The Minimum Elements
For a Software Bill of Materials (SBOM)

Pursuant to
Executive Order 14028
on Improving the Nation's Cybersecurity

The United States Department of Commerce

July 12, 2021

Table of Contents

I. Executive Summary

The Executive Order (14028) on Improving the Nation's Cybersecurity directs the Department of Commerce, in coordination with the National Telecommunications and Information Administration (NTIA), to publish the "minimum elements" for a Software Bill of Materials (SBOM). An SBOM is a formal record containing the details and supply chain relationships of various components used in building software. In addition to establishing these minimum elements, this report defines the scope of how to think about minimum elements, describes SBOM use cases for greater transparency in the software supply chain, and lays out options for future evolution.

An SBOM provides those who produce, purchase, and operate software with information that enhances their understanding of the supply chain, which enables multiple benefits, most notably the potential to track known and newly emerged vulnerabilities and risks. SBOM will not solve all software security problems, but will form a foundational data layer on which further security tools, practices, and assurances can be built. The minimum elements as defined in this document are the essential pieces that support basic SBOM functionality and will serve as the foundation for an evolving approach to software transparency. These minimum elements comprise three broad, interrelated areas.

Minimum Elements	
Data Fields	Document baseline information about each component that should be tracked: Supplier, Component Name, Version of the Component, Other Unique Identifiers, Dependency Relationship, Author of SBOM Data, and Timestamp.
Automation Support	Support automation, including via automatic generation and machine-readability to allow for scaling across the software ecosystem. Data formats used to generate and consume SBOMs include SPDX, CycloneDX, and SWID tags.
Practices and Processes	Define the operations of SBOM requests, generation and use including: Frequency, Depth, Known Unknowns, Distribution and Delivery, Access Control, and Accommodation of Mistakes.

This document identifies minimum elements that will enable basic use cases, such as management of vulnerabilities, software inventory, and licenses. It also looks forward, beginning a conversation on recommended SBOM features and advances beyond the minimum elements that may be seen as priorities for further work. This includes key security features such as SBOM integrity, as well as tracking more detailed supply chain data. As additional SBOM elements become feasible, tested, and built into tools, they will enable broader use cases. Some of these aspirational elements are being implemented today or have already shown great potential.

The Administration has identified SBOM as a priority to drive software assurance and supply chain risk management, and starting today is better than waiting for perfection. Following the publication of this report, next steps include the development of guidance for providing an

SBOM to software purchasers, as directed by the Executive Order, as well as continued collaboration and public-private partnerships to refine and operationalize SBOM work.

II. Background

On May 12, 2021, the President issued Executive Order (EO) 14028 on Improving the Nation's Cybersecurity.[1] The order focuses on modernizing cybersecurity defenses by: protecting Federal networks, improving information sharing between the U.S. Government and the private sector on cyber issues, and strengthening the United States' ability to respond to incidents when they occur. Among numerous other taskings, it directed the Secretary of Commerce, in coordination with the Assistant Secretary for Communications and Information and the Administrator of the National Telecommunications and Information Administration (NTIA), to publish minimum elements for a software bill of materials (SBOM) within 60 days of the order.

NTIA has been focusing on this issue since 2018, when it convened an open, transparent multi-stakeholder process on software component transparency. NTIA acted as a convener and a neutral facilitator, bringing together experts from around the world that represented multiple sectors and perspectives in the software ecosystem. The resulting resources were drafted by stakeholders, with frequent opportunities for the community to weigh in as the deliverables took form. The documentation from that process provided valuable insight and background material for this activity to propose the minimum elements of an SBOM as directed by EO 14028.

The Case for Transparency

The Administration notes in the EO, "the trust we place in our digital infrastructure should be proportional to how trustworthy and transparent that infrastructure is."[2] In the modern world, software systems involve complex, dynamic — and, too often, obscure — supply chains. Bringing transparency to the components and connections within and across supply chains is important to discovering and addressing the weak links in those chains. SBOMs are a critical step toward securing the software supply chain. Without them, a lack of transparency into the contributors, composition, and functionality of these systems contributes substantially to cybersecurity risks and increases costs of development, procurement, and maintenance.

Transparency is best achieved using an understandable model supported by industry. An SBOM model achieves this systematic sharing by tracking component metadata, enabling mapping to other sources of information, and tying the metadata to software as it moves down the supply chain and is deployed. To scale this model globally, it is necessary to address the problem of universally identifying and defining certain aspects of software components to allow the data to be effectively and efficiently consumed by downstream users.[3]

[1] Exec. Order No. 14,028, 86 Fed. Reg. 26,633 (May 12, 2021).

[2] *Id.* § 1.

[3] Framing Working Grp., Nat'l Telecomm. & Info. Admin., *Framing Software Component Transparency* 4 (2019), https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf.

Identification of software components is central to SBOM, providing visibility and awareness. SBOM data can be used for specific purposes, from simple (*e.g.* mapping to vulnerability database) to complex (*e.g.* ongoing monitoring of an included OSS package for specifically defined threats by correlating and analyzing multiple data sources). In both cases, external data may still be needed. The SBOM is the necessary glue to allow the relevant external data to be mapped to the software products in question.

An SBOM is useful to those who produce, purchase, and operate software. It enables an understanding of the software ecosystem and provides benefits across multiple use cases and users.[4] Among these are the use of SBOMs for inventory, vulnerability, and license management by producers and operators, and risk evaluation (license and vulnerability analysis) by purchasers.

SBOMs offer advantages to producers such as ensuring that components are up to date and allowing a quick response to new vulnerabilities. SBOMs also offer benefits beyond security such assupporting greater efficiency and effectiveness through visibility, which in turn enables prioritization and better management. For example, they assist the producer with knowing and complying with license obligations.

