SYNOPSYS®

NTIA Request for Comments - Software Bill Of Materials (SBOM) – Synopsys response

As a participant in the NTIA multi-stakeholder process since its inception, Synopsys welcomes the valuable work NTIA has completed to date and looks forward to the opportunity that broad adoption of Software Bill of Materials (SBOM) provides in support of the objectives outlined in Executive Order 14028. Having SBOMs for network-connected assets provides one of the better means for organizations to more quickly respond to threats by aiding in identifying if the threat is applicable and, if so, in which assets the threat is targeting vulnerabilities and weaknesses.

In response to the NTIA request for comments, Synopsys is pleased to provide a perspective that focuses on four topics, which we summarize as follows:

- Operational considerations for managing software components in modern software delivery processes

  Software delivery processes are wide and varied, and the line between who is a 'producer' and who is a 'consumer' may be blurred – meaning that the architecture of SBOM should be amenable to this seamless flow of software from one entity to another, typically through automation.

  Open source software distribution introduces specific challenges with regard to determining authenticity, provenance, and pedigree, and SBOM must be flexible enough to allow automated trust decisions to achieve success at scale.

  To facilitate this, SBOM cannot exist in isolation and cross-correlation with additional data sources should be expected in order to provide a complete picture of risk and authenticity.

  Risk scoring should be cross-referenced by SBOM data, such as provided by OpenSSF's CII Best Practices badge project and others, including: Community Health Analytics Open Source Software (CHAOSS), the OpenSSF Security Metrics Project, the OpenSSF Security Reviews, and the OpenSSF Security Scorecards.

  Recommendation for minimum SBOM:

  - Leverage existing methods and practices for generating open source SBOM information, already established at software vendors today, to support the generation of minimum SBOM data.
  - Leverage currently available risk scoring schemes for open source components to provide better informed risk decisions associated with software.

- Baseline data and the benefit of SBOM as the key to a database of software trust and authenticity information

  Minimal core SBOM information, with the ability to support linkage to and from external data sources as required for specific use cases should be preferred – recognizing that such external data sources may include proprietary, confidential, or embargoed information that could change over time.

  Resolving software identity and historical provenance problems within the SBOM should not be a near-term goal; instead, initial SBOMs should capture enough information elements to enable the recipient of the SBOM to make their own determination regarding the identity of the software being described, according to the level of confidence desired for their particular usage.

  Rather than introducing new or adjunct forms of identifiers, an initial approach would be to adopt best usage of already defined namespaces and delegate responsibility to each of the respective originators to maintain conventions for usage.

Recipients of SBOM should be able to determine the authenticity of the SBOM and authority of the SBOM author with ease.

Recommendation for minimum SBOM:

- Leverage those existing data fields already easily accessible via current SBOM practices – package names, version numbers, and suppliers – and augment in future with richer data as SBOM matures.

– Retrospective SBOM generation for forensic and legacy purposes – including embedded systems and SaaS services

Generating limited SBOMs for legacy devices may be achieved utilizing binary analysis techniques, however with some caveats. There remains a question of whether forced SBOM generation by third parties might, in some cases, be prevented by contractual license terms.

For SaaS services – the questions of SBOM for ephemeral instances and infrastructure as code, as well as the requirement and granularity of SBOM retention for forensic purposes must all be considered by service providers as well as consumers. Similarly, in a shared responsibility security model commonly seen in cloud services – responsibility for SBOM maintenance should be clear at each different layer.

Recommendation for minimum SBOM:

- Accept or expect incomplete binary SBOM data for legacy devices as practiced today, to provide initial visibility, in lieu of more detailed SBOM in future.

– Critical factors in successful and widespread SBOM adoption

Many organizations worldwide already leverage Synopsys tools and services to generate a Software Bill of Materials today, typically for the purposes of declaring their use of open source for license compliance reasons. Synopsys tools support the SPDX standard which is well established and fulfills this need as well as the requirements stated previously.

In order to achieve broad adoption, SBOM must scale though automation. In turn, SBOM tool providers will need a mechanism to assure that their tool can produce the required SBOM data, and end-users will seek confidence in this before they begin to adopt automated SBOM practices.

Whilst the technical ‘how to’ will enable initial SBOM start-up, in the medium term, challenges will primarily be organizational and operational. Consideration must also be given to providing clear guidance targeted by persona – executives, systems and procurement managers, and end-users – such that they may plan the roles and responsibilities, and resource requirements to achieve success.

