Securing the Software Supply Chain: Recommended Practices for Managing Open-Source Software and Software Bill of Materials

Enduring Security Framework

December 2023

Executive Summary

Cyberattacks target an enterprise's use of cyberspace to disrupt, disable, destroy, or maliciously control a computing environment or infrastructure, destroy the integrity of the data, or steal controlled information.[1]

Cyberattacks such as those executed against SolarWinds and its customers—and the exploits that take advantage of vulnerabilities such as Log4j—highlight weaknesses within software supply chains. This issue spans both commercial and open-source software and impacts private and government enterprises. Accordingly, there is an increased need for software supply chain security awareness and cognizance regarding the potential for software supply chains to be weaponized by nation-state adversaries using similar tactics, techniques, and procedures (TTPs).

In response, the White House released an Executive Order on Improving the Nation's Cybersecurity (EO 14028)[2] that established new requirements to secure the federal government's software supply chain. The Enduring Security Framework (ESF) [3], led by a collaborative partnership across private industry, academia and government, established the Software Supply Chain Working Panel, which released a three-part *Recommended Practices Guide* series to serve as a compendium of suggested practices to help ensure a more secure software supply chain for developers, suppliers, and customer stakeholders.

Similarly, the ESF Software Supply Chain Working Panel established this second phase of guidance to provide further details for several of the *Phase I Recommended Practices Guide* activities. This guidance may be used to describe, assess, and measure security practices relative to the software lifecycle. Additionally, the suggested practices listed herein may be applied across a software supply chain's acquisition, deployment, and operational phases.

The software supplier is responsible for liaising between the customer and software developer. Accordingly, vendor responsibilities include ensuring the integrity and security of software via contractual agreements, software releases and updates, notifications, and the mitigation of vulnerabilities. This guidance contains recommended best practices and standards to aid customers in these tasks.

This document aligns with industry best practices and principles that software developers and software suppliers can reference. These principles include managing open-source software and software bills of materials to maintain and provide awareness about software security.

[1] Committee on National Security Systems (CNSS)

[2] https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[3] ESF is a cross-sector working group that operates under the auspices of Critical Infrastructure Partnership Advisory Council (CIPAC) to address threats and risks to the security and stability of U.S. national security systems. It is comprised of experts from the U.S. government as well as representatives from the Information Technology, Communications, and the Defense Industrial Base sectors. The ESF is charged with bringing together representatives from private and public sectors to work on intelligence-driven, shared cybersecurity challenges.

DISCLAIMER

DISCLAIMER OF ENDORSEMENT

PURPOSE

The National Security Agency (NSA), the Office of the Director of National Intelligence (ODNI), and the Cybersecurity and Infrastructure Security Agency (CISA) developed this document in furtherance of their respective cybersecurity missions, including their responsibility to develop and issue cybersecurity recommendations and mitigation strategies. This information may be shared broadly to reach all appropriate stakeholders.

CONTACT

Client Requirements / Inquiries: Enduring Security Framework nsaesf@cyber.nsa.gov

Media Inquiries / Press Desk:

- NSA Media Relations, 443-634-0721, MediaRelations@nsa.gov
- ODNI Media Relations, dni-media@dni.gov
- CISA Media Relations, 703-235-2010, CISAMedia@cisa.dhs.gov

Table of Contents

1 Introduction

Unmitigated vulnerabilities in the software supply chain continue to pose a significant risk to organizations and our nation. This paper builds on the previously released Recommended Practices Guide for a software supply chain's development, production and distribution, and management processes, to further increase the resiliency of these processes against compromise. This guidance also builds on and supports the Office of Management and Budget memorandum on *Enhancing the Security of the Software Supply Chain through Secure Software Development Practices* (M-23-16)[4].

All organizations, whether they are a single developer or a large industry company, have an ongoing responsibility to maintain software supply chain security practices in order to mitigate risks, but the organization's role as a developer, supplier or customer of software in the software supply chain lifecycle will continue to determine the shape and scope of this responsibility. The information contained in this guidance supports development activities of a single developer as well as activities of large industry companies. Activities should be planned for and acted upon one at a time, solidifying the new technique in the process before adding the next to be successful.

Because the considerations for securing the software supply chain vary, this document which focuses on the management of "*Open-Source Software (OSS) and Software Bill of Materials (SBOMs)*" will help continue to foster communication between the different roles and among cybersecurity professionals that may facilitate increased resiliency and security in the software supply chain process.

