June 17, 2021

VIA EMAIL: SBOM_RFC@ntia.gov

Attn: Attn: Evelyn l. Remaley
Acting NTIA Administrator
National Telecommunications and Information Administration
U.S. Department of Commerce
1401 Constitution Avenue NW,
Room 4725,
Washington, DC 20230

Re: McAfee's comments in response to NTIA's Request for Information (RFI) on *"Software Bill of Materials Elements and Considerations"*, Docket No. 210527–0117

McAfee Corp appreciates the opportunity to respond to the National Telecommunications and Information Administration's Request for Information on "*Software Bill of Materials and Considerations*," posted on June 2, 2021.

McAfee, a world leading independent cybersecurity company, is focused on accelerating ubiquitous protection against security risks for consumers, businesses, and governments worldwide. Inspired by the power of working together, McAfee creates cybersecurity solutions that make the world a safer place. For consumers, we help secure their digital lifestyle at home and away. For businesses, McAfee Enterprise cloud security extends from device to cloud with data visibility, data loss prevention and advanced threat protection on a platform that supports an open ecosystem. Our holistic, automated, open security platform allows disparate products to co-exist, communicate, and share threat intelligence with each other across the digital landscape. We enable the convergence of machine automation with human intelligence so our customers can streamline workflows more efficiently, be freed from operational burdens and be empowered to strategically combat threats from adversaries.

Our response includes answers to the specific questions asked in the Request for Information, as well as general comments on aspects of the proposed use, development, and adoption of and areas to be considered during the SBOM development process. We do not address Continuous Integration / Continuous Development processes that are more cloud focused.

Myths and Reality of an SBOM

Before beginning our comments and recommendations as a part of this request for comments, we would like to level set on what an SBOM is not.

- An SBOM is not a piece of product security by itself. It is a set of information about the software package that can be useful in providing transparency to the end user. It identifies what third-party components are included in the product. It specifies official product naming, it indicates specific version information and SBOMs should be required to be digitally signed by the software publisher, so as to be used authoritatively. An SBOM is simply a file of delivered software package data points.

- Having an SBOM would not have helped or assisted in any way in preventing or identifying the Solarwinds attack. In fact, it would not have been beneficial in any of the recent high-profile hacks.
- An SBOM is not an attack vector for malicious actors. Public comments by panelists at the June 2021 NIST EO Workshop stated SBOMs would be used by attackers to hack the associated products. The thought that adversaries will actually use the SBOM information to hack a product is rather humorous. While they may read it, attackers will reverse engineer the software's binary files and determine specifically what libraries, components and system calls are used. They do this today. By reverse engineering software, malicious actors have full access to all the information that constitutes the binary. This has been done for a more than a decade. An SBOM does not and will not change that.
- There is no standard format for an SBOM, as indicated by the Executive Order 14028. The EO requested NTIA produce a documented set of *minimum SBOM elements* within 60 days. If there was a standard format, it most likely would have been specified for use.
- SBOMs are not commonplace today in the vast majority of delivered software. Government agencies are discussing and considering requiring SBOMs as if they are real and existed today. In fact, in Section 10 Definitions of the EO, the description of an SBOM describes an object that does not exist today past small proof of concepts and a very few vendors.

> *(j) the term "Software Bill of Materials" or "SBOM" means 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. It is analogous to a list of ingredients on food packaging. An SBOM is useful to those who develop or manufacture software, those who select or purchase software, and those who operate software. Developers often use available open source and third-party software components to create a product; an SBOM allows the builder to make sure those components are up to date and to respond quickly to new vulnerabilities. Buyers can use an SBOM to perform vulnerability or license analysis, both of which can be used to evaluate risk in a product. Those who operate software can use SBOMs to quickly and easily determine whether they are at potential risk of a newly discovered vulnerability. A widely used, machine-readable SBOM format allows for greater benefits through automation and tool integration. The SBOMs gain greater value when collectively stored in a repository that can be easily queried by other applications and systems. Understanding the supply chain of software, obtaining an SBOM, and using it to analyze known vulnerabilities are crucial in managing risk.*

Sadly, this is not the case. The picture painted in this section of the EO is aspirational. While we truly believe this vision is the best direction for the software industry to go, and we are highly supportive of SBOMs in general, there is still a required major uphill push to make SBOMs prevalent within commercial software today. The U.S. Government should not be requiring vendors to produce something before real specifications, as to what is required, how they work for specific use cases, are documented, and understood by government and those required to provide them.