In the use case of vulnerability management, SBOM data helps producers and operators more quickly and accurately assess the risk associated with a newly uncovered vulnerability by providing transparency across dependencies within the software ecosystem. As such, it improves both vulnerability identification and the speed of response.

Understanding the software supply chain, obtaining comprehensive SBOM data across software components, and using it to identify and analyze known vulnerabilities and potential mitigations are crucial in managing risk. This can only be realized with machine-readable SBOMs supporting automation and tool integration, and the ability for applications to query and process this data.

III. Scope

This document establishes the “minimum elements” of an SBOM. These minimum elements will establish the baseline technology and practices for the provisioning of SBOMs and are deemed necessary to achieve the goals expressed in Executive Order 14028. This document also reviews

[4] For a more complete discussion on the use cases of SBOM across the software lifecycle, see Multistakeholder Process on Software Component Transparency Use Cases & State of Prac. Working Grp., Nat’l Telecomm. & Info. Admin., *Roles and Benefits for SBOM Across the Supply Chain* (2019), https://www.ntia.gov/files/ntia/publications/ntia_sbom_use_cases_roles_benefits-nov2019.pdf.

how these basics can be expanded upon and offers some guidance on the tension between having a predictable SBOM format and the need for flexibility, depending on the technology in question and the needs of consumers.

SBOMs alone will not address the multitude of software supply chain and software assurance concerns faced by the ecosystem today.[5] It is a necessary cliché to acknowledge that there are no cybersecurity panaceas, and SBOM is no exception. As noted above, SBOMs can facilitate better and faster responses to known vulnerabilities. The number of known vulnerabilities for a given piece of software is a function of its install base, the research community, and the supplier's disclosure process and product security team. More disclosed vulnerabilities may mean the software is less risky to use, since this means that researchers are paying attention and the supplier is managing the disclosure process.

SBOM is a starting point that builds on identified vulnerabilities. The minimum elements that are deemed feasible in today's environment do not capture the full range of metadata around software source, processing, and use that is likely to emerge from modern software processes. Some of this data will be incorporated into future extensions of SBOM data. At the same time, SBOMs will not be the sole resource or mechanism for supply chain security or software assurance. Other data is quite valuable for a range of use cases, but should be considered as separate and complementary to SBOM. Rather than treat the SBOM as a single model for all assurance and software supply chain data, a linkable, modular approach is encouraged to maximize the potential for flexibility and adoption. Linkability enables SBOM data to be easily mapped to other important supply chain data, while a modular architecture supports extensibility for more use cases as software supply chain transparency and management data and tools mature.

Certain key points of the software supply chain discussion are out of scope of this report, including the question of regulatory and procurement requirements. The minimum elements should not be interpreted to create new federal requirements. The potential benefits of centralizing or pooling SBOM data for operations, threat intelligence or research has not been addressed. Lastly, the "software" in the "Software Bill of Materials" naturally limits considerations of hardware. While software embedded in hardware and devices is certainly in scope, the key supply chain and security issue pertaining to hardware is distinct and complex enough to deserve its own treatment.

Finally, nothing in this document should be seen to limit SBOM use or constrain the innovation and exploration occurring across the software ecosystem today. These minimum elements are the

[5] For an overview of the range of attacks on the software supply chain, see Dr. Trey Herr et al., *Breaking Trust: Shades of Crisis Across an Insecure Software Supply Chain*, Atlantic Council (Jul. 26, 2020), https://www.atlanticcouncil.org/in-depth-research-reports/report/breaking-trust-shades-of-crisis-across-an-insecure-software-supply-chain/.

starting point. Broadly speaking, this document represents a key, initial step in the SBOM process that will advance and mature over time.

IV. Minimum Elements

The minimum constituent parts of an overall SBOM – referred to as elements – are three broad, inter-related areas. These elements will enable an evolving approach to software transparency, capturing both the technology and the functional operation. Subsequent efforts will certainly incorporate more detail or technical advances. As noted above, these are the minimum at this point; organizations and agencies may ask for more, and the capabilities for transparency in the software supply chain may improve and evolve over time. (See "Future of SBOM" below)

These three categories of elements are:

- Data Fields
- Automation Support
- Practices and Processes

A piece of software can be represented as a hierarchical tree, made up of components that can, in turn, have subcomponents, and so on. Components are often "third party," from another source, but might also be "first party," that is, from the same supplier but able to be uniquely identified as a freestanding, trackable unit of software. Each component should have its own SBOM listing their components, building the hierarchical tree. The data fields apply to each component, which are, in turn, encoded with tools and formats for automation support following the defined practices and processes.

Data Fields

The core of an SBOM is a consistent, uniform structure that captures and presents information used to understand the components that make up software. Data fields contain baseline information about each component that should be tracked and maintained. The goal of these fields is to enable sufficient identification of these components to track them across the software supply chain and map them to other beneficial sources of data, such as vulnerability databases or license databases. This baseline component information includes:

Data Field	Description
Supplier Name	The name of an entity that creates, defines, and identifies components.
Component Name	Designation assigned to a unit of software defined by the original supplier.
Version of the Component	Identifier used by the supplier to specify a change in software from a previously identified version.
Other Unique Identifiers	Other identifiers that are used to identify a component, or serve as a look-up key for relevant databases.
Dependency Relationship	Characterizing the relationship that an upstream component X is included in software Y.
Author of SBOM Data	The name of the entity that creates the SBOM data for this component.
Timestamp	Record of the date and time of the SBOM data assembly.

The majority of these fields assist with the identification of the component to expressly enable mapping to other sources of data. *Supplier* refers to the originator or manufacturer of the software component. *Component Name* is determined by the supplier. The capability to note multiple names or aliases for both supplier and component name should be supported if possible.