Recommendation for minimum SBOM:

- Utilize an existing established format such as SPDX, already in use for exchange of software license information, to accelerate rapid machine readable SBOM transmission.

We look forward to continued collaboration with NTIA, NIST, and other entities in support of the ongoing integration of SBOM as a foundational element in securing the software supply chain across government and industry.

Operational considerations for managing software components in modern software delivery processes

Modern applications are no longer exclusively created within the proverbial four walls of a supplier by known employees and using commercially produced technology stacks. Instead, applications often include software from loosely affiliated, globally distributed teams of developers who create software components and solutions following open source development patterns. This also means software components may be used and modified in ways it was never intended or thought of. Managing acquisition, usage, and sustainment of that open source is fundamentally different than managing commercial or contracted software.

The evolution of SBOM should be compatible with the complexity of modern software development including the integration of software and dependencies from many sources to create user applications and provide transparency into both the provenance of code and the associated environment each component was developed within, as well as the environment where it was all integrated.

Meeting that desired outcome requires greater transparency into the software development processes employed by all Producers, including:

- An understanding of where all code composing an application originated. For commercial/proprietary and contracted applications, this origin point is likely to be a single software supplier (or prime contractor), but with the prevalence of open source software and the reality that significant portions of commercial/proprietary and contracted software are of open source origin, how that open source software is developed and released must be factored in. Key to this understanding is a recognition that access to source code implies that users can create derivative works, implying that there might be no unique repository for most open source components, nor is there a single download location for any given open source component. As an example, the source code for OpenSSL is currently available from almost 7000 forks on GitHub and from numerous suppliers each with varying implementations;
- An understanding of what suitable content of the SBOM. Should items such as configuration files, documentation, firmware, and other secondary items be included. Criteria should exist for definition of a ‘starting level’ and ‘end level’ of granularity, content type, and inclusion or exclusion of specific types of artifact.
- An understanding of how update and patch information is communicated to users of software applications. While commercial and contracted software suppliers know their customers and can proactively push update information to those customers, creators of open source solutions often have no knowledge of who their users are, or how those users are using the open source solution. The responsibility for obtaining updates or patches for open source components then falls on the user who is expected to actively subscribe to update information and perhaps actively tell all their internal and downstream customers to apply the patch.
- A recognition that procurement of open source components is usually as simple as downloading the component and using it – without the benefit of any security reviews, audit trails, or other source code management controls.
- A recognition that many open source software solutions are accessible in both a pre-release, or source form, and in a released form, such as with a purchased maintenance agreement, where pre-released software may have limited security testing, if any.
- A recognition that malicious actors do attempt to contribute code changes to legitimate open source projects and that, even after it is discovered, such malicious code might remain available for download in a project fork or branch even after the malicious code is removed from the primary repository.

Each of the bullets above highlights how open source software differs from commercial/proprietary or contracted software. With commercial/proprietary and contracted software, the onus for responsible development and management practices can be placed on the supplier, but that paradigm is reversed when open source software is used. Proper usage of open source software necessitates that the primary consumer and any downstream consumers of that software either implement robust controls to validate

its suitability to a specific purpose, or to contract with an entity that will proxy that responsibility and become an official supplier for specific open source components. Even when such a proxy supplier is used, there must be a process to ensure that individual components subject to that proxy relationship are not replaced with versions accessible from a public open source repository which by definition might contain different functionality or implementations.

Within the open source community, risk scoring is already in practice and could be cross-referenced by SBOM data – for example, OpenSSF's CII Best Practices badge project and others, including:

- Community Health Analytics Open Source Software (CHAOSS) that focuses on creating analytics and metrics to help define community health and identify risk
- The OpenSSF Security Metrics Project, to collect, aggregate, analyze, and communicate relevant security data about open source projects.
- The OpenSSF Security Reviews to provide a collection of security reviews of open-source software.
- The OpenSSF Security Scorecards to provide a set of automated pass/fail checks to provide a quick review of arbitrary OSS.

This information, when combined with software component inventory, forms a basis for critical risk decision making – and so it is crucial that the two are compatible and linkable. The reliability of this data is very much contingent upon intimate knowledge of the development environment where the software is developed, downloaded, stored, built, packaged, and distributed – which in the case of open source software may be neither standardized nor centrally maintained.