Organizations that include OSS in the development of their products are encouraged to proactively manage OSS risks as a part of evolving secure software development practices. It is recommended that software development and supplier organizations read and implement the strategies described here. Recent high profile software supply chain incidents have prompted acquisition organizations to assign supply chain risk assessments to their buying decisions. Software developers and suppliers should improve their processes, and reduce the risk of harm, not just to employees and shareholders, but also to those affected by the use of their software.

To help achieve this, this document recommends seven areas of improvement related to software development and OSS. These areas are designed to allow an organization to mature their software development process and although there are many tools that can be used, no tool will be promoted over another. The seven areas are:

- Open-Source Selection Criteria,
- Risk assessment,
- Licensing,
- Export control,
- Maintenance,
- Vulnerability response, and
- Secure Software and SBOM Delivery.

[4] https://www.whitehouse.gov/wp-content/uploads/2023/06/M-23-16-Update-to-M-22-18-Enhancing-Software-Security.pdf

1.1 Background

Common methods of compromise used against software supply chains continue to include exploitation of software design flaws, incorporation of vulnerable third-party components into a software product, infiltration of the supplier's software development lifecycle with malicious code prior to the final software product being delivered, and injection of malicious software that is built and then deployed by the customer.

Stakeholders should continually mitigate security concerns specific to their area of responsibility. However, other concerns may require a mitigation approach that dictates a dependency on another stakeholder or a shared responsibility by multiple stakeholders. Dependencies that are inadequately communicated or addressed may lead to vulnerabilities and the potential for compromise. Transparency into the software supply chain is necessary to manage that risk.

1.2 Document Overview

The four sections of this document and the associated activities of the Secure Software Development Framework (SSDF)[5] they implement are identified in Table 1 below.

Table 1: Associated SSDF Activities

Section	SSDF Activity(ies) Implemented
2. Open-Source Software Management	• Prepare the Organization (PO)
3. Creating and Maintaining a Company Internal Secure Open-source Repository	• Protect the Software (PS) • Produce Well-Secured Software (PW) • Respond to Vulnerabilities (RV)
4. Maintenance, Support and Crisis Management	• Protect the Software (PS) • Respond to Vulnerabilities (RV)
5. SBOM Creation, Validation and Artifacts	• Protect the Software (PS) • Produce Well-Secured Software (PW) • Respond to Vulnerabilities (RV)

The guidelines and specifications identified within this document are evolving, refer to the following resources for the latest recommendations and updates:

- Cybersecurity and Infrastructure Security Agency (CISA) Software Bill of Materials[6]

[5] https://csrc.nist.gov/Projects/ssdf

[6] CISA, Software Bill of Materials, https://www.cisa.gov/SBOM

This document also contains the following appendices:

Appendix A: Ongoing Efforts

Appendix B: Secure Supply Chain Consumption Framework (S2C2F)[7]

Appendix C: References

Appendix D: Acronym List

2 Open-Source Software Management

This document is a continuation of the work released in "*Securing the Software Supply Chain, Recommended Practices Guide For Developers*"[8] and "*Securing the Software Supply Chain Recommended Practices Guide For Suppliers.*[9] The previous work included an examination of how OSS is incorporated into the development, build and release environments. In this work, we go into more detail on OSS adoption and the things to consider when evaluating and deploying an open-source component into an existing product development environment. OSS components may have downstream dependencies that contain embedded vulnerabilities. Therefore, we pay particularly close attention to how these modules are used and bundled with the software at release. This section describes the overall OSS acceptance process, to include its composition, the process and procedures used when adopting open-source software, and the management, tracking and distribution of approved software components using an SBOM. The roles of the developer and supplier are defined as:

- **Developer** - The developer, an employee of the supplier, is the originator of the source code for a product who identifies the need for OSS and/or third-party components to meet the specific need of a product. Once identified, they obtain the OSS, check for license and vulnerability issues, integrate it into the product, and create an SBOM.
- **Supplier** - The supplier is the vendor of a software product or library. They validate that the product, as developed, meets all development requirements, as well as licensing, export control and vulnerability assessments guidelines defined as shipping criteria for use of the product.
- **Both Developer and Supplier -** In small organizations, these tasks may be performed by the same team.