The challenges created by the lack of a single, well-understood and widely used name space for software are well documented.[6] The best practice is to use an existing identifier when possible. When none exists, use an existing component identification system. *Supplier name* and *component name* are human-readable strings to support identification, although some features of the software ecosystem such as corporate mergers and open source forking will make ubiquitous and permanent solutions on this basis unlikely.

Complexity around *versions* of software is another example of how the diversity in the software ecosystem requires building some flexibility into component identification approaches. Different types of software or suppliers of software track versions and distributions differently. Resolving

[6] A single, centralized model that can cover the vast, diverse, and rapidly growing software ecosystem presents very real challenges, notably scaling. A decentralized model where suppliers identify and manage their own software namespace seems the optimal path forward. For more information, see *Framing Software Component Transparency*, *supra* note 3, at page 24.

this is outside the scope of the initial SBOM discussion.[7] For these minimum elements, the *version* is that offered by the supplier since that party has the ultimate responsibility for tracking and maintaining the software in question, similar to the component name. The desired function of a version string is to identify a specific code delivery. While there are versioning best practices (*e.g.* semantic versioning[8]), they are by no means ubiquitous today.

Other unique identifiers support automated efforts to map data across data uses and ecosystems and can reinforce certainty in instances of uncertainty. Examples of commonly used unique identifiers are Common Platform Enumeration (CPE),[9] Software Identification (SWID) tags,[10] and Package Uniform Resource Locators (PURL).[11] These other identifiers may not be available for every piece of software, but should be used if they exist.

Dependency relationship reflects the directional aspect of software inclusion, and it enables the representation of transitivity from a piece of software to its component and a potential sub-component. Lastly, the SBOM-specific metadata help with the tracking of the SBOM itself. *Author* reflects the source of the metadata, which could come from the creator of the software being described in the SBOM, the upstream component supplier, or some third party analysis tool. Note that this is not the author of the software itself, just the source of the descriptive data. *Timestamp* records when the data is assembled -- the point of the SBOM creation. These further support the origin of the data, and help identify updated versions of the SBOM. These data fields provide context to the SBOM data source, and can potentially be used to make trust determinations.

Automation Support

Support for automation, including automatic generation and machine-readability, allows the ability to scale across the software ecosystem, particularly across organizational boundaries. Taking advantage of SBOM data will require tooling, which necessitates predictable implementation and data formats. For example, some agencies may want to integrate this capability into their existing vulnerability management practices; others might desire real-time

[7] As more visibility emerges through SBOM use and consumption, we can expect further discussions, and potentially greater convergence of diverse models, approaches, and schemas.

[8] Semantic Versioning 2.0.0, https://semver.org/ (last visited July 1, 2021).

[9] *See* Framing Working Group, Nat'l Telecomms. & Info. Admin., *Software Identification Challenges and Guidance* (2021), https://www.ntia.gov/files/ntia/publications/ntia_sbom_software_identity-2021mar30.pdf; *Official Common Platform Enumeration (CPE) Dictionary*, Nat'l Inst. Standards & Tech., https://nvd.nist.gov/products/cpe (last visited July 2, 2021).

[10] *See Software Identification Challenges and Guidance*, *supra* note 9; *ISO/IET 19770-2:2015 Information Technology–IT Asset Management—Part 2: Software Identification Tag*, Int'l Standards Org., https://www.iso.org/standard/65666.html (last visited July 2, 2021).

[11] *See Software Identification Challenges and Guidance*, *supra* note 9; *Package-url/purl-spec*, GitHub, https://github.com/package-url/purl-spec (last visited July 2, 2021).

auditing of compliance against security policies. Automation will be key for both, which in turn requires common, machine-readable data formats.

While a single standard may offer simplicity and efficiency, multiple data formats exist in the ecosystem and are being used to generate and consume SBOMs. These specifications have been developed through open processes, with international participation. In addition to being machine-readable, these data formats are also human-readable, better supporting trouble-shooting and innovation. More importantly, these standards have been deemed interoperable for the core data fields and use common data syntax representations.

The data formats that are being used to generate and consume SBOMs are:

- Software Package Data eXchange (SPDX)[12]
- CycloneDX[13]
- Software Identification (SWID) tags[14]

The SBOM must be conveyed across organizational boundaries in one of these interoperable formats.[15] If a new specification should emerge that is compatible with the other data formats, then it should be included for automation support in the context of minimum elements for SBOM. Similarly, if a broad-based determination is made that a data format is no longer cross compatible, or is not under active maintenance and supporting the SBOM use cases, that data format should be removed from the automation requirement as part of the SBOM minimum elements. Standards utilizing proprietary data formats should not be included. This accomplishes the goal of building on existing tools for ease of adoption, supporting future evolution, and extensibility.

Practices and Processes

An SBOM is more than a structured set of data; to integrate it into the operations of the secure development life cycle an organization should follow certain practices and processes that focus on the mechanics of SBOM use. A number of elements should be explicitly addressed in any policy, contract, or arrangement to ask for or provide SBOMs. Some of these (*e.g.*, frequency) have straightforward requirements. In other cases (*e.g.*, access), multiple practices exist and more are being developed, so the minimum element is a requirement that some arrangement is specified.

[12] SPDX, https://spdx.dev/ (last visited May 18, 2021).

[13] CycloneDX, https://cyclonedx.org/ (last visited May 18, 2021).

[14] *See* David Waltermire et al., *Guidelines for the Creation of Interoperable Software Identification (SWID) Tags* (2016) (Nat'l Inst. of Standards & Tech. Internal Rep. 8060), http://dx.doi.org/10.6028/NIST.IR.8060 (SWID tags are defined by ISO/IEC 19770–2:2015).

