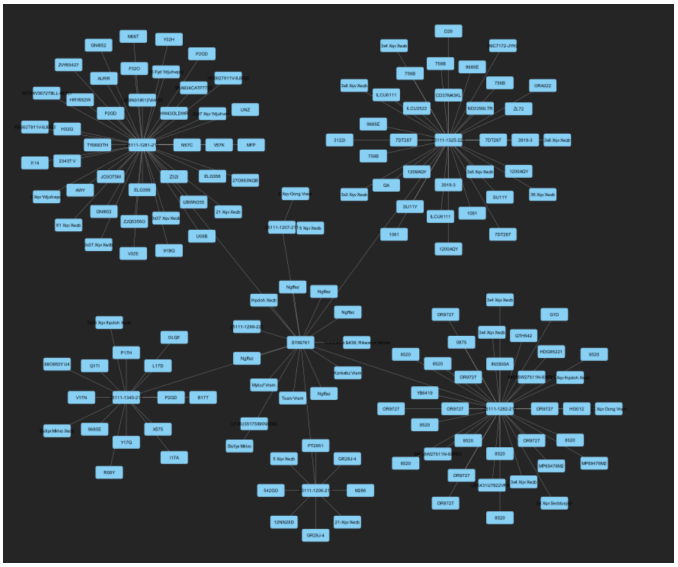

Fig. 1. Visualization of HBOM 1.

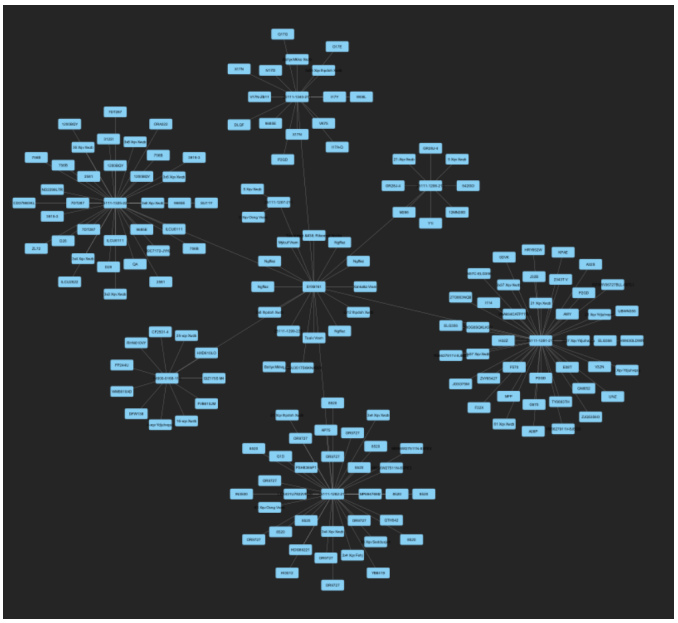

Fig. 2. Visualization of HBOM 2

This visualization is immediately useful. The most notable difference is the presence of a yellow circle in the top of Figure 3. This turned out to be the additional circuit board that was unexpectedly present in one of the devices, and we can quickly understand which of the different components correlate to the addition of that board. The remaining differences are drastically reduced by enforcing the graph structure (e.g. the component must be on the same board). The differences identified fell into one of three categories:

1) The component name was transcribed incorrectly. For example, the names were recorded as $IN3S00A$ and $IN3500$, where a 5 is switched for an S.
2) The difference was real and describes a component that had been switched out in production with a different but equivalent component.
3) The name in one HBOM was recorded with more specificity than the name of the equivalent component in the other HBOM. One component was named $V17N$ and one was recorded as $V17N - ZB11$.

Fuzzy match on the component names using the graph was particularly useful for this comparison. Instead of 743 possible matches, there are only 21 matches, identifying transcription errors and incremental component variations with high precision. As with the SBOM comparison, leveraging multiple comparison methods proved to be useful, but notably the graph provided significantly increased utility in the hardware example.

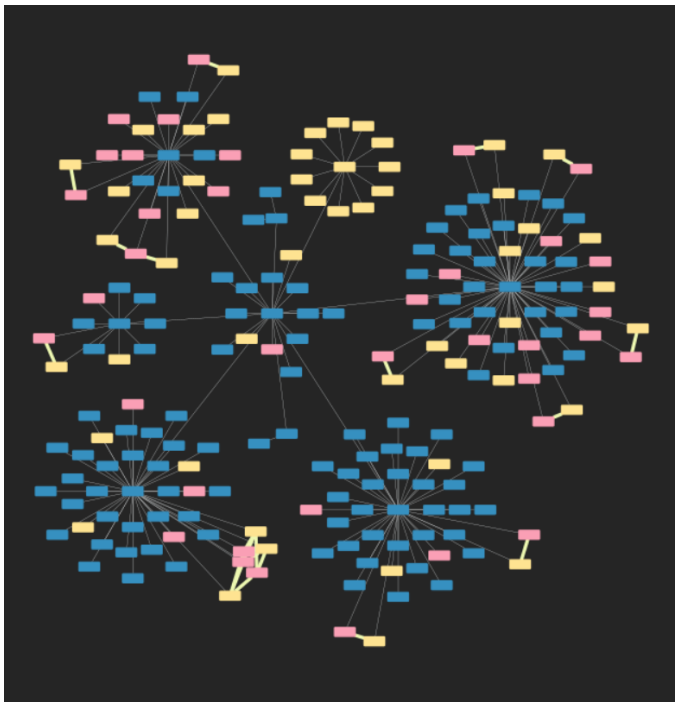

Fig. 3. Visualization of Merged Graph. Blue represents nodes found in both HBOM 1 and HBOM 2. Pink nodes are only found in HBOM 1 and yellow nodes are only in HBOM 2. Yellow edges indicate nodes where the names fuzzy matched.