And it should also be noted that alone, an SBOM solves nothing.

With the myths and misconceptions on the table, it is important to note that SBOMs can and will provide the solid and vital foundation for critical future software supply chain functionality, addressing asset management, vulnerability, and software naming issues, at a minimum.

From the Request Comments

Potential Elements of an SBOM

The RFC specified three areas of focus, data fields, operational considerations, and support for automation. As stated earlier, while an SBOM alone does little by itself, it can provide a foundation for additional capabilities that enhance software supply chain security.

From the RFC:

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*

Some of these data fields could be expanded. For example, the ''dependency relationship'' generally refers to the idea that one component is included in another component, but could be expanded to also include referencing standards, which tools were used, or how software was compiled or built. Other data fields may need more clarity, including data fields for component and supplier name. ...

While we understand the intent of an SBOM is to assure third-party components are identified, this entire request for comments assumes that ALL software uses someone else's code. That is neither accurate nor helpful. There has to be an understanding that SBOMs can be acceptable without third-party components specified. (If we are talking about minimum data fields and they are not all needed, what is the real minimum?)

It must be pointed out there is more than one way to develop software and if SBOMs are going to be required at some point in the future, they need to be able to support all types of software delivered. For example, software such as firmware is not addressed at all in the RFC. While embedded systems are briefly mentioned, it would be hard to provide a SBOM for embedded systems without more specific information being supplied about the type of device the embedded system was to be run on.

From the RFC

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:*

We disagree with the above statement. A specification describing what each of the SBOM elements mean in respect to the operational and business decisions has not been officially defined. Currently this is still very much a work in progress. There is a general understanding and several draft documents that are subject to change. The purpose and use of the individual elements must have their operations and business logic documented and made available to the software publisher community as official specifications before the above statement is accurate.

- *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.*

We believe that SBOMs must always reflect the current status of the delivered software. To accomplish this, it is critical that SBOMs are generated as an artifact of the actual build process and as such, any new build or update MUST create a new SBOM. Any process requiring a distinct step involving humans is bound to break and cause incorrect information to be delivered in the SBOM. SBOMs must be created automatically, with no human involvement, during the product's build process.

- *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.*

This is where things get a bit confusing. Why can this not be specified? Product SBOMs should indicate the third-party component(s) incorporated into the product. There is no real value to the purchaser to clutter up an SBOM with massive amounts of information that is not relevant to the specificity of what is a direct part of the product. It will rarely be completely possible to accurately list every single dependency at all levels for every open-source project all the time. If this is required, every SBOM will have a statement "Known unknowns" in the SBOM. What and where is the real value in this? This type of information is needed for the proper support and assurance for the software in a pre-build situation. This is really not appropriate in an SBOM delivered to an end user. With component associated URL information provided in the SBOM, (described below) vulnerability information can be available immediately instead of waiting for vendors to provide it.

Dependency information could be provided on request of the product vendor or third-party commercial or open-source project if there is a need to see all the dependencies of dependencies at all levels. The only thing this accomplishes is to make the SBOM files nearly unreadable and massive in size. Many commercial products use the same open-source projects. How many sets of "openssl" dependencies should one network have to be burdened with? This type of dependency information should be stored once on a network to be queried if desired. It should not be incorporated directly into every single product SBOM for every product the organization owns and uses. Additionally, requiring this type of massive dependency information in an SBOM, makes automatic SBOM creation during the build process highly problematic. This type of requirement is better suited for additional tooling, not an SBOM.

