Software Identification Challenge and Guidance

Draft version 0.2
2020-10-22

This document is available at:
https://docs.google.com/document/d/1tOtO90AIrsHSIPvcVhvSBmkuS1mkFWIYWFNlPnjSzhQ

Introduction

Possibly the biggest single challenge to supply-chain transparency and the SBOM model is the difficulty in identifying software components globally with sufficient uniqueness. This paper captures some of the major challenges of global software component identification and offers some guidance on how to address these challenges. For more background on SBOM, please see the NTIA Software Component Transparency document library.[1]

The terms in this document reflect the definitions in Framing Software Transparency[2] unless otherwise noted.

Global Software Component Identification

Currently, there are no globally authoritative sources to obtain component identity from SBOM information. As such, two different SBOM authors could use two different identifiers for the same component, for example, one might use `com.sun.java` while another may use `com.oracle.java`, depending on when they started keeping their records. The reverse problem can also occur if two authors use the same identifier for different components.

Lack of clarity about component identity can also make it difficult to map an SBOM component to vulnerability, license, or other data of interest to an SBOM consumer.

In the first multi-organization SBOM proof of concept exercise in 2019,[ref] the lack of common identifiers was noted as a key obstacle for automation.

In a perfect world, the SBOM of a given component would be the source of the authoritative, sufficiently-globally-unique “correct” identity of that component.

[1] https://www.ntia.gov/sbom

[2] https://www.ntia.doc.gov/files/ntia/publications/ntia_naming_use_cases_-_framing_2020-04-11.pdf

Suppose an SBOM serves to help organizations track their third party components. In that case, some path towards convergence of identifiers is needed, such that two components can be identified as the same or different in largely automated systems. A single, universally used set of identifiers may not be possible, especially in the near term, because there is not currently a single or central naming authority.

The reason for the current heterogeneity in naming extends beyond the lack of a single canonical namespace. The creators of software components define names according to their own needs. Tools have developed their own approaches. Several standards have emerged over the years, including Common Platform Enumeration (CPE), SWID tags, Package URLs, and Software Heritage ID. As organizations keep their own records and integrate this data into their build processes, they might use some of these standards or define their own internal schema. Component names assigned by the original supplier cannot even be assumed to be static, as dynamics like corporate acquisitions and project forking may lead to a change in a preferred name. Moreover, the challenges of defining a namespace and the identity of digital artifacts themselves is a known hard problem.

A software component can be identified by many attributes. The Framing document specifies Supplier Name, Component Name, and Component Version. The issue of component name and identity is closely related to the challenge of identifying the component supplier. In many cases, if a supplier can effectively determine the namespace of the software they write, the supplier can effectively resolve any confusion about naming. A supplier is not just the source of the code, but often plays related roles, including vulnerability reporting, version maintenance, and licensing. However, the supplier name may not be clear cut either. For example, a commercial software company could be referred to by both Acme and Acme_Software. Project reorganization, corporate acquisition, and open-sourcing of commercial packages also further complicate long term supplier identifiers.

Name, Namespace, Identification

Commented [1]: Possibly you might want to consider this as "Relative Component Identification" Namespaces for the SBOM documents and unique identification of a component within.

Commented [2]: section meant as sidebar of terms

Commented [3]: point 1: how "name" and "identity" are related, point 2: define "namespace"

Commented [4]: but also added some text about federating namespaces

The core issue this document addresses is global identification of software components. Naming is part of, but the equivalent of identification. A name is one of many attributes that can help to identify a component. A name alone may or may not be sufficient, and components can have more than one name. A comprehensive component identification system will need to handle multiple names for the same component, likely through an alias or equivalency relationship.

A namespace is a way to partition names or identifiers. Component names within a namespace are unique. URLs and DNS are a common example of hierarchical namespaces. For example:

www.example.com and www.example.net represent two components named www, but the components are different because they fall within different namespaces (example.com and example.net). example also refers to two different components under different namespaces (.com and .net).

It is unrealistic at present to expect a single global namespace for all software components. As demonstrated in other large scale identification systems (e.g., DNS and XML namespaces), a scalable solution will involve a way to federate and interrelate different hierarchical namespaces.