VII. Conclusion

This paper provides an introductory discussion of the methods and challenges associated with comparing BOMs. Despite the recent abundance of energy and research in BOMs by government, industry, and academia, tools and methods to effectively compare BOMs lag behind. There is no single method of comparison that can effectively compare BOMs today. List, set, and graphical comparisons are complimentary and contribute to a foundational capability. As reference BOMs become more readily available, comparison methods will be vital to leveraging them. Ultimately BOM adoption, spurred by policy and regulation will continue to grow. The

ability to compare BOMs will be essential for understanding and reasoning about BOMs for supply chain risk management.

Acknowledgment

The authors wish to thank the Department of Energy (DOE) Cybersecurity, Energy Security, and Emergency Response (CESER) and the Cyber Testing and Resilience of Industrial Control Systems (CyTRICS) Program including Idaho National Laboratory (INL), Lawrence Livermore National Laboratory (LLNL), National Renewable Energy Laboratory (NREL), Oakridge National Laboratory (ORNL), and Sandia National Laboratory (SNL). Information Release: PNNL-SA-202952.

References

[1] A. O. Aydin and A. Güngör *. "Effective relational database approach to represent bills-of-materials". In: *International Journal of Production Research* 43.6 (2005), pp. 1143–1170. DOI: 10.1080/00207540512331336528.

[2] Sheng-Hung Chang, Wen-Liang Lee, and Rong-Kwei Li. "Manufacturing bill-of-material planning". In: *Production Planning & Control* 8.5 (Jan. 1997), pp. 437–450. ISSN: 0953-7287, 1366-5871. DOI: 10.1080/095372897235019.

[3] Yunkung Chung and Gary W. Fischer. "A conceptual structure and issues for an object-oriented bill of materials (BOM) data model". In: *Computers Industrial Engineering* 26.2 (1994), pp. 321–339. ISSN: 0360-8352. DOI: https://doi.org/10.1016/0360-8352(94)90065-5.

[4] Matteo Cinelli et al. "A network perspective for the analysis of bill of material". In: *Procedia CIRP* 88 (2020), pp. 19–24. ISSN: 22128271. DOI: 10.1016/j.procir.2020.05.004.

[5] The United State Departement of Commerce. *The Minimum Elements For a Software Bill of Materials (SBOM)*. July 12, 2021. URL: https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf.

[6] European Commission. *Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on horizontal cybersecurity requirements for products with digital elements and amending Regulation (EU) 2019/1020*. Sept. 15, 2022. URL: https://eur-lex.europa.eu/resource.html?uri=cellar:864f472b-34e9-11ed-9c68-01aa75ed71a1.0001.02/DOC_1&format=PDF.

[7] D. CONTE et al. "THIRTY YEARS OF GRAPH MATCHING IN PATTERN RECOGNITION". In: *International Journal of Pattern Recognition and Artificial Intelligence* 18.03 (2004), pp. 265–298. DOI: 10.1142/S0218001404003228.

[8] Cybersecurity and Infrastructure Security Agency. *A Hardware Bill of Materials (HBOM) Framework for Supply Chain Risk Management*. Sept. 2023. URL: https://www.cisa.gov/sites/default/files/2023-09/A%20Hardware%20Bill%20of%20Materials%20Framework%20for%20Supply%20Chain%20Risk%20Management%20%28508%29.pdf.

[9] Cybersecurity and Infrastructure Security Agency. *Meltdown and Spectre Side-Channel Vulnerability Guidance*. May 1, 2028. URL: https://www.cisa.gov/news-events/alerts/2018/01/04/meltdown-and-spectre-side-channel-vulnerability-guidance.

[10] Akamai Security Intelligence Group. *XZ Utils Backdoor — Everything You Need to Know, and What You Can Do*. Apr. 1, 2024. URL: https://www.akamai.com/

blog/security-research/critical-linux-backdoor-xz-utils-discovered-what-to-know.

[11] John Alexis Guerra-Gómez et al. "TreeVersity: Interactive Visualizations for Comparing Hierarchical Data Sets". In: *Transportation Research Record: Journal of the Transportation Research Board* 2392.1 (Jan. 2013), pp. 48–58. ISSN: 0361-1981, 2169-4052. DOI: 10.3141/2392-06. URL: http://journals.sagepub.com/doi/10.3141/2392-06.