[7] https://github.com/ossf/s2c2f

[8] https://media.defense.gov/2022/Sep/01/2003068942/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF

[9] https://media.defense.gov/2022/Oct/31/2003105368/-1/-1/0/SECURING_THE_SOFTWARE_SUPPLY_CHAIN_SUPPLIERS.PDF

Recommended activities for open-source adoption by developers and suppliers include:

- Developer
    - Identify potential OSS solutions for consideration.
    - Create an internal secure repository which is maintained using the guidelines in section 4.1 “*Maintain Open-Source Software*.”
    - Integrate OSS into the secure build process of the product using the same guidelines as with the in-house developed components.
    - Track updates to OSS or third-party components.
    - Produce updates of the product to specifically address changes to the OSS.
- Supplier
    - Monitor for license change issues and vulnerabilities of the OSS included in any third-party software.
    - Manage updates of the product that specifically address changes to the OSS. The use of an SBOM tracking mechanism is strongly recommended to aid in ensuring the ongoing integrity of the product.

2.1 Primary Considerations with the use of Open-Source Software

The primary considerations with the use of open source in a product or service offering are Selection, Risk assessment, Licensing, Export control, Maintenance, Vulnerability response and Secure Software Delivery. Even before the adoption process outlined in section 3.1 “Open-Source Software Adoption Process” is initiated, the software should first be evaluated using precursory analysis such as the use of the National Vulnerability Database (NVD) to determine whether the software should be considered for selection. Once selected, additional analysis as defined in section 3.2 “Vulnerability and Risk Assessment” is used to fully understand the risk associated with the software. If acceptable, the software is integrated within the development process defined in “*Creating and Maintaining a Company Internal Secure Open-Source Repository*. When products are released, they follow the guidelines for maintenance and vulnerability response described in section 4 “*Maintenance, Support and Crisis Management*.” The process to securely deliver software and SBOMs is outlined in section 5 "*SBOM Creation, Validation and Artifacts.*" Additional considerations for licensing and export control are covered in section 2.2 “*Licensing*” and section 2.3 “*Export Controls*.”

2.2 Licensing

Licensing considerations should be applied when considering OSS for adoption. An OSS license governs the use, modification, and distribution of open source. OSS licenses can impose obligations and constraints which may have an impact on software distribution.

Suppliers, preferably working with legal assistance, determine and make known to their developers, any restrictions of use, and provide the language that should be displayed and agreed to by the customer obtaining the product.

Scanning with a tool designed to identify open source within a product is useful, however the best practice is to independently track the OSS used in your products (with the applicable OSS name, version, and download location). OSS in third-party software used in your products should be scanned[10] and approved. If OSS incorporates other OSS (which is sometimes called a dependency or transitive dependency), this incorporated OSS should also be scanned and approved.

2.2.1 License Compliance

Developers are expected to be aware of and adhere to OSS license requirements[11] such as stipulations for the use of credit banners and the presentation of the acceptance of usage during initial installation of the product. Developers and suppliers should also ensure that:

- Their organization has the necessary legal rights to use the OSS you select,
- Their use of OSS may not taint or encumber their proprietary code with code sharing obligations or otherwise negatively affect intellectual property rights[12]; and
- They have read and agree to comply with the associated license policy as well as the terms of the licenses for all OSS you use.

While this may be a lot for individual programmers to track, organizations can provide tools to make this consideration easy or transparent for the humans at keyboards. The Open-Source Initiative[13] provides detailed information about the various license types and associated usage conditions (for tools that can help scan for license compliance see section 5.1.4 "License and Export Control" for further information).

2.3 Export Controls

Some countries have export regulations that may require anyone incorporating open-source content into their products ensure the included open-source project meets those regulations. In the US, they are the Export Administration Regulations[14] (EAR). The European Union[15] (EU) and other jurisdictions have a similar set of regulations.

Legal guidance for export control concerns is necessary to include in your OSS process. However, it should be noted that anything, including an OSS item, may be added to the EAR's Commerce Control List[16] (CCL) at any time regardless of the terms of a license agreement between commercial parties or if someone posts it online with an open-source agreement. Thus, it will always be prudent to verify a specific package is not on the CCL.

In summary, for many of those in the US who wish to include open source software in a product or service should (1) take steps to ensure that the open source software is indeed publicly available without restriction and, (2) if it includes non-standard encryption or is related to neural computing,

[10] See section 5.1.3 "Software Composition Analysis and the VEX Format of this document.

