The mission of the Automating Compliance Tooling (ACT) project umbrella is to support development of reference tooling for efficient and effective exchange of software bills of materials to enable compliance, security, export control, pedigree and provenance workflows. The projects that make up ACT participate in a technical advisory committee (TAC) and have discussed DEPARTMENT OF COMMERCE National Telecommunications and Information Administration [Docket No. 210527–0117] RIN 0660–XC051 Software Bill of Materials Elements and Considerations and would like to provide the following feedback.

If you have any questions or require clarification, please contact me, Rose Judge, ACT TAC Chair, at rjudge@vmware.com.

## 1. Are the elements described above, including data fields, operational considerations, and support for automation, sufficient *[Table 1]*? What other elements should be considered and why *[Table 2]*?

**Table 1: Feedback for existing elements**

| *Elements above* | *Description* | *Comments* |
|---|---|---|
| **Dependency relationship** | Refers to the idea that one component is included in another component | This definition of "dependency" is too narrow. SPDX has a list of around 50 dependency types (CONTAINS, obviously, but also DEPENDS_ON, GENERATED_FROM, et. etc.). And this is before we tackle the issue of AI/ML (models, training data, and all this wonderful stuff) |
| **Cryptographic hash of the component** | An algorithmic hash that verifies the identity of the package | “Cryptographic hash of the component” is a vague definition and does not reflect the way software packages are identified in their specific ecosystem. For example, one way of generating a checksum of a collection of files is to checksum the files, sort the list, and checksum the list, while other methods use a merkle tree to generate a package checksum, while still others just checksum an archive of the collection of files. It makes more sense to define a package format such as “npm”, “wheel”, “oci”, etc and then the verification string. |
| **Unique Identifier** | A string that uniquely identifies a component | Is “Unique Identifier” local to the SBOM or can this be a Globally Unique Identifier? It would be useful to have a way of uniquely identifying the SBOM document itself, helping teams reuse SBOM documents along with the software they describe. |

**Table 2: New element considerations**

| *New Elements* | *Description* | *Reason for consideration* |
|---|---|---|
| **Time Stamp** | When the SBOM was created | Understanding when the SBOM is created is useful to understand the most current version, and what is known at a point in time. |
| **SBOM License** | License for the SBOM document itself | To understand how the data in the SBOM is able to be used. Note this does not restrict the ability for other confidentiality terms being applied. |
| **Origin** | Where the component was acquired from | Essentially: where did you get this component from?<br>Note: if you're using the binary openssl libs from Ubuntu, the origin is "upstream Ubuntu repo", NOT openssl.org |
| **Licensing and copyright information** | The license and copyright text that governs a software artifact | To accommodate the additional use case of legal compliance, stated below. |

## 2. Are there additional use cases that can further inform the elements of SBOM?

**Legal Compliance**. Therefore, there is a need to include relevant information. For most open source licenses, this translates to "license and attribution".

**Trade Compliance**. Several countries have export and customs regulations (ECC, export control and customs) which needs to be also considered for software delivery. As such, software components can have a classification according to the internationally used export classification system for goods. Based on a SBOM, the relevant components for export regulations can be identified and the classification of the delivery can be applied accordingly.

## 3. SBOM creation and use touches on a number of related areas in IT management, cybersecurity, and public policy. We seek comment on how these issues described below should be considered in defining SBOM elements today and in the future.

*a. Software Identity: There is no single namespace to easily identify and name every software component (be it sources or binaries). The challenge is not the lack of standards, but multiple standards and practices in different communities.*

There’s no such thing as a unique global identifier that scales. Using the multiple existing identifiers and supplementing them with content-based identifiers works well but there is no perfect solution. We think it is a bad idea to create one global unique identifier to try to identify every software artifact, but instead recognize a set of identifiers that the tooling should encompass.

*b. Software-as-a-Service and online services: While current, cloud-based software has the advantage of more modern tool chains, the use cases for SBOM may be different for software that is not running on customer premises or maintained by the customer.*

We agree. Some of the minimum elements of an SBOM are missing the consideration of a timestamp and being able to track what is executing at a point in time which is critical in a SaaS environment.

*c. Legacy and binary-only software: Older software often has greater risks, especially if it is not maintained. In some cases, the source may not even be obtainable, with only the object code available for SBOM generation.*

It would be useful to have the ability to flag components when information is incomplete, so it may be considered. SPDX has a concept of "no assertion" to indicate that there is not enough information to declare about the component.