- *Delivery. SBOMs should be available in a timely fashion to those who need them and have proper access permissions and roles in place.*

As we have stated before, initial delivery of SBOMs must be automatically included with all software products delivered. All software needs to have an SBOM, and it needs to be created when the software is built and updated when any new build or update takes place. SBOMs need to be available in the product and on demand from a customer. They could even be published on a vendor's support website. Once created, there are multiple ways an SBOM would be provided to the customer. There would be no reason to limit the availability of that build artifact.

From the RFC

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.*

We could not agree more. If automated, the SBOM becomes just another part of a delivered product. If software publishers need to add in any number of manual steps to build and deliver an SBOM, the costs and errors will slow adoption.

SBOM files must be machine and human readable. We do not believe it was the intention of the EO to limit SBOM files to only machine readable. They were simply stating the SBOM file must be structured and paersable via software. There is absolutely no reason to prevent the SBOM attribute information from being readable by a human. By restricting it to just a machine-readable format then additional tooling has to be supplied if SBOM files are to support a Product Support use case.

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, CycloneDX, and SWID tags. 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.

The only real international standard of the three that have been specified, SWID was initially created and approved in 2009 and further updated to the current version as specified in ISO/IEC 19770-2:2015. The standard was reviewed and confirmed in 2021 and remains the current version. The National Institute of Standards and Technology (NIST), in cooperation with the Department of Homeland Security (DHS) and the National Security Agency (NSA), has developed NIST Internal Report (NISTIR) 8060: Guidelines for the Creation of Interoperable SWID Tags. as a companion to the ISO/IEC 19770-2:2015 standard. SPDX is pursuing an international standard status and is in draft form but not completed at this point. ISO/IEC DIS 5962 is currently a work in progress. CycloneDX is not an official national or international standard but a consensus document.

Additionally, an IETF draft (https://datatracker.ietf.org/doc/draft-ietf-sacm-coswid/) is in the final stages of specification development to be able to leverage SWID usage for memory constrained environments.

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.

We agree there will need to be additional automation defined to support specific use cases. The vulnerability use case was the initial use case discussed, but with a proper specification created on what constitutes an SBOM and documented means to expand SBOMs to support additional use cases, SBOMs can become a valuable aspect of a maturing software supply chain.

However, a word of caution. Multiple ways to create and interpret SBOM files can be confusing and more costly for SBOM creators, SBOM interrogation vendors, such as vulnerability scanners or asset management tools and the end user community. The situation could get even worse if there were capabilities available in one format but not in another. If the desire is to be able to support any "SBOM format" that comes along, then we really fragment the intended value of SBOMs.

Request for Comment

To inform, validate, and update NTIA's understanding of SBOM, NTIA seeks comment on the following questions:

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

No. The list is not complete to support the vision of the SBOM definition in the EO. There are multiple things missing from our perspective just to make a minimum SBOM possible.

Supplier name:

The Supplier name needs to have more to it than just a name. There needs to be a globally unique identifier so there are no conflicts or misunderstanding as to who the "supplier" is. Additionally, what type of supplier created the associated SBOM? Are they the software's creator, are they a software distributor selling multiple products in a bundle, are they a third-party creating an SBOM for legacy use? There are other roles that may be needed and possible. There should be some indication as to the role of the SBOM supplier.

Product or Product Bundle:

The list of minimum data fields does not really address the specific product naming and product versioning needs of an SBOM. Product naming is critical and needs to be specifically identified. Product versioning information needs to be specified here as well at a minimum.

One thing that complicates naming is the software market in general. Acquisitions of companies and corporate carve-outs can cause products to be renamed. It would be useful to have an optional data field for those products that have had their name officially change. An "alias" or similar field should be specified so as to allow the previous official name to be recognized for use in inventory, vulnerability or procurement use cases.

It is not unusual for products to be sold as a bundle. Bundle information is totally missing. Bundles too have versions so there should be the ability to specify the Bundled version information.