Need for Component Identification

SBOM component identity information may be generated and changed under several different conditions. While a full set of use cases is outside the scope of this document,[3] (as are other key issues like data integrity or how to share SBOM data), it is important to acknowledge some of the more common use cases and the actors with key roles.

Primary SBOM Authorship

A supplier creates an SBOM for a primary component. The supplier is the creator of the component. The supplier defines its own identifier and component name as author of the component data. The supplier is the source of truth for the baseline elements and any other information associated with the primary component. If the entire ecosystem were to adopt this approach, the recursive use of SBOM data would make SBOM assembly relatively straightforward, as long as those SBOMs were made readily available..

Secondary SBOM Authorship

In cases where the supplier has not created an SBOM, or there is too much uncertainty about the component data, an SBOM stakeholder who is not the authoritative supplier can author the SBOM data. The author may be a consumer of SBOM data such as a downstream supplier or an end user, or the author may provide SBOM data as part of other services, such as a source composition analysis (SCA) tool. The author will make a best-effort approach to establish the baseline information, prefer existing identification systems and formats (see below), and follow other SBOM guidance. An SBOM consumer will be able to determine that the supplier of the component did not create the SBOM because the author name will not be the same as the supplier name.

Commented [5]: If the supplier is the author/manufacturer of the component, who is then 'SBOM stakeholder' and how relations between them are defined?

Is this indeed feasible that the enterprise customer authors the SBOM data and has so detailed information about the component?

If the SBOM data is a part of the IP, how legally can this data be available to anyone besides the supplier?

SBOM Assembly

A supplier using a third party component includes SBOM data about their components in their SBOM. They use component data from the primary supplier (preferred) or from another author when the primary information is not available. If component SBOM is not available, a best effort approach should be used to establish component information. The supplier name and

[3] For a more complete discussion, see *Roles and Benefits for SBOM across the Supply Chain* https://www.ntia.gov/files/ntia/publications/ntia_sbom_use_cases_roles_benefits-nov2019.pdf

component name should follow best practices (see below). If no component information is available, incompleteness of the dependency data should be noted in the SBOM.

SBOM assembly depends significantly on the capability to share and exchange SBOMs.[4]

Guidance for Component Identification

The goal of the following guidance is to reduce the number of different identifiers for software components in SBOMs. This model follows builds on existing practices and organizational structures with minimal adoption of new technology or changing practices. This guidance is particularly aimed at the secondary authorship use case above, where an SBOM stakeholder must determine the appropriate data for component fields in an SBOM they create.

It follows a two-part test. First, if an authoritative source of identifying information exists, the author should use it, enabling a federated approach that reflects the supplier of the software as the optimal source of information. Failing that, an SBOM author should use one of a small set of established naming practices, as outlined below.

These two cases alone will not completely solve global component identification. For instance, SBOM formats typically have some way to denote component identity, but the formats themselves may not define a namespace, naming convention, or an identification system.

Preferred case – Existing supplier and namespace

If there exists an established, well-defined identification system, the SBOM author should use that system. This would include established package managers with unique internal names of components. This would also include commercial suppliers that clearly communicate the identity of their components. Lacking an existing established format, suppliers are encouraged to use one of the widely accepted SBOM formats below,

In cases where it is not clear that two components are identical, they should be treated as separate components. SBOM authors should select the component that most closely aligns with their development and maintenance processes, such as the source from where they will obtain future versions of a component.

Examples of existing identification namespaces include:

- npm (package manager for Node.js)
- GitHub
- Java Language Specification (JLS)[5]
- A commercial example? Oracle, Microsoft?

[4] Sharing and Exchanging SBOMs https://docs.google.com/document/d/1XuGix4AIcXKqPlPVMvjQEj7zybvnYy4DkgpYfF2EwRc
[5] https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html

Alternate case – Established software identity standard

If an existing, widely accepted name does not exist, then using an existing namespace will not be sufficient to help identify a component. A similar challenge arises if an identifier is not available through a canonical source such as a widely used package manager, or there are multiple widely-used identifiers for a given piece of software.

While a solution for a universal single name is not currently in sight, r the following best practices are good options to establish to 1) use an established identity standard that is 2) used by some reasonable subset of the existing community.