[15] Some commenters have emphasized the value of a single format. However, these data formats are actively used and supported today, and many SBOM experts have noted that the government may not be in the best position to pick the winner among competing standards. The SBOM community has emphasized maintaining interoperability and automated translation tools, noting that computers are good at format conversions.

Frequency. If the software component is updated with a new build or release, a new SBOM must be created to reflect the new version of the software. This includes software builds to integrate an updated component or dependency. Similarly, if the supplier learns new details about the underlying components or wishes to correct an error in the existing SBOM data, the supplier should issue a new, revised SBOM.

Depth. An SBOM should contain all primary (top level) components, with all their transitive dependencies listed. At a minimum, all top-level dependencies must be listed with enough detail to seek out the transitive dependencies recursively.

Going further into the graph will provide more information. As organizations begin SBOM, depth beyond the primary components may not be easily available due to existing requirements with subcomponent suppliers. Eventual adoption of SBOM processes will enable access to additional depth through deeper levels of transparency at the subcomponent level. It should be noted that some use cases require complete or mostly complete graphs, such as the ability to "prove the negative" that a given component is *not* on an organization's network.

An SBOM consumer can specify depth by number of transitive steps. Alternatively, they could specify depth in operational terms. This could include software attributes, such as "all non-open source software," or all components of a certain function or complexity. Organizations can also incentivize greater reporting depth and completeness by offering different requirements on reporting and remediation of vulnerabilities in components enumerated in the SBOM versus those that are not. Such specifications are outside the scope of these minimum elements.

Known Unknowns. For instances in which the full dependency graph is not enumerated in the SBOM, the SBOM author must explicitly identify "known unknowns." That is, the dependency data draws a clear distinction between a component that has no further dependencies, and a component for which the presence of dependencies is unknown and incomplete. This must be integrated into the automated data. To avoid erroneous assumptions, the default interpretation of the data should be that the data is incomplete; the author of the data should affirmatively state when the direct dependencies of a component have been fully enumerated, or when a component has no further dependencies. Today, this is implemented in the dependency relationship data field.

Distribution and Delivery. SBOMs should be available in a timely fashion to those who need them and must have appropriate access permissions and roles in place. The SBOM data can accompany each instance of the software, or merely be accessible and directly mappable to the specific version of the software in question (*e.g.* through a version-specific URL). Sharing SBOM data down the software 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*). Similar to other areas of software assurance, there will not be a one-size-fits-all approach. Anyone offering SBOMs must have some means to make them available and support ingestion, 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.

Access Control. Many suppliers, including open source maintainers and those with widely available software, may feel their interests are best served by making SBOM data public. Other organizations, especially at first, may wish to keep this data confidential, and limit access to specific customers or users. If access control is desired, the terms must be specified, including specific allowances and accommodations for integrating SBOM data into the user's security tools. Such specification can be determined through licensing, contracts, or other existing mechanism used to circumscribe the use and rights around the software itself. Given the variation in software licensing and contracts, the nature of this specification is outside the scope of this document.

Accommodation of Mistakes. A final practice area, accommodation of mistakes, should be built into the initial implementation phase of SBOM, allowing for omissions and errors. As many commentators have observed, while internal management of supply chain data may be a best practice, it is still evolving. The Administration has identified SBOM as a priority to drive software assurance and supply chain risk management, and starting today is better than waiting for perfection. In light of the absence of perfection, consumers of SBOMs should be explicitly tolerant of the occasional incidental error. This will facilitate constant improvement of tools: suppliers should offer updated data as they come across issues with past SBOMs, and consumers should encourage these updates by welcoming supplements and corrections without penalty when offered in good faith. As stated above regarding frequency, when new data is known an updated SBOM should be issued. Notably, this tolerance should not apply to intentional obfuscation or willful ignorance.

V. Beyond Minimum Elements: Enabling Broader SBOM Use Cases

The above characterize the "minimum elements" that comprise SBOM creation, maintenance, and use across the software supply chain. As noted, these are the initial steps and requirements needed to support the basic use cases. There is more work to be done to expand transparency in the software supply chain and to support visibility for securing software. This section describes further additions and inclusions that can support broader SBOM use cases.

The fact that they are not part of the minimum set does not mean that these areas can be ignored; indeed, some will ultimately be critical to successful and efficient implementation of SBOM use cases. Many of these are in production today, or will be shortly with some small further refinements or testing. Organizations seeking SBOMs should feel comfortable working with their suppliers to ask for them. In many of the elements below, specific recommendations are proposed.

Recommended Data Fields

In addition to the data fields described in the minimum elements above, the following data fields are recommended for consideration, especially for efforts that are planned over several years or that require higher levels of security. In many cases and contexts these fields are well-defined and already implemented; others may require greater details for specification and clarity. As these are added to the minimum elements, any data formats that do not already include the below must add them or risk getting relegated.

Hash of the Component. When referring to a piece of software, robust identifiers are important for mapping the existence of a component to relevant sources of data, such as vulnerability data sources. A cryptographic hash would provide a foundational element to assist in this mapping, as well as helping in instances of renaming and whitelisting.

Hashes also offer confidence that a specific component was used. The consumer could compare the hash of the component with a known, trusted version. This could help verify that an "approved" version of a component was used, and is necessary to identify whether the component has been altered in unauthorized fashion. A hash is a key foundation for using SBOM to have trust in the software supply chain.

There are some situations when a hash may not be possible, or convey relatively little value. If component information was obtained from a tool that did not have direct access to the underlying component (*e.g.* a binary analysis tool), then the component author may not be able to credibly determine the exact bits used, and so be unable to generate a hash.

There are benefits of added assurance that come with a hash, but the diversity of the potential targets makes implementation somewhat complex. A file is straightforward, but an executable will have some differences compared to source packages and different hash algorithms will, of course, produce different values. A number of resolutions to this challenge exist, including providing enough detail for the consumer to replicate the hash from the original, or multiple hashes for the small number of potential implementations.

