Response on SBOM Requirements

- Frederick Kautz, Author

- The author would like to thank Tory Clasen for additional comments and review.

It is first essential to rationalize what we are trying to represent. For this discussion, we suggest the following data model be adopted:

1. The contents of the artifact are data.
2. For identity, we recommend using a definition close to the mathematical definition. The content of an artifact (e.g., array of bytes) is the primary identity. A secondary identity may be derived from the contents, such as a hash. However, it is essential to note that hashes may have an infinite number of artifacts it may represent. In practice, we may treat a sufficiently strong hash as a unique identifier.
3. Metadata is any related information that is not explicitly in the contents of the artifact.
   a. Metadata may include but is not limited to: location information such as filenames, directories, create/modification/access timestamps, git commit trees, tooling used to generate, provenance/chain of custody, or relationship to other artifacts such as source code.
   b. There may be multiple SBOMs for identical pieces of data. If a single byte array's supplier is different, this is metadata about the SBOM rather than the data itself.
   c. metadata may vary based on time, space, location, or other factors.
4. The location of an artifact is independent of its identity. E.g., if an exploit named exploit.exe is renamed to printspooler.exe, it is still an exploit despite living in a different location.
5. An artifact may be addressable in numerous, independent ways. Examples include file name, hash when accessed in a content-addressable storage system, object name, URL, or primary key in a database.

Metadata is data about data. Explicitly describing metadata as data about data is crucial because this allows for metadata to be composable. Composability allows for richer and deeper insights, e.g., the facts of data. Signatures derived from Sigstore, PGP, or PKI would benefit significantly from composability.

RFC from NTIA

> 1. Are the elements described above, including data fields, operational considerations, and support for automation, sufficient? What other elements should be considered and why? (Supplier name, Component name, Version of the component, Cryptograph hash of the component • Any other unique identifier, Dependency relationship, Author of the SBOM data)

The information listed above is insufficient to provide an inventory. It solves for neither source nor provenance. Source code is often copied from BSD licensed projects (or equivalent) without attribution. A CVE reported in the upstream project would not percolate to projects which have copied the source unless the CVE is tagged to the contents instead of the component's name and version.

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

Ideally, the SBOM should be informed with the information provided by the compiler or linker, where the compiler produces a cryptographically secure artifact tree describing the inputs and outputs of the compilation process.

> 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. The challenge is not the lack of standards, but multiple standards and practices in different communities.

It is essential to recognize that the current SBOM proposals generally do not provide "Software Identity." Instead, they represent a claim by a software supplier. The controls are administrative rather than technological. Trust in the current SBOM formats is based on faith that the information from the supplier is correct and complete. Cryptographic hashes of the content help but are insufficient until the compiler process automates describing the relationships. The process and tooling must also be described with repeatable results.

> > 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.

Ideally, the tooling will be sufficiently flexible to handle future use cases to be developed.

> 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.
> 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 strongly recommend using signed GitBOMs to validate the source of SBOM data and transparency ledgers such as SigStore for establishing the integrity of both data and its associated metadata, including SBOMs.

> 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?

The SBOM does not need to be co-located with the operational system. Instead, we recommend providing a URL where the metadata may be protected separately from the system under attack. Furthermore, cryptographically secure Bill of Receipts systems such as GitBOM may provide a layer of indirection where metadata may be made available based on classification and role of the accessor with validation that the data and metadata have not been tampered with.

> 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 14028.13 How can SBOM data be integrated with this additional data in a modular fashion?

The Linux Foundation and Cloud Native Computing Foundation have several efforts which are complementary to the SBOM initiative. These include:

- SPDX - A SBOM format
- SigStore - Transparency ledger for signing artifacts which may include but not limited to released software and their associated SBOMs.
- In-Toto - A framework used to attest the steps of a software supply chain with cryptographically signed claims that the process was followed.
- SPIFFE - A specification for providing cryptographically secure identities to workloads.

When paired with GitBOM, the above four projects may be combined to attest the end-to-end build artifacts, related metadata, and build process in a modern CI/CD process.

> 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.

Like the data discussion above, we must separate the SBOM's identity from the SBOM's location.

> 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.

Depth may be solved by separating the data from metadata cleanly. Separation of data and metadata may be accomplished through the use of receipts. E.g., in GitBOM, receipts point only to the relevant data's hash and the receipt of its children. Data is stored in a Merkle Tree, which is a type of Content Addressable Storage.

```
blob [hash of object] bom [hash of child gitbom]
```

The use of receipts with cryptographic hashes allows for metadata to be shared only when policy permits while simultaneously maintaining the integrity of the associated children.

Further work may include "Zero-Knowledge" proofs may allow for proving claims without leaking further information. E.g. the existence of a hash may be proven without revealing the hash itself. A working example in the wild is zk-SNARKs.

> 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.

This use case alludes to the initial premise of this document. In short, we are at a crossroads. As an industry, we will either:

- Continue the status quo of tracking CVEs by the (vendor, product, version) tuple with marginal incremental improvements on security through slightly more linkage.

- Or we will begin to tag CVEs to the content where the CVE originated and build automation to discover where that content is consumed. Best case scenario, developers are notified at build time or in their Integrated Development Environment (IDE) of a known vulnerability.

The latent copying of software without attribution limits the efficacy of the status quo. We must identify vulnerabilities based upon the source code content that the CVE originated and which high-level applications contain the vulnerability.

> 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”),14 but other solutions may exist.

A quick note on this topic, it is important to determine which environments a vulnerability is active in. A vulnerability may be mitigated or cauterized by environmental factors such as operating system, configuration, or system libraries with zero code changes.

> 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 fulfilled through alternate means? What accommodations and alternate approaches can deliver benefits while allowing for flexibility?

The most important takeaway is this: Without proper tooling support, these approaches will all be challenging to adopt in any environment, regardless of open source/proprietary status. Compilers must support producing output that is consumable by an SBOM and cryptographically verifiable. SBOM developer communities must produce reference implementations that are much more mature than the status quo.

References (in alphabetical order):

- GitBOM: https://hackmd.io/aZ7czCDvRI2atAxhtYecrA?both (https://hackmd.io/aZ7czCDvRI2atAxhtYecrA?both) and https://hackmd.io/ExsL_iGvSp6pbFmFSxr4Vg?view (https://hackmd.io/ExsL_iGvSp6pbFmFSxr4Vg?view)
- In-toto: https://in-toto.io (https://in-toto.io)
- SDPX: https://spdx.dev (https://spdx.dev)
- SPIFFE support for In-Toto: https://github.com/boxboat/in-toto-golang (https://github.com/boxboat/in-toto-golang)
- SPIFFE: https://spiffe.io (https://spiffe.io)
- SigStore: https://sigstore.dev (https://sigstore.dev)
- zk-SNARKs: https://z.cash/technology/zksnarks/ (https://z.cash/technology/zksnarks/)