Cryptographic hashes of the product MUST be provided. In the case of a bundle of products, all products that makes up the bundle should each have an associated hash.

In some cases, the proper permissions for the installed software may need to be specified to assure the product(s) are installed by default in a restricted and more secure configuration.

Components

Component names are generally assumed here to be third-party components, whether they are open source or commercial, used to describe a piece of the overall product and not the product itself. Components are, for the sake of this response, considered some piece of software that makes up the overall product. Third-party externally developed software and software libraries are what is specified most often. However, it is possible for configuration files, and even software files used to construct the program by the vendor could be included for developmental and product support functionality. The use of specific software files would not normally be needed or required as a part of the delivered product SBOM. However, an expanded SBOM could be generated and retained by the software creator at build time for solely development and support purposes.

Component names should refer to the additional parts of the program that the SBOM / product publisher wishes to provide transparency on. Each component needs to have a

- Component name
- Version of the component
- Cryptograph hash of the component

Additional Component information that should be considered to be included:

- URL to location of the original source of the third-party component. This could be the location of the open-source project or if a commercial library, a support page on a site of the company that produced the component.
- A URL link to a page on the vendor's website that can be used as a means for the vendor to communicate status on the use of the library component for the end users. This is important for allowing the vendor to communicate status and intent in the case vulnerabilities are discovered in the third-party component. In some cases, the vendor may not be affected by the vulnerability issue and this webpage provide a means to state that without having massive customer calls to their support desk. In other cases, the vendor may be vulnerable. In that case, the web page can be used to inform customers of the status, potential temporary mitigations, the availability of a patch or expected time to patch availability. This provides the capability for better, more transparent communications between the vendor and their customers while not putting any undue burden on the vendor's support staff. Vendors that provide this type of capability demonstrate they are committed to transparency and productive and timely communications with their customers.

Extending SBOM use case capabilities

- Any other unique identifier

  There needs to be mechanisms to support the expansion of SBOMs to address capabilities beyond what is specified in the minimum set of elements. In adding support for additional use cases, additional information will need to be specified in an SBOM. The outcome of the EO and NTIA efforts will not result in successfully defining a complete SBOM. There will be expanded use cases and functionality possible because of the existence of SBOM. It is critical that SBOM supplied information be extendable. The use of specifying a minimum element as "Any other unique identifier" in the perspective of the EO, seems odd and out of place. This is counter to the "publish Minimum elements for an SBOM".

Dependency relationship

The request for comments stated "the ''dependency relationship'' generally refers to the idea that one component is included in another component, but could be expanded to also include referencing standards, which tools were used, or how software was compiled or built.”

In the first case, the dependency relationship is an outcome of the structure / format used to implement an SBOM. In the second case it is a specific set of additional information not specified in the minimum set of SBOM elements. In both cases, these need to be documented in the required specification for SBOM creation and not a concept. Without specifics, it is not possible to accurately respond.

Author of the SBOM data

It is important that the creator of the SBOM is specified. This “author” could be specific a specific organization, but it will not likely be a human. It will be more likely a piece of software that creates the SBOM. Knowing what organization and tool actually created the SBOM and when would be useful.

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

The following are examples of potential use cases for SBOM files given a digitally signed SBOM file:

- **Package Verification** – Verification of installation media at acquisition and at the time of installation would rely on cryptographic hashes specified in the SBOM file to assure all the components of the software package are installed on the disk appropriately.

- **Software Inventory** – Validation by asset tools in locating and determining the numbers and currency of the installed products. This allows a site to discover if they are running more than they should and that the versions of the software being run is the current version with the latest patches.

- **Software Integrity** – Software tampering detection is not a capability that has seen widespread use in the software industry but can be critical to assuring the pieces and parts of the software package or suite/bundle have not been altered by malicious actors or by accidental disk corruption. Signed SBOM files with cryptographic hashes for the individual parts that make up the software can be used to assure the software provided by the publisher is what was delivered and should execute in the manner the software publisher expects it to. This can be done just prior to execution or on-demand, validating the cryptographic hashes to assure the software and its associated components have not been tampered with.