Referring to the original objectives of Executive Order 14028, this poses further questions for the scope and purpose of SBOM in this context:

- Should there be a separate SBOM provided that declares details of the software tool chain used to create the delivered software components to convey information about provenance and pedigree?
- How would an SBOM consumer gain complete traceability, correlating back to the original source code and contributing authors, of each of the declared software components?
- Could risk analysis frameworks encompass SBOM as part of their criteria for communicating component risk based on the presence or absence of SBOM practices or quality of SBOM data, and are there different levels of SBOM data confidence that should be considered at different levels of trust?
- What mechanisms might suffice to enable these different levels of trust, such as attestation, hashing, and signing?

Baseline data and the benefit of SBOM as the key to a database of software trust and authenticity information

The broad deployment of SBOM will provide a significant step forward in capability for organizations to make vastly improved decisions when accepting software components for deployment, as well as providing significantly higher resolution information for after-the-fact response and investigations when software vulnerabilities are reported, threats targeting specific assets, or other events occur.

To this end, a role of SBOM is that of a database key. External resources should thus be able both to cross-reference and correlate based upon identifying attributes contained within the SBOM.

The types of external data that might be referenced include information that is both public as well as proprietary or confidential in nature – for example, embargoed vulnerability reports, subscription vulnerability feeds, and emerging incident information.

In practice, what this means for SBOM implementation is:

- A minimal amount of required core data elements should be stored in the SBOM itself;
- Transient or dynamic data is linked from the SBOM to an authoritative source, with the recognition that such external data or the protection level of this data can change over time. This would be counter to the approach of attempting to catalog all of this information within the SBOM, and thus causing the requirement for the SBOM itself to be continually updated as this information changes;
- A recognition that not all information, particularly as it pertains to undisclosed threats and vulnerabilities, is suitable for broad dissemination, and that there should not be a requirement to declare information of this nature within the baseline SBOM;
- The SBOM format should permit sufficient flexibility for parties to later augment an SBOM with custom or proprietary linkage as required, as part of a post-processing or data enrichment exercise and perhaps using different fields for different customers who have different requirements;
- That there is some description of trust, authenticity, or integrity associated with the SBOM, where it is possible for a third party to quickly and independently verify without requiring access to, or a subscription with, tooling that created the SBOM or any linked authoritative sources.

The SBOM is nonetheless dynamic in nature; in so far as it should be expected that the SBOM is updated each time software is patched or re-released. Version identification fields and checksums should be sufficient to discern the differences in a similar SBOM from one software revision to the next.

Today, organizations who already produce SBOM data to inventory their open source using Synopsys and other tools, will commonly use data fields which specify:

- The package name
- Version number – as defined by the original package supplier
- License and copyright information – as asserted by the original package supplier

Upon receiving an SBOM, a recipient may wish to answer questions regarding the authenticity not just of the SBOM, but of the software declared within the SBOM:

- Is the SBOM authentic from the perspective of being issued by the claimed author, or could it have been tampered with or even be wholly fraudulent and created to match equally fraudulent software?
- Is the author – or more likely the organizational representative that ‘owns’ the automation that creates SBOMs piece by piece – of the SBOM the appropriate entity to provide an SBOM covering this particular piece of software?
- Does the software accompanying the SBOM match the inventory of components that are declared in the SBOM?
- Does the software package or final product received contain any additional, undeclared, elements compared with the SBOM?

- Assuming that all other checks succeed – in so far as the SBOM is authentic, the software matches the SBOM, and that the author is legitimate – are there any other considerations when evaluating whether a software package and SBOM are to be trusted to a level commensurate to the software's use, deployment environment, compliance requirements, and so on?

In some cases it may be desirable to consult with external resources to identify the origin or authenticity for certain components (e.g., the National Software Reference Library [NSRL]), it should also be possible to perform elementary SBOM verification based solely on information that is contained with the SBOM and using basic tooling without requiring reference to external resources, or subscriptions to external services – for use in environments where no external connectivity exists or where limited processing power is available.

Retrospective SBOMs for forensic and legacy purposes - embedded systems and SaaS services

Although there is much discussion regarding automated SBOM generation for new software components which are currently in development based upon source code, package manager data, or developer declaration during the build process, there are several scenarios whereby retrospective SBOM generation or access to historical SBOM records may be required across a large base of existing installed and legacy systems.

In this section, we discuss two examples at opposite ends of the spectrum of this requirement – embedded systems and Software-as-a-Service (SaaS) services.

