Potential Elements for an SBOM

NTIA proposes a definition of the ''minimum elements'' of an SBOM that builds on three broad, inter-related areas: Data fields, Operational considerations, and Support for automation. Focusing on these three elements will enable an evolving approach to software transparency, and serve to ensure that subsequent efforts will incorporate more detail or technical advances. The information below is preliminary, and the ultimate list published by NTIA will be revised based on public input.

Data fields. To understand the third- party components that make up software, certain data about each of those components should be tracked. This ''baseline component information'' includes:

- Supplier name
- Component name
- Version of the component
- Cryptograph hash of the component
- Any other unique identifier
- Dependency relationship
- Author of the SBOM data

Operational considerations. SBOM is more than a set of data fields. Elements of SBOM include a set of operational and business decisions and actions that establish the practice of requesting, generating, sharing, and consuming SBOMs. This includes:

- Frequency. Operational considerations touch on when and where the SBOM data is generated and tracked. SBOM data could be created and stored in the repository of the source. For built software, it can be tracked and assembled at the time of build. A new build or an update to the underlying source should, in turn, create a new SBOM.

- Depth. The ideal SBOM should track dependencies, dependencies of those dependencies, and so on down to the complete graph of the assembled software. Complete depth may not always be feasible, especially as SBOM practices are still novel in some communities. When an SBOM cannot convey the full set of transitive dependencies, it should explicitly acknowledge the ''known unknowns,'' so that the SBOM consumer can easily determine the difference between a component with no further dependencies and a component with unknown or partial dependencies.

- Delivery. SBOMs should be available in a timely fashion to those who need them and have proper access permissions and roles in place. Sharing SBOM data down the supply chain can be thought of as comprising two parts: How the existence and availability of the SBOM is made known (advertisement or discovery) and how the SBOM is retrieved by or transmitted to those who have the appropriate permissions (access).[8] Similar to other areas of software assurance, there will not be a one-size-fits-all approach. Anyone offering SBOMs must have some mechanism to deliver them, but this can ride on existing mechanisms. SBOM delivery can reflect the nature of the software as well: Executables that live on endpoints can store the SBOM data on disk with the compiled code, whereas embedded systems or online services can have pointers to SBOM data stored online.

Automation support. A key element for SBOM to scale across the software ecosystem, particularly across organizational boundaries, is support for automation, including automatic generation and machine-readability. As the Executive Order notes, SBOMs should be machine-readable and should allow ''for greater benefits through automation and tool integration.'' Manual entry or distribution with spreadsheets does not scale, especially across organizations.

The SBOM community has identified three existing data standards (formats) that can convey the data fields and be used to support the operations described above: SPDX,[9] CycloneDX,[10] and SWID tags.[11] Experts in these formats have mapped between them to create interoperability for the baseline described above. Because these formats already are subject to public input and translation tools exist, they serve as logical starting points for sharing basic data.[12]

In addition to the three SBOM formats, the need for automation defines how some of the fields might be implemented better. For instance, machine-scale detection of vulnerabilities requires mapping component identity fields to existing vulnerability databases.

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

A SBoM must be the authoritative record of when who did what to Software.

From the outset, an SBoM must be considered a multi-party asset. Changes to it can occur in many places and it is inefficient to keep asking, checking, or sending release notes to all parties. This does not imply all information should be public but a system that allows authorized participants to send and receive such information must offer high levels of assurance and trust.

All stakeholders need a system that helps parties reach consensus on the state of a SBoM, where no single actor has full control and an ability to corrupt the record or shred evidence. The record entries need integrity, provenance, availability, non-repudiation, and immutability. All these principles of transparency and accountability build trust.

Software is only secure until it is not. Labelling software as "safe and trustworthy" will require new continuously delivered services throughout the operational lifespan of code that must also be noted in the SBoM.

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

Zero Trust - The call toward Zero Trust Architecture can gain a boost from SBoMs. A fully-machine readable, automated SBoM, analyzed against automatable treat intelligence can provide vital meta-information for instantaneous Zero-trust authorization decisions. A SBoM plays a key role in the gathering the right data in the right place for a decision at the right time - real-time compliance based on real-time context.