- **Component Vulnerability** – Provide a means for correlating known vulnerabilities in third-party software and the identification of patches and updates to remediate a vulnerability. An enterprise can determine what components of their environment are vulnerable based on a known library vulnerability (e.g., Heartbleed). Not knowing when there are vulnerable software components in the enterprise environment serves no one but the attackers. It is important for network staff to be able to quickly identify vulnerable software that needs to be updated or mitigated. Being able to see what third party components are included in software provides the ability for the user community to be able to know the true security posture of their deployed software.

It is also important to know what products are not vulnerable. Today commercial and open-source library components are commonplace and while many components have a common origin, their support and evolving development often take different paths. If a vulnerability is discovered in an open-source version, it does not mean that all versions derived from that project are vulnerable. The same holds true for derived software. If a commercially derived version has a vulnerability, it does not mean the open-source version does. It is important to be able to accurately identify and distinguish which third party components are and are not vulnerable as it will reduce the organizational activities needed to respond.

- **Procurement Transparency –** Construction of software may include third party libraries that are not desired by the organization purchasing it. There may be other considerations as to the development of the software, such as the percentage of third-party code to vendor code, those in control of a third-party project development, etc. Requiring third-party component information to be specified in an SBOM allows the end-user organization to be able to set their own rules for what are acceptable third-party components to run in their infrastructure.

- **Development Support -** SBOMs can assist developers in exactly knowing which version of the software they are running while developing either additional features or bug fixes. There is no ambiguity when it comes to addressing a customer problem as to which version of the software, when it was built and from what components, the bug affects. Hashes can be used to assure the developer or tester is recreating a bug with the exact version a customer has on-site. This too makes it more efficient for the developer to address external issues, whether reported as a functional problem or a security issue. If SBOMs are automatically generated for every single build (as reality requires), developers can now use this information to document development issues before the product is ever released.

- **Product Support**
  SBOMs provide the ability for product support teams to quickly determine what version of their software the customer is calling about. They can ask the customer to read entries from the SBOM file or simply share it on-screen during a video call. Immediately, support staff knows what it is they're dealing with, what version and can use that info to determine if there are additional updates the customer should install or information the customer should know about. This reduces the support costs as it minimizes the amount of time that support staff needs to be on the phone with any specific customer.

- **License Analytics** - Correlation of software to licenses for planning, acquisition, reconciliation, and optimization of license use. This is similar in nature to the software inventory use case but often accomplished by a different organizational team.

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.*

      Response: We disagree with the assumption that there are too many standards for software identity and that is why it has not been successful. The problem lies with the software creators/publishers/vendors. For far too long vendors have had multiple ways to name their software. They created product names for families of software, they created individual product names, and they have created different naming for marketing purposes. Sometimes version information further complicates product naming. Often a product is known in the market by three or four "common" names. Combine that with the inconsistent ways that product versioning is done, and we have a real mess.

Efforts around producing consistent naming for software products have produced solid standards for syntax and usage. The assumption with those standards was that vendors would use them to name their products. It was hoped that vendors would supply "official" names for their products that industry could use and recognize. This has never happened in a universal way valuable to the marketplace. The NIST Common Platform Enumeration (CPE) was seen as the way forward at one point in hopes of solving the naming problem. What NIST and industry discovered was without software vendors supplying their official product names, we still had the same problem. It is critical that the responsibility for proper product naming rests in the hands of the vendors. Only they can designate an "official" product name.

The product name specified in SBOM files must be understood moving forward as being the "official" name for the product the SBOM file is associated with. Using SBOM files to allow the producing organization to define this for industry-wide usage is vital to properly addressing the product naming issue. If SBOMs are widely adopted, and the SBOM files are generated as an automatic part of the build process, this would allow vendors to indicate an official name without having to specify it "out-of-band" as was required by efforts such as CPE. This becomes just an artifact of the SBOM generation at build time. We believe this is the proper way to address the naming issue and do so as a natural outcome of the use of SBOMs.