For existing deployed software – either standalone applications, or software which is embedded within products and systems – a number of unique challenges exist. For example, this type of requirement could encompass the software elements within a broad range of everyday consumer devices, including but not limited to medical devices, ATMs, and automobiles, through to power plants, industrial control systems and their supporting operational technology (OT) infrastructure – including associated servers, workstations, and newly introduced connectivity gateways that bridge these to enterprise IT networks and cloud services.

As another example, many of these types of systems have characteristics which distinguish them from ordinary web, desktop PC, or mobile applications. The systems which this software is contained within may have been originally manufactured many years prior, may no longer be maintained by the original supplier (or the original supplier entity may no longer exist), and were designed to have an operational lifespan measured in decades. Gaining service access to either obtain or update the software itself may invoke an extensive change management regimen and in some cases require plant downtime or invalidate certifications, incurring penalties for service outage or a lengthy recertification process.

While each of these examples related to industries where commercial or contracted software is common, that software likely contains open source componentry or libraries from other commercial suppliers. Technology exists today to facilitate the creation of SBOMs for a wide range of binary-only software, which may permit generation of a limited SBOM in some of these legacy use cases. The input to this process could be a firmware image for an embedded microcontroller or IoT device, or a complete desktop workstation installation package.

This binary analysis method of SBOM creation does however raise a number of both technical and non-technical considerations:

- For embedded devices which are no longer supported – how should access to the software be facilitated so that SBOM generation can be performed? Should this be for a ‘live’ as-running device, or is a specimen SBOM from a similar model or family of device sufficient?
- For products under a sustaining or maintenance only contract – which party should bear the cost of SBOM creation and how will this be funded? Where a consumer of the product elects to create an SBOM, if that SBOM identifies unpatched open source components or libraries, does that create an obligation for the supplier to update the product?
- What level of investigation should be performed for ‘unidentified’ or ‘unidentifiable’ components, and who should perform this?
- Would generation of SBOM data from a software binary by a third party constitute the practice of ‘reverse engineering’, and would this create a breach of a common condition in many end user license agreements? Should contracting parties request a dispensation of this condition for the special case of SBOM creation?
- The consequent cost impact of vulnerability response based on inaccurate software identification may be prohibitive – an unnecessary update requiring recall or recertification of a device could cause this for example. How can organizations best obtain maximum confidence in accurate identification of vulnerable software components before making the decision to patch?

Beyond embedded device software, the requirement for retrospective SBOM data may also occur for software delivered via publicly hosted SaaS services. In this scenario, a consumer of a SaaS service may

in some circumstances need to request that the service provider provide an SBOM retrospectively as part of an investigation or vulnerability response process, perhaps for a system that existed years ago for a total of a few minutes.

Operational considerations for retrospective SaaS SBOM provision might include:

- For how long should service providers be reasonably expected to retain SBOM records, and at what granularity?
- Under a 'shared responsibility' model for security management commonly adopted amongst cloud service providers, should the SBOM requirement thus cascade down to underlying infrastructure, and even compute hardware providers?
- For frequently-changing software as commonly produced by teams practicing DevOps and Continuous Integration/Continuous Delivery or Deployment (so-called 'CI/CD' workflows), in some cases updated multiple times during a single day, how should a service provider best capture SBOM data for small increments of change?
- In the case of a microservices architecture, whereby functionality is broken down into many smaller blocks of software components each with their own lifecycle, is there a requirement for a system-wide SBOM which should inventory not only the software code components, but also configuration and orchestration information required to compose the complete system, which is often also described in code? (so-called 'infrastructure as code')?
- For the use of hybrid and cloud native applications that may leverage architectural paradigms such as Function-as-a-Service ('FaaS'), or other references to ephemeral external resources – how should these best be described via SBOM?
- For SaaS services that utilize 'A/B' testing – whereby different users' inbound traffic is routed to effectively different variants of underlying application services – how could an SBOM distinguish between which users or tenants in a SaaS service were serviced by which software components – given that some instances of a particular service may utilize a vulnerable component and some may not?
- For applications created in low-code and no-code environments, such as some developed and fielded in an SAP or Salesforce technology stack, what goes into the no-code supplier's SBOM and how is that tied, and to what time granularity, to the larger supplier's SBOM?