[12] The White House. *Executive Order on Improving the Nation's Cybersecurity*. The White House. May 12, 2021. URL: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/.

[13] Xiaodu Hu et al. "Graph Model Based Bill of Material Structure for Coupling Product Development and Production Planning". In: *Intelligent and Transformative Production in Pandemic Times*. Cham: Springer International Publishing, 2023, pp. 593–605. DOI: 10.1007/978-3-031-18641-7_55.

[14] Rebecca Jones and Lucas Tate. "Visualizing Comparisons of Bill of Materials". In: *2023 IEEE Symposium on Visualization for Cyber Security (VizSec)*. 2023, pp. 12–16. DOI: 10.1109/VizSec60606.2023.00008.

[15] M. Kashkoush and H. ElMaraghy. "Product Design Retrieval by Matching Bills of Materials". In: *Journal of Mechanical Design* 136.1 (Jan. 1, 2014), p. 011002. ISSN: 1050-0472, 1528-9001. DOI: 10.1115/1.4025489.

[16] Mohamed Kashkoush and Hoda ElMaraghy. "Matching Bills of Materials Using Tree Reconciliation". In: *Procedia CIRP* 7 (2013), pp. 169–174. ISSN: 22128271. DOI: 10.1016/j.procir.2013.05.029.

[17] Mohamed Kashkoush and Hoda ElMaraghy. "Product family formation by matching Bill-of-Materials trees". In: *CIRP Journal of Manufacturing Science and Technology* 12 (Jan. 2016), pp. 1–13. ISSN: 17555817. DOI: 10.1016/j.cirpj.2015.09.004.

[18] Edward Kost. *Log4Shell: The Log4j Vulnerability Emergency Clearly Explained*. June 20, 2023. URL: https://www.upguard.com/blog/apache-log4j-vulnerability.

[19] S. Melnik, H. Garcia-Molina, and E. Rahm. "Similarity flooding: a versatile graph matching algorithm and its application to schema matching". In: *Proceedings 18th International Conference on Data Engineering*. 2002, pp. 117–128. DOI: 10.1109/ICDE.2002.994702.

[20] G. Nandakumar. "The design of a Bills of Material Processor using a relational data base". In: *Computers in Industry* 6.1 (1985), pp. 15–21. ISSN: 0166-3615. DOI: https://doi.org/10.1016/0166-3615(85)90066-1.

[21] Ganesan Nandakumar. "Bills of material processing with a SQL database". In: *Computers Industrial Engineering* 18.4 (1990), pp. 471–483. ISSN: 0360-8352. DOI: https://doi.org/10.1016/0360-8352(90)90005-7.

[22] OWASP Foundation. *OWASP CycloneDX Software Bill of Materials (SBOM) Standard*. URL: https://cyclonedx.org/.

[23] C.J. Romanowski and R. Nagi. "On Comparing Bills of Materials: A Similarity/ Distance Measure for Unordered Trees". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 35.2 (Mar. 2005), pp. 249–260. ISSN: 1083-4427. DOI: 10.1109/TSMCA.2005.843395.

[24] Carol J. Romanowski and Rakesh Nagi. "A data mining and graph theoretic approach to building generic bills of materials". In: 2002. URL: https://api.semanticscholar.org/CorpusID:17069084.

[25] P.S. Rusk. "The role of the bill of material in manufacturing systems". In: *Engineering Costs and Production Economics* 19.1 (May 1990), pp. 205–211. ISSN: 0167188X. DOI: 10.1016/0167-188X(90)90044-I.

[26] Michael Schmidt et al. "Graph-based similarity analysis of BOM data to identify unnecessary inner product variance." In: 21st International Conference on Engineering Design (ICED17). Vol. 1. Vancouver, Canada, Aug. 2017.

[27] Han M. Shih. "Product structure (BOM)-based product similarity measures using orthogonal procrustes approach". In: *Computers & Industrial Engineering* 61.3 (Oct. 2011), pp. 608–628. ISSN: 03608352. DOI: 10.1016/j.cie.2011.04.016.

[28] Trevor Stalnaker et al. "BOMs Away! Inside the Minds of Stakeholders: A Comprehensive Study of Bills of Materials for Software Systems". In: ICSE '24: IEEE/ACM 46th International Conference on Software Engineering. Feb. 6, 2024, pp. 1–13. DOI: 10.1145/3597503.3623347.

[29] Dina Temple-Raston. *A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds Hack*. https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack. Last accessed 2023-06-30. Apr. 16, 2021. URL: https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack.

[30] The Linux Foundation Projects. *International Open Standard (ISO/IEC 5962:2021) - Software Package Data Exchange (SPDX)*. URL: https://spdx.dev/.

[31] Yao Wang, Wei Guo Wang, and Song Mao. "A New Type of BOM Model and its Application". In: *Applied Mechanics and Materials* 347-350 (Aug. 2013), pp. 1234–1238. ISSN: 1662-7482. DOI: 10.4028/www.scientific.net/AMM.347-350.1234.

[32] Boming Xia et al. "An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead". In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, May 2023, pp. 2630–2642. DOI: 10.1109/ICSE48619.2023.00219.