Organizations can request hashes for SBOMs today, especially those focused on high assurance use cases. It is recommended that they specify further details about how the hash should be generated. Further defining and refining best practices and specifications for hash generation and consumption should be a priority for the SBOM community.

Lifecycle Phase. The data about software components can be collected at different stages in the software lifecycle, including from the software source, at build time, or after build through a binary analysis tool. Due to unique features of each of these stages, the SBOM may have some differences depending on when and where the data was created. For example, a compiler may pull in a slightly different version of a component than what was expected from the source. For this reason, it would be helpful to have some means of easily conveying where, when, and how the SBOM data was recorded. As noted in the *Future of SBOM* section below, this may ultimately allow for the documentation of more supply chain data. In the short run, simply noting how this data was captured, (*e.g.* “source,” “build,” or “post-build”) will be helpful for consumption and data management.

Other Component Relationships. The minimum elements of SBOM are connected through a single type of relationship: dependency. That is, X is *included* in Y. This relationship is implied in the SBOM graph structure. Other types of dependency relationships can be captured, and have been implemented in some SBOM standards. One approach that can be captured today beyond direct dependencies is “derivation” or “descendancy”. This can indicate that a component is similar to some other known component, but that some changes have been made. It can be useful to track for its shared origins and content. Further suggestions on other types of dependencies are explored below.

License Information. License management was an early use case for SBOM, helping organizations with large and complex software portfolios track the licenses and terms of their diverse software components, especially for open source software. SBOMs can convey data about the licenses for each component. This data can also allow the user or purchaser to know if the software can be used as a component of another application without creating legal risk.[16]

Cloud-based Software and Software-as-a-Service

Many modern software applications are provided as a service.[17] This affords both distinctions and unique challenges with respect to SBOM data. Since the software is not running on the customer’s infrastructure or under their control, the risk management roles are different. The user is not responsible for maintenance, nor can they control any environmental factors. The responsibilities for understanding and acting on vulnerability or risk information lies with the service provider. Moreover, modern web applications often have much faster release and update cycles, making direct provisioning of SBOM data less practical.

At the same time, there are challenges to capturing the software supply chain risks in the cloud context. The service provider must not only track metadata from the software supply chain of the software they are responsible for producing, but in the infrastructure stack that supports the application, whether under the direct control of the provider or from some external service provider. Many applications also take advantage of third-party services, sending data and requests to other organizations through application programming interfaces. Capturing meaningful metadata about the full application stack and third-party services is ongoing work, but not yet standardized or sufficiently mature for cross-organization implementation.

[16] Both CycloneDX and SPDX support the expression of licenses in several ways, including a license ID on the SPDX license list, or using SPDX license expressions. *See SPDX License List*, SPDX https://spdx.org/licenses/ (May 20, 2021). SWID tags were designed, in part, to convey information around commercial licenses. *See Guidelines for the Creation of Interoperable Software Identification (SWID) Tags*, *supra* note 14, at page 1.
[17] Peter Mell & Timothy Grace, Nat’l Inst. of Standards and Tech., Special Pub. 800-145, *The NIST Definition of Cloud Computing* 2 (2011).

The NIST definition of "EO-critical software" applies to cloud-based software, but NIST recommends that the initial implementation phase focus on "on-premise software."[18] A similar approach is valuable for SBOM.

In the short run, it is recommended that cloud service providers assert that they have an internal SBOM. That SBOM must be maintained with the rough functional equivalents of the minimum elements above, although the exact format and architecture may vary based on a provider's internal system. The organization must also have the capability to act on this information and have a process to do so in a timely fashion. Over time, best practices will emerge to integrate SBOM data into third party risk management and supply chain risk management tools and processes. One use case that might be relevant for government agencies is forensic SBOM analysis: whether the cloud provider can determine whether or not a particular component was part of the deployed system at some time in the past.

SBOM Integrity and Authenticity

An SBOM consumer may be concerned about verifying the source of the SBOM data and confirming that it was not altered. In the software world, integrity and authenticity are most often supported through signatures and public key infrastructure. As SBOM practices are implemented, some existing measures for integrity and authenticity of both software and metadata can be leveraged. Some of the SBOM data formats described above can explicitly support these security measures today, while ongoing open source work is tackling the priority of signing metadata from development environments. Similarly, existing software signing infrastructure can be leveraged for tools and management of cryptographic materials, including public key infrastructure.

Those supplying and requesting SBOMs are encouraged to explore options to both sign SBOMs and verify tamper-detection. Such a mechanism should allow the signing of each component of a given piece of software and allow the user to determine whether the signature is legitimate. Integrity and authenticity are a priority for many government agencies, especially in the national security domain. Some users of SBOM data may insist on requiring digital signatures for SBOMs today.

Vulnerabilities and SBOM

The primary security use case for SBOM today is to identify known vulnerabilities and risks in the software supply chain. Some developers may choose to store vulnerability data inside the SBOM, and multiple SBOM data formats support this. There is clear value for the developer in this approach. However, SBOM data is primarily static. That is, it reflects the properties of the

[18] *Critical Software – Definition & Explanatory Material*, NIST – Info. Tech. Lab'y (Jun. 25, 2021), https://www.nist.gov/itl/executive-order-improving-nations-cybersecurity/critical-software-definition-explanatory.

specific built software at a point in time. Vulnerability data, meanwhile, is dynamic and evolves over time. Software that was not previously deemed vulnerable may "become" vulnerable as new bugs are discovered.

Vulnerability data in the SBOM cannot be assumed to be complete and up-to-date, unless very specific conditions and processes are in place. This is unlikely across organizational boundaries. SBOM data will most likely have to ultimately be linked to vulnerability data sources. (This does not, however, limit the value of providing vulnerability, software weaknesses, and risk information to the consumer of the software).