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

We believe this is not an appropriate use of an SBOM. The SBOM is for software to be delivered to a customer organization so they can determine the components used in the delivered software package. While there may be a need to be able to determine if a CSP is running vulnerable software in their execution environment, we should not try to us the SBOM as a means to achieve this. This adds unnecessary complexity and confusion. CSPs use Continuous Integration / Continuous Development techniques that need a different means for making this useful for an end-user organization.

Many of the use cases and reasons SBOMs are valuable are not present in a cloud-based service. Asset management is now the cloud providers responsibility. The run time environment is not being managed by the customer organization. Having the transparency of what is built into the software is no longer a concern to the customer organization as they are paying only for a service. Searching for vulnerabilities now falls on the side of the service provider. Potential use cases around licensing are fully supported by the service provider in terms or quotas on access and the need to true-up licensing is not relevant in the CSP environment from the customer organization/vendor relationship perspective.

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.*

   We hope the generation of SBOMs is based on an authoritative source in order to provide a legitimate set of the information within. It does no one any good if you cannot trust the information in an SBOM. If this is to be generated by a third party--such as NIST, in association with the National Software Reference Library (NSRL)—the information cannot be truly trusted. If the organization is running End-of-Life or End-of-Support software, they have bigger problems than an SBOM. SBOMs are not a silver bullet. SBOMs should be an authoritative source of information about a software product.

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.*

   All SBOM files should be digitally signed with the corporate keys used by the organization producing the software. This provides a basic foundation of trust that the information contained within the SBOM came from the actual authoritative producer of the software. This also means that the hash information for the specific components listed within the file came from the vendor. That component hash information can be used to then validate of the software installation on disk. SBOMS must contain hashes of all the individual components that are part of the software package being installed. These hashes can be used to assure that all components of the software package are installed on the disk appropriately the hashes can also be used at execution time to validate that the hash of the application being executed matches what is in the documented signed SBOM file. This gives the execution environment one additional check to assure what is about to run has not been tampered with.

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?*

We believe this may be overreach and not an appropriate use case for an SBOM. This question seems to ask, can an SBOM file solve all the ills of the software supply chain? The answer to that question is NO. The way in which development organizations develop software are not standardized. Some have highly mature Secure Software Development Lifecycles while others have a compiler and a set of source files. Trying to be a one-size-fits-all solution the entire secure software supply chain is a recipe for failure on all levels. Let the SBOM be focused on being the foundation for the set of use cases defined earlier and not try to be some massive overreach that cannot be successful.

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?*

Without knowing what is actually envisioned it is not possible to even provide the most basic answer to this question. SBOMs must be extendable past the current set of proposed use cases. Once more is known as to what NIST's definition of "critical software" is, there may be options and there will definitely need to be more discussions. At this point in time any answer to this question is pure speculation.

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.*

This question seems more like a statement than an informational request. Once an SBOM is generated during the build process, it is available for delivery to those authorized to have it in multiple ways, such as from a product support web site, delivered separately to the customer outside the actual software package, delivered as a part of the actual software package, etc. The market and specific customer needs will determine the ultimate answer to this question.

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.*

As stated previously, a complete dependency graph of all dependencies for every aspect of every piece of third-party code is problematic in many different ways. It will make automatically creating SBOMs during the product build process more extensive and potentially error prone. The SBOM is not the right palace for this type of information.

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.*

j. Direct vulnerability information should never be included in an SBOM. Vulnerability information should be a use case for generating correlations between the third-party components and known vulnerabilities such as through the CVE data. Embedding any direct vulnerability information, past a description for a patch related SBOM is a mistake. As indicated earlier in this response, there are ways to "*note that the existence and status of vulnerabilities."* There are really two questions being asked here. How to keep an SBOM up to date and how to convey information to the customers? If an SBOM is an automatic artifact of a software patch build, then it is up to date when the patch is delivered to the customer. As for the communications aspect of *signal about whether the SBOM data is up-to-date relative to all relevant and applicable vulnerability data sources"* this can be addressed by the product vendor providing a static link for each third-party component used in the product to a vendor support web page. This allows the vendor to reflect the status of the vulnerability discovered in a third-party component. The page can provide the status if affected by the vulnerability, or a disclaimer that is it not vulnerable and why.