Critical factors in successful and widespread SBOM adoption

It is likely that SBOM will mature through several phases – the initial 'start up' period, a second 'interoperability' phase, and then toward highly automated 'self-driving SBOM' – whereby SBOM information is automatically generated, processed, and the capability exists for software acceptance decisions on the recipient side to be made without human intervention against a prescribed machine-readable policy based on risk profile as soon new software component deliveries occur (e.g., as part of a governance-as-code structure). This trajectory may occur at a different pace in different sectors, and could be a slow linear progression, or rapid parallel iterations depending on the nature and criticality of the systems in scope. The establishment of a maturity plan that is clearly defined and achievable for providers and consumers is key.

Focusing on the initial two phases – a number of technical requirements must be addressed to provide unambiguous guidance for those seeking to get started with SBOM implementation. This will form the foundation for SBOM creation, transmission, consumption and maintenance, and provide a future framework for decision support.

- Data file formats:
    - A number of formats exist for SBOM encapsulated as highlighted by NTIA, however each of these differ in terms of their maturity, accessibility, scope, and level of support
    - The attributes of an ideal format would include:
        - Mature, well-established, across many industry segments
        - Freely available specification for implementors without license constraints
        - Sufficiently extensible to support augmentation with additional data and linkage as required to support derivative use-cases
        - Broad and well-tested support amongst software tools and libraries in multiple programming languages
- Flexibility in software identity and identifiers:
    - Since software can be identified by many different attributes, it is important that the SBOM framework permits a broad and flexible range of data points to facilitate reliable identification
    - This should encompass the range of both open source, packaged, un-packaged, and commercial software distribution methods, and also consider the points highlighted above regarding forking and derivative works – either genuine or maliciously misrepresented – and how these might be discerned as being of distinctly different lineage from the original authoritative source of the package
    - Attempting to establish new conventions in this area should be avoided, instead focusing on providing enough meaningful data for an SBOM consumer to make their own determination as to the identity of the software component being described by the SBOM
- Validated tool inter-compatibility for base data elements:
    - Although numerous tools exist across the gamut of SBOM use-cases, and certainly more will emerge in future, it is crucial that end-users as well as tool providers are able to validate their inter-compatibility
    - Without certainty when selecting or deploying an SBOM tool, end users may hesitate or delay their decision to begin adoption of SBOM
    - Furthermore, unless compliance can be empirically evaluated, tool providers and prospective SBOM tool providers may be reluctant to invest in building on top of the foundation that SBOM provides, or may splinter off into proprietary SBOM ecosystems creating fragmentation and impeding adoption

- It is therefore critical that a mechanism exists to confirm that a particular tool is capable of producing or processing SBOMs with both the required data content as well as formatting and structure

Once these conventions are established, the secondary challenge is for those seeking to operationalize SBOM. In this regard, what is required is clear directional guidance on a number of commonly encountered topic areas including, but not limited to:

- Conventions for producing and interpreting SBOM data for each of the different purposes or scenarios – in particular the legacy and forensic use-cases described above, as well as SBOM integrity and how to validate and verify integrity and authenticity of the SBOM itself
- Best practices for categorizing different levels of SBOM quality and resolution – especially when incomplete or uncertain information is available. Is a poor quality SBOM with known missing information better than no SBOM at all?
- Clarification on topics such as whether generation of SBOM data from a pre-existing software package binary may violate commonly adopted commercial software license covenants with regard to reverse engineering
- Reference architectures for SBOM sharing and distribution – considering a variety of scenarios, including but not limited to: point-to-point explicit sharing, continuous and automated sharing on a push/pull basis, and public/private distribution models. Consideration should also be given to the range of different roles, permissions, and access rights in the sharing architecture.

Finally, to illustrate the full benefit of complete SBOM automation, there is a need to provide a catalog of complete end-to-end user stories that describe workflows for SBOM from software creation, to deployment, to subsequent events such as security vulnerability, license enquiry, or incident response and investigation.

These would in turn provide a valuable educational resource for prospective SBOM adopters, should be tailored for several different levels of audience:

- Executive leadership, on the obligations, roles and responsibilities of the organization with respect to SBOM, and how it may benefit the adoption of secure development practices for both inbound and outbound software packages
- Compliance and management teams, tasked with balancing SBOM requirements against other market demands and resource priorities
- Implementors and end-users, requiring hands-on knowledge on how best to realize the goals and benefits of SBOM in a way which is minimally disruptive to day to day activities

This guidance should address the question of ‘how’ an organization may determine the scope and requirement for human resources and new processes to introduce and maintain an SBOM program.