*d. Integrity and authenticity: An SBOM consumer may be concerned about verifying the source of the SBOM data and confirming that it was not tampered with. Some existing measures for integrity and authenticity of both software and metadata can be leveraged.*

We agree this is an issue and would like to have some mechanisms agreed to that are easy to incorporate into tools (without concerns about patents, etc.). Ideally, trusted open source libraries that tooling can use.

*e. Threat model: While many anticipated use cases may rely on the SBOM as an authoritative reference when evaluating external information (such as vulnerability reports), other use cases may rely on the SBOM as a foundation in detecting more sophisticated supply chain attacks. These attacks could include compromising the integrity of not only the systems used to build the software component, but also the systems used to create the SBOM or even the SBOM itself. How can SBOM position itself to support the detection of internal compromise? How can these more advanced data collection and management efforts best be integrated into the basic SBOM structure? What further costs and complexities would this impose?*

This is a work in progress that projects like In-toto and the SPDX 3.0 Profiles are trying to address.

*f. High assurance use cases: Some SBOM use cases require additional data about aspects of the software development and build environment, including those aspects that are enumerated in Executive Order 14028. How can SBOM data be integrated with this additional data in a modular fashion? Exec. Order No.14028 § 4(e)(i) – (x), 86 Fed. Reg. 26,633, 26,638 – 39 (May 12, 2021).*

The projects under the ACT umbrella align with the goals of section 4 of the Executive Order ("*Enhancing Software Supply Chain Security*") and aim to support development of reference tooling for efficient and effective exchange of software bills of materials to enable compliance, security, export control, pedigree and provenance workflows.

*g. Delivery. As noted above, multiple mechanisms exist to aid in SBOM discovery, as well as to enable access to SBOMs. Further mechanisms and standards may be needed, yet too many options may impose higher costs on either SBOM producers or consumers.*

It is an important topic. It is important to have known discovery processes for different types of software delivery that can be automatically used by tools.

*h. Depth. As noted above, while ideal SBOMs have the complete graph of the assembled software, not every software producer will be able or ready to share the entire graph.*

Being able to signal what is known, and what not, is important for setting expectations, so the capability needs to be possible to express. Most build tools are meant to build code and not to produce an SBOM. As a result, Software Composition Analysis tools on the market generally do a best effort approach. The default notion in an SBOM should be that the graph is incomplete unless explicitly stated e.g. a human has reviewed it.

*i. Vulnerabilities. Many of the use cases around SBOMs focus on known vulnerabilities. Some build on this by including vulnerability data in the SBOM itself. Others note that the existence and status of vulnerabilities can change over time, and there is no general guarantee or signal about whether the SBOM data is up-to-date relative to all relevant and applicable vulnerability data sources.*

It needs to be possible to represent known vulnerabilities in an SBOM at a **specific point in time.**
At a minimum, the SBOM should have the possibility to provide the identifiers of software components which are used to reference vulnerability information and can be re-used at a later time to get an updated list of vulnerabilities.

*j. Risk Management. Not all vulnerabilities in software code put operators or users at real risk from software built using those vulnerable components, as the risk could be mitigated elsewhere or deemed to be negligible. One approach to managing this might be to communicate that software is "not affected" by a specific vulnerability through a Vulnerability Exploitability eXchange (or "VEX"),but other solutions may exist.*

Exploitability is something that should be possible to signal about an SBOM, rather than a property about vulnerabilities.

**4. Flexibility of implementation and potential requirements. If there are legitimate reasons why the above elements might be difficult to adopt or use for certain technologies, industries, or communities, how might the goals and use cases described above be David Braue, Software 'Bill of Materials' To Become Standard?, Info. Age (Oct. 22, 2020, 11:34 AM), https://ia.acs.org.au/article/2020/software-bill-of-materials-to-become-standard.html. fulfilled through alternate means? What accommodations and alternate approaches can deliver benefits while allowing for flexibility?**

There needs to be open source solutions and tooling available for full supply chain coverage, as not all organizations publishing software (including universities, open source projects, individuals) have access to commercial tooling. There also needs to be open data sources to avoid duplication of effort.

Feedback provided by ACT project participants:

| **Person** | **Tool** |
|---|---|
| Nisha Kumar (VMware) | Tern |
| Alexios Zavras (Intel) | SPDX |
| Michael Jaeger (Siemens) | SW360, FOSSology |
| Philippe Ombredanne (AboutCode.org and nexB Inc.) | ScanCode Toolkit and AboutCode projects |
| Rose Judge (VMware) | Tern |
| Thomas Steenbergen (HERE Technolgies) | OSS Review Toolkit and SPDX |