- Component name
- Version of the component
- Cryptograph hash of the component
- URL to location of the original source of the third-party component.
- URL link to a page on the vendor's website component support page.

As stated earlier, this provides the capability for better, more transparent communications between the vendor and their customers while not putting any undue burden on the vendor's support staff.

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

VEX is in the very early stages of development and it remains to be seen if it is in fact a viable solution. As mentioned in the previous (j) question, if the vendor believes they are not vulnerable for whatever reason, they can document that on the product's component support page. The vendor can then explain to its community of users why they should not be concerned. This is a simple and effective way to provide timely communication to their user community.

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?

We believe we have specified why we feel SBOMs are the right way to provide information to the end-user organizations. The SBOM effort was started to provide focused software transparency for customers. Situations such as Heartbleed caused massive numbers of customers to call our support desks to determine if we were using the openssl libraries that were affected. This was a serious problem for most organizational vulnerability management teams and the US Federal government as they could not determine what products on their network were affected. The idea behind SBOMs in the first place was to provide transparency to the third-party components included in vendor related products delivered to end-user organizations. While the intentions may have been diluted/changed/lost, SBOMs must be beneficial to the end-user customer community, or they have no value at all.

McAfee Summary

The need for SBOMs is about software transparency on one side and cost reductions and process efficiencies on the other. Software transparency is really a problem with the vendor community, not the open-source community. Federal network and security operations staff can download open-source packages and read all the code, understand all the dependencies, and get some history on those developing it. That is not the case for binary-delivered vendor software products. Proprietary software needs to shine a light on what third-party components are being used.

- SBOM files should be automatically generated as a natural artifact of the product build environment with no manual (human) steps needed. This way every build regardless of development or production depicts the SBOM information of that built version.
- Product names in SBOM files must be understood as being the official name for the product the SBOM file is associated with. (Naming is a hard problem. Using SBOM files to allow the producing organization to define this for industry wide usage is a plus.)
- Cryptographic hashes of all pieces of a software package must be available including third-party components.
- SBOMs must be digitally signed with the corporate keys used by the organization producing the software.
- SBOMs need to interrogatable on disk by third-party products such as the installed product, vulnerability scanners or asset management tools.
- SBOMs should provide a means for improving the inevitable communications needed to describe the vulnerability status of a product affected (or not) by a third-party component vulnerability.
- There must be a real specification developed for defining the elements of an SBOM, it's *operational and business decisions and actions that establish the practice of requesting, generating, sharing, and consuming SBOMs.*
- Finally, the development of the SBOM specification needs to focus on the use cases discussed and the ability to expand into new use cases when deemed appropriate. It

should not try to be all things to all people. It cannot solve the software supply chain problems. It is but a piece. Focus for success and adoption.

Software publishers need to have efficient ways of alerting their customer community of vulnerabilities in those components, while informing them of the timeline to correction, short term mitigation techniques or that there is nothing to be done because the product is not affected and why.

As more and more open source is incorporated into application development, it is important customer organizations can determine the sensitivity or vulnerability of certain applications running on their network.

Summary

As NTIA has publicly stated, an SBOM alone solves nothing, but is a foundation for other functionality that will address asset, vulnerability, and software naming issues. We could not agree more.

McAfee supports the definition and adoption of SBOMs focused on providing the transparency that the binary delivered software product vendors do not provide today. As we have done in the past, we will be happy to work with NTIA and other governmental agencies to define a way forward beneficial to its public and private customers and the software vendor community. We need to assure we are using the right tools in the right way to continue to improve and mature our software supply chain infrastructures.

Respectfully Submitted

Kent Landfield
Chief Standards and Technology Policy Strategist
McAfee