It is recommended that vulnerability data be tracked in separate data structures from the SBOM. Operations should focus on mapping and linking between the two types of data as each evolve and the technologies mature. If vulnerability data is shared across organizations, both the vulnerability data and the SBOMs can use similar models for distribution, access control, and ingestion.

Vulnerability and Exploitability in Dependencies

While software vulnerabilities are a key component of understanding risk, not all vulnerabilities put users and organizations at risk. This is especially true when dealing with transitive dependencies. Not all vulnerabilities in components create risks in the software that depends on them. Some vendor data suggests that a relatively small percentage of vulnerable components have a security impact in the environment where that software is deployed. In the SBOM context, focusing on upstream vulnerable components that have been deemed not to have an impact on the downstream software will waste time and resources, without offering immediate security benefits

Addressing this challenge requires two steps. First, the supplier must make some reliable determination that a vulnerability does not affect a specific piece of software. This could be for a range of reasons: the compiler might remove the affected code from the component, the vulnerability may not be reachable in the execution path, in-line protections exist, or a host of other reasons. These determinations are ideally already made today by product security incident response teams (PSIRTs) who track internal dependencies and risks.

The second step requires communication downstream to the next user of this SBOM data, asserting that the vulnerability does not put the organization at risk. This is straightforward, linking of a piece of software (the vulnerability in question) and the status of that vulnerability. The community refers to this as a "Vulnerability Exploitability eXchange," or VEX. The core of VEX is the communication of whether or not a given piece software is "affected" by a given vulnerability. In this case, if no action is deemed necessary, then the status is "not affected." VEX is being implemented today as a profile in the Common Security Advisory Framework,[19]

[19] OASIS Common Security Advisory Framework, http://oasis-open.github.io/csaf-documentation/ (last visited July 6, 2021).

which enables machine-readable information about whether software is affected or not affected by a vulnerability and can link to specific SBOM data. Other implementations are possible. It is recommended that tools that analyze SBOM data for the customer build in the capability to automatically incorporate VEX data.

Legacy Software and Binary Analysis

From an efficiency and utility perspective, SBOM data should be provided by the supplier. However, that is not always possible, nor the best option. In some cases, the source may not even be obtainable, with only the object code available for SBOM generation. Software that is not maintained is at greatest risk of being exploitable. Older software is at a greater risk of not being maintained. Legacy software's older code base, and its frequent use in important parts of critical infrastructure, often makes transparency more important, especially for assessing risk from known vulnerabilities. In these cases, binary analysis tools can be used to better understand the components and dependencies in the systems in question. Binary analysis can also be used to validate SBOM contents, or help understand gaps in the SBOM data.

Nonetheless, there is a key difference in how SBOMs are generated from a source repository, at the point of the building of the software, and for already-built software. While there are many unique circumstances, those requesting SBOM data should try to obtain it from the instance of the build since the instance of the build captures the details of the software as built, including reflecting any changes made by the compiler or other tools.

Flexibility vs Uniformity in Implementation

In many areas of security that cover a diverse range of software and contexts, a fundamental tension exists between the needs for flexibility and uniformity. This is not unique to SBOM. The sheer scope and scale of the software ecosystem leads to a host of unique considerations. This not only includes key distinctions between the uses of software (*e.g.*, traditional enterprise software vs. embedded systems vs. containerized software), but also the unique features of different languages and tools. At the same time, there is a clear need for some convergence and uniformity. Any organization would incur non-trivial costs to handle a wide range of SBOM implementations that are not easily compatible. The Federal Government and its agencies are no exception, and moving toward the benefits of the SBOM use cases described above requires some predictability and harmonization.

Successful implementation of SBOMs across the ecosystem will require both broad rules and policies, as well as specific areas of flexibility that are explicitly acknowledged. For the U.S. Government, the selection of these areas should reflect feedback from the community and agency stakeholders. Specific areas include legacy technology and higher assurance software, where active and ongoing threats may require more detailed supply chain information and stricter requirements.

Ultimately, all requirements built on the minimum elements should draw from two key concepts. First, all security, especially SBOM, is a process and not a single goal. Second, the fundamental principle behind SBOM is the power of transparency, and any rules or guidance should focus on enabling the use cases described in this document and elsewhere.

VI. Future SBOM Work

As this document has tried to emphasize, SBOM is an emerging technology and practice. Organizations are implementing SBOM today, but there is much more to do. The suggestions below are not intended to constrain future work or fully enumerate the potential for SBOM. Instead, they are highlights from a large and dedicated community from industry and government experts.

Most notably, it is important to stress that SBOM will not solve all security or supply chain attacks. Several recent high profile attacks in the supply chain did not target software components, but the tools and systems used to manage the software development and build process. Defenses against this type of threat are beginning to be discussed and even deployed in certain corners of the ecosystem.

The foundation for a more complete approach to securing the software supply chain is to securely capture details from across the software lifecycle, with cryptographic assurance. The minimum elements of SBOM starts this process, but there is more to do. Simply capturing more metadata is helpful, but effectively using this data requires automation, and automation requires the potential for both automated consumption and policy enforcement. This will require not just machine readability, but also semantic interpretation, which in turn, will require further work on data specifications and standardization.

Some of this data will naturally fit into the SBOM approach. This includes data about the *pedigree* and *provenance* of individual components, tracking the respective source of components, and their chain of custody across the software lifecycle. Other types of data, including some of the other secure development and supply chain security steps called for in EO 14028, may relate to software development, but might be better tracked separately and correlated with SBOMs.