Using an established identity standard helps the community move towards greater automation by using existing data formats and predictability Using an existing identifier moves us closer towards convergence of a namespace. That is, any secondary SBOM author should avoid creating a novel component identifier if at all possible.

This community recommends one of the below identity standards:

- SWID tags[6]
- Package URL (PURLs)[7]
- Software Heritage IDs[8]

Supplier-generated component names should, if possible, also follow one of these standards.

NIST has discussed the idea of phasing out CPE in favor of SWID for use in the NVD.

indicated that they intend to phase out use of CPE in the NVD in favor of SWID tags. Therefore, it would be recommended to consider the schema above rather than leverage CPE.

Commented [6]: check with NVD

Commented [7]: source?

Commented [8]: hinted? we have no public evidence?

Commented [9]: rewrite around this?

Supplier Identification

Identifying suppliers is a similar problem to identifying software components. While there may be orders of magnitude fewer suppliers than components, there are multiple ways to identify suppliers. Multi-national organizations may use different legal names in different countries, marketing brands may differ from legal or official organization names, and mergers, acquisitions, and other lifecycle changes impact supplier identification. The Japan Exchange

[6] Note that SWID tags are used here in their role as a software identifier, not their broader use as a standard to convey the SBOM dependency graph data. See more here: https://csrc.nist.gov/projects/software-identification-swid/guidelines (update with more NIST guidance when available)

[7] https://github.com/package-url/purl-spec

[8] https://docs.softwareheritage.org/devel/swh-model/persistent-identifiers.html

Group,[9] D-U-N-S Numbers,[10] and IANA Private Enterprise Numbers[11] are just three existing supplier namespaces.

In developing a highly-scalable SBOM model, dependency on any centralized source of data or administration raises concerns about resilience, cost (both to operate and to suppliers), and organizational (even political) bias. Despite these issues, a “supplier registrar” model relying on a single global namespace for *suppliers* (not *components*) may provide a practical way forward. Conceptually, if supplier names are sufficiently globally unique, suppliers would have considerable discretion in how to identify components within their namespaces. Such an SBOM model would delegate component identity details to suppliers, who are likely the least cost avoiders. The model would not need to thoroughly solve global component identification, or in other words, the solution is to delegate to suppliers.

The details of such a supplier registrar model are beyond the scope of this paper, and further discussion is needed to assess the tradeoffs of such a model. Some key considerations include:

- SBOM model design: Supplier identity is globally unique, suppliers have significant autonomy in component identification
- Cost to suppliers to obtain and maintain registration
- Supplier lifecycle, including merger, acquisition, and closure
- Support for a wide range of suppliers, including part-time, individual developers
- Direct cost to operate the registrar
- Resilience and longevity of the registrar, including distributed or decentralized mechanisms
- Organizational and political bias in the operation of the registrar

Pending further work, a supplier registrar model may be a significant part of the solution to global component identification, particularly if existing supplier identification systems can be leveraged.

Conclusion

Phil Karlton, a renowned programmer of Netscape during early development of Internet based software, has said “There are only two hard things in Computer Science: cache invalidation and naming things.” After many years, software identity continues to be a difficult problem with many options and few that provide global software identity in reliable ways. If you are currently using one of the naming conventions identified earlier (like JLS specification or NPM), the current recommendation is for software vendors to continue using this identity as your approach to identify software. This provides some inherent capabilities that SBOM can take advantage of in

[9] https://www.jpx.co.jp/english/

[10] https://www.dnb.com/duns-number.html

[11] https://www.iana.org/assignments/enterprise-numbers/enterprise-numbers

resolving the software identity problem. As an alternative vendors are encouraged to explore the software identification standards (such as PURL) that are actively addressing these issues. Finally, if none of these options are viable for your organization, it is recommended that you explore our guidelines in the Supplier Identification section to identify the organization or the entity producing the software. This gives vendors flexibility in choosing a hierarchical namespace with their organization identifier at the top of this hierarchy, this giving the organization greater flexibility in picking names for their software with a greater amount of flexibility. Suppliers who choose this method can thus be prepared to adopt a prevailing software naming standard that will emerge as a winner in the future.