Continuous Compliance - Demonstrating real time compliance to regulations on a continuous basis will fundamentally change the "security theatre" of annual audits. Users would be no longer forced to choose between out-of-date compliant configurations or up-to-date secure ones and regulators could be well positioned to play an active participation in dealing with systemic risks.

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.**

Sustainability of software - The SBoM can be a means by which suppliers gain operational insight of versions in use, versions to target long-term support and versions to retire. This can aid lifecycle resource planning, contractual decisions, business modelling and performance management to industry best practice.

Artificial Intelligence - Future software concerns the emergent properties of networks, especially when using machine learning. Trustworthy Artificial Intelligence will need more than a SBoM to prove trust in its operational state:

- What AI Model is loaded?
- What was the quality of data sources that trained/optimized the AI?
- What records are kept regarding decisions made?

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.

A multi-standard approach must not force one opinionated view of naming convention – tokenizing the SBoM can enable coexistence of naming conventions.

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.

The risk of using data produced, extracted, transformed, or loaded by software on someone else's computer is as present as running it on your own – whether that's software running on a cloud service or in constrained connected things on premises: industrial control systems or connected sensors may not afford the end user full control of the whole technology stack. Just because you bought it, doesn't mean you own everything in it!
This is all about data – and full traceability of everyone's SBoM and intellectual property that could impact that data – would bring full transparency and the highest level of trust.

Understanding that context is vital in digital systems, especially when it comes to a more complete Digital Bill of Materials which should include data provenance, chain-of-trust and AI data models.

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.
SBoM data sharing mechanisms should allow for vendors to notify expected end-of-life events and for end users to record what information they know of unsupported legacy code.

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.
SBoM users need authenticated provenance of sources along with integrity guarantees of the data itself. A system of record that does not allow data to be changed, only appended can help in delivering provenance, integrity, non-repudiation and immutability to foster trust between parties. Identity verification of all actors involved should be strongly encouraged. A consistent record amongst authorized participants must not be and tamper evident to spot any corruption by participants.

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 old saying "trust but verify" can and should be reversed with machines. Verify then trust – the machines won't be offended! Every stage of the software build cycle should include verification of the previous step(s).

Artefacts of a secure software development process can be included in a SboM such as.

- who signed off on what?
- who committed what code?
- who reviewed and merged the code?
- What tools complied the code in what order?
- Who authorized release?
- What protects the code signing keys and who has access to them?

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?

SBOM must be extensible for connections to many more systems and human actors. Code compilers, static code analysis, vulnerability assessments, secure development practices, audits, security evaluations, should be considered.

High Assurance – trust in software is underpinned by an underlying Hardware Bill-of-Materials and Trust Bill-of-Materials. Private and secret key handling is the basis of all cryptographic security. On that basis it's also important to consider:

- How were private keys or secrets generated and who had potential access to them while created?
- Where are private keys are stored and what is their level of attack resistance?
- What security-oriented software handles the keys in operation – A secure enclave, trusted execution environment or in plain sight within an app that a compromised OS could see?

Just because there's a certificate, does not mean all is secure. Just because you have a TPM, does not mean all is forever secure either https://en.wikipedia.org/wiki/ROCA_vulnerability

And just because you have a SBoM, does not mean all is secure. Hardware protections, cryptographic implementations and operational data models should be considered in an extensible future-proof SBoM framework.

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.

Governance mechanisms should allow any network to easily invite supply chain participants to share trustworthy data.

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.

Sharing everything with everyone may be impractical and a SBoM sharing mechanism must allow for private communication to authorized users.

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.

Things are only secure until they are not. Tracing the timestamps of who knew what when can help establish whether known vulnerabilities are present in certain versions of software.

Responsible disclosure can pressure vendors to fix before a publication deadline. A SboM data sharing mechanism could notify users of a responsibly disclosed vulnerability without sharing full details until the disclosure deadline. This would let users know that vendors are aware of an issue and creating a patch, and then prepare a maintenance window as soon as the fix is available to minimize time of exposure.

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.

Decisions to proceed with full knowledge of a vulnerability, should be recorded in data sharing systems that notify federal customers of assessed and accepted risks.

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?

Easy to access, distributed ledger based technologies are key to building the multi-stakeholder trust and transparency required in xBoM implementations.