The unique nature of modern application development and cloud-native architectures deserves further consideration for software transparency as well. Some modern software execution involves dynamic dependencies, calls to third-party services, and other dependencies not directly included in the software build. Inclusion of these dependencies ensures software is operated as intended and that vulnerabilities are not introduced through misuse. Further work is needed to fully characterize this data and assist in automated interpretation and use.

It is worth noting that several efforts to this end are under development today, and several more have been tried in the past, to varying degrees of success and longevity. As noted in the *Scope* section above, modular architecture can best support diverse innovation and adaptability.

Many of the issues discussed above will need further refinement, including software identity and SBOM distribution. Software identity will remain a hard problem, especially across different ecosystems. While a single, widely-used namespace might appear ideal, obstacles such as scaling, diversity, and the evolving landscape of suppliers make this unlikely. A diversity of versioning methods and systems also inhibits scalable automation for SBOMs and presents a number of related security data issues. Further coordination work can help each supplier identify and manage their own identification namespace, as well as reconcile incompatible standards and practices.

Since SBOM is an emerging technology, suppliers are still learning how to share this data with their customers. Fortunately, many suppliers already have trusted channels with their downstream users, including for software updates and support, although not all of these are automated or flexible. Several promising technologies have emerged or are being developed that might be well-suited to enable the discovery, access, and automated ingestion of SBOM data. Several technologies are worth noting here. The Manufacturer Usage Descriptor is a mechanism to allow a device to communicate important information about itself on the network, including network functionality[20] and, notably for this document, SBOMs.[21] The "Digital Bill of Materials" is a solution that supports multiple attestations about an open-ended set of supply chain data, including SBOM, and the enforcement of sharing and the access policies around these attestations.[22] OpenC2 is a standardized language for the command and control of technologies that provide or support cyber defense; work has begun to use it to handle, process, and act on SBOM data.[23]

SBOMs can also be handled by some set of trusted third-parties, which raises many of the usual strengths and weaknesses of relying on centralization. Challenges include trust and funding sources. One added benefit of some centralization that has been suggested is that "SBOMs gain greater value when collectively stored in a repository that can be easily queried by other applications and systems."[24] This can be used to gain more insights into systematic risk facing organizations, or even ecosystems or the country itself. It can facilitate coordinated vulnerability

[20] E. Lear, *et al.*, Internet Eng'g Task Force, *Manufacturer Usage Description Specification* (Mar. 2019), https://datatracker.ietf.org/doc/html/rfc8520 (specifies Manufacturer Usage Description (MUD)).
[21] E. Lear, *et al.*, Network Working Grp., *Discovering and Accessing Software Bills of Materials* (May 18, 2021), https://datatracker.ietf.org/doc/html/draft-ietf-opsawg-sbom-access.
[22] *Digital Bill of Materials – Documentation*, DBOM, https://dbom-project.readthedocs.io/en/latest/ (last visited July 6, 2021).
[23] Open Command and Control (OpenC2), https://openc2.org/ (last visited July 6, 2021).
[24] Exec. Order No. 14,028, *supra* note 1, § 10(j).

disclosure by giving coordinators an idea of which organizations or suppliers are at risk. Pooled data could help a centralized actor understand, which components are used the most widely, or in the most critical sectors, and prioritize security research or hardening practices. However, some have raised concerns that adversaries could take advantage and target those critical components for novel attacks. Further research is necessary to understand the optimal structure and incentives for sharing, protecting, and using SBOM data.

Ultimately, SBOM should not be seen as a unique security phenomenon, but yet another practice that can support the broader effort to secure the supply chain. In addition to linking this data to more supply chain data in the software domain, this data can be tied to hardware data. Hardware offers the features of a hardware root-of-trust and greater end-to-end assurance. At the same time, risks to the hardware supply chain pose their own challenges, and hardware-specific metadata should be considered and integrated into overall supply chain risk management along with SBOM data.

Lastly, as SBOM technology, tools, and practices mature, standards organizations should consider integrating them into voluntary, consensus-based standards. As noted in this discussion of minimum elements, SBOM covers both specific data and organizational practices. These have been developed in an iterative fashion to prove what works and affords rapid innovation. Given the cross-organizational nature of SBOM, it is important to demonstrate both feasibility and scalability *before* they are locked into formal standards. However, as evidence accrues of successful widespread implementation, sectors and standards experts should codify the technology and practices into international standards.

VII. Conclusion

This report is the product of carefully considered input from stakeholders across the government, private sector, and academia. The minimum elements of an SBOM are a starting point, based on what is conceivable today. The process of determining what constitutes the minimum elements of an SBOM should not end here. As the additional elements identified above in *Beyond Minimum Elements* and *The Future of SBOM* also become feasible, tested, and built into tools, those should be added to the minimum elements.

These minimum elements will be a key input into the Federal Government's work to improve the security and integrity of the software supply chain, particularly for critical software. Executive Order 14028 defines these next steps, notably calling for specific guidance, including "standards, procedures, or criteria."[25] To support and complement this work, the Federal Government should encourage or develop resources on how to implement SBOMs, potentially involving sector-specific or technology-specific details. It will be important to build on, and potentially expand,

[25] Exec. Order No. 14,028, *supra* note 1, § 4(e).

the public-private partnerships that have already been established and which have focused on defining and operationalizing SBOM's related supply chain work.

Finally, defining the minimum elements of an SBOM is an iterative process. Some elements are omitted from this listing simply because they are not yet feasible for the majority of stakeholders. However, in the near future, they likely will become feasible. This report should be seen as a starting point, rather than the last word.

Appendix A: Methodology

The Executive Order on Improving the Nation's Cybersecurity directed the Department of Commerce, in coordination with the National Telecommunications and Information Administration (NTIA), to publish the minimum elements for a Software Bill of Materials (SBOM). Upon receiving the tasking in EO 14028, the Department, through NTIA, began working to develop a preliminary approach to the minimum elements, basing the initial draft on the work drafted by stakeholders in NTIA's open, public multistakeholder process on software component transparency.

On June 2, 2021, NTIA published a Notice and Request for Comments (RFC) on the minimum elements for an SBOM, and what other factors should be considered in the request, production, distribution, and consumption of SBOMs. The Notice and RFC included the preliminary minimum elements for comment.[26] Comments were due on or before June 17, 2021. NTIA received approximately 88 comments, from stakeholders across the public and private sector, in response to this request.

In addition, NTIA performed interviews with a total of 32 Federal government officials representing both mission-driven and operational approaches to cybersecurity, from civilian, intelligence, and national security communities. NTIA analyzed the input received in the comments and the interviews and synthesized the material for this report on minimum elements.

The NTIA multistakeholder process on software component transparency was a separate process it convened in 2018. The first open, public meeting in July of 2018 allowed the participants in Washington and participating online to debate the overall issue of software component transparency. From this initial meeting, a series of working groups were established to tackle transparency from several different perspectives. One group tackled the overall challenge of what a "software bill of materials" should look like, the second documented existing and potential use cases for transparency, while the third reviewed the existing standards and formats that could be used to convey SBOM data. A final working group proposed and executed a "proof of concept" exercise with medical device manufacturers and the hospitals that used these devices. Working groups met weekly or biweekly to define their charter and workstreams. All working groups were open, and new stakeholders joined as the project progressed. Each group was led by 2-3 co-chairs who volunteered from the community. The broader community reconvened every 3-4 months to share progress from the different working groups and discuss the overall direction of this initiative.

For the multistakeholder process, NTIA acted as a convener and a neutral facilitator, and helped the different working groups coordinate as their work progressed. Those work products were drafted by stakeholders with frequent opportunities for the community to weigh in as the

[26] NTIA, *Software Bill of Materials Elements and Considerations*, 86 Fed. Reg. 29568 (June 2, 2021), https://www.ntia.gov/files/ntia/publications/frn-sbom-rfc-06022021.pdf.

deliverables took form. The documentation from that process provided valuable insight for this activity to propose the minimum elements of an SBOM, called for in EO 14028.

Appendix B: Glossary

Authenticity
The property that data originated from its purported source.[27]

Author
An entity that creates an SBOM. When author and supplier are different, this indicates that one entity (the author) is making claims about components created or included by a different entity (the supplier).[28]

Component
A unit of software defined by a supplier at the time the component is built, packaged, or delivered. Many components contain subcomponents. Examples of components include a software product, a device, a library, or a single file.

Consumer
Entity that obtains SBOMs. An entity can be both a supplier and consumer, using components with SBOM data in its own software, which is then passed downstream. An "end-user" consumer (that is not also a supplier) may also be called an operator or a leaf entity.[29]

Dependency
Characterizing the relationship that an upstream component X is included in software Y.

Downstream
Referring to how a component is subsequently used in other pieces of software at a later stage in the supply chain.

Integrity
Guarding against improper information modification or destruction.[30]

Lifecycle Phase
The stage in the software lifecycle where an SBOM is generated (*e.g.* from source, at the time of build or packaging, or from a built executable).

[27] Morris Dworkin, Nat'l Inst. of Standards and Tech., Special Pub. 800-38F, *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping* 3 (2012).

[28] *Framing Software Component Transparency, supra* note 3, at page 24.

[29] *Id*.

[30] 44 U.S.C. § 3542.

Open-source software
Software that can be accessed, used, modified, and shared by anyone.[31]

Pedigree
Data on the origins of components that have come together to make a piece of software and the process under which they came together. This could include data beyond the minimum elements, such as compiler details and settings.[32]

Provenance
Data about the chain of custody of the software and all of the constituent components, potentially including data about the authors and locations from where the components were obtained.[33]

SBOM (Software Bill of Materials)
A formal record containing the details and supply chain relationships of various components used in building software. Software developers and vendors often create products by assembling existing open source and commercial software components. The SBOM enumerates these components in a product.[34]

Software-as-a-Service
The capability provided to the consumer to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (*e.g.*, web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.[35]

Subcomponent
Constituent part(s) of a component.

Supplier
An entity that creates, defines, and identifies components and produces associated SBOMs. A supplier may also be known as a manufacturer, vendor, developer, integrator, maintainer, or provider. Ideally, all suppliers are also authors of SBOMs for the suppliers' components. Most suppliers are also consumers. A supplier with no included upstream components is a root entity.[36]

[31] Nat'l Inst. of Standards and Tech., S 6106.01, *Open Source Code* 2 (2018).
[32] *Roles and Benefits for SBOM Across the Supply Chain*, *supra* note 4, at page 26.
[33] *Id*.
[34] Exec. Order No. 14,028, *supra* note 1, § 10(j).
[35] *The NIST Definition of Cloud Computing*, *supra* note 17, at page 2.
[36] *Framing Software Component Transparency, supra* note 3, at page 24.

Transitive Dependency
Characterizing the relationship that if an upstream component X is included in software Y and component Z is included in component X then component Z is included in software Y.

Upstream
Referring to the origins of components or subcomponents, at an earlier stage in the supply chain.

Appendix C: Acronym List

CPE	Common Platform Enumeration
CPU	Central Processing Unit
EO	Executive Order
NIST	National Institute of Standards and Technology
NTIA	National Telecommunications and Information Administration
OSS	Open-Source Software
PSIRT	Product Security Incident Response Team
PURL	Package Uniform Resource Locator
RFC	Request for Comments
SBOM	Software Bill of Materials
SPDX	Software Package Data eXchange
SWID	Software Identification
VEX	Vulnerability Exploitability eXchange