[11] https://opensource.org/osd/

[12] https://opensource.org/licenses/review-process/

[13] https://opensource.org/licenses

[14] https://www.bis.doc.gov/index.php/regulations/export-administration-regulations-ear

[15] https://www.ecfr.gov/cgi-bin/text-idx?node=pt15.2.734&rgn=div5#se15.2.734_12

[16] https://www.bis.doc.gov/index.php/documents/regulations-docs/2329-commerce-control-list-index-3/file

advise the government as directed in the EAR. Those in other countries should consult with their government.

When adopting OSS, developers should extract export-related information such as cryptographic algorithms used in the OSS and any other cryptographic dependencies the OSS requires. During the development of the product, developers may determine the best way to adhere to the export requirements defined by policies set forth by suppliers and may determine that a second, distinct product having a subset of capabilities may be required as the final deliverable for some customers. Developers may also be required to make known the use of Personally Identifiable Information (PII) that may be used within the OSS.

Suppliers may provide export guidance to include implementation criteria of the product under development. Once packaged, suppliers validate the process and policies defined have been adhered to during the development of the product. Suppliers may use automated tools to perform product package export validation, and the validation process may vary depending on where the product is being sold and used. Suppliers understand where and when export controls need to be considered and handle the distribution of the product based on those criteria.

2.4 Software Bill of Materials Overview

A SBOM is used to define all aspects of a product to include open source and commercial third-party software. SBOMs often include licensing data for components. There are two primary widely used data formats that express the syntax of an SBOM:

- SPDX[17] is "an open standard for communicating software bill of material information, including components, licenses, copyrights, and security references." It originated with the Linux Foundation and is an international open standard (International Organization for Standardization/International Electrotechnical Commission (ISO/IEC 5962:2021[18]).
- CycloneDX[19] "is a full-stack SBOM standard designed for use in application security contexts and supply chain component analysis." It originated within the Open Web Application Security Project (OWASP)[20] community. CycloneDX has expanded to include a wide range of other, related use cases, including software-as-a-service BOM (SaaSBOM)[21]

Software Identification (SWID) Tagging[22] is an international standard [ISO/IEC 19770-2:2015[23]] that originated from the National Institute of Standards and Technology (NIST).

[17] https://spdx.dev/

[18] https://www.iso.org/standard/81870.html

[19] https://www.cyclonedx.org/

[20] https://www.owasp.org/

[21] https://cyclonedx.org/capabilities/saasbom/

[22] NIST Software Identification (SWID) Tagging, https://csrc.nist.gov/projects/Software-Identification-SWID

[23] https://www.iso.org/cms/%20render/live/en/sites/isoorg/contents/data/standard/06/56/65666.html

It provides descriptive information about a specific release of a software product or component but currently does not provide a dependency graph[24]. SWID tags may be incorporated into both SPDX and CyloneDX SBOM documents to allow an easy transition between formats.

Care should be taken to ensure SBOMs are provided in a format that can be processed by their consumers without the loss of integrity and that the generated SBOM meets the minimum element requirements documented in the 2021 National Telecommunications and Information Administration (NTIA) "*The Minimum Elements For a Software Bill of Materials* (SBOM).[25]

While translation tools are available to convert between formats, digitally signed component documents that are transformed outside the boundaries of the supplier may lose the proof of authenticity provided by the originating author. For information on how licensing and export control information is created and shared, refer to section 5.1.4 "*License and Export Control."*

3 Creating and Maintaining a Company Internal Secure Open-Source Repository

An internal repository can help automate key processes around OSS usage, including security testing, policy enforcement, integrity verification, and auditing. This section describes the process used to create and maintain open-source software that has been approved for use within a company. It describes the mechanisms used to create an internal secure repository which is made available to multiple product development groups/organizations and how this repository and the third-party components are shared, maintained and continually checked for vulnerabilities (see Figure 1).

[24] See David Waltermire et al., Guidelines for the Creation of Interoperable Software Identification (SWID) Tags (2016) (NIST Internal Report 8060), http://dx.doi.org/10.6028/NIST.IR.8060

[25] NTIA The Minimum Elements For a Software Bill of Materials (SBOM) https://www.ntia.doc.gov/sites/default/files/publications/sbom_minimum_elements_report_0.pdf; https://www.ntia.doc.gov/report/2021/minimum-elements-software-bill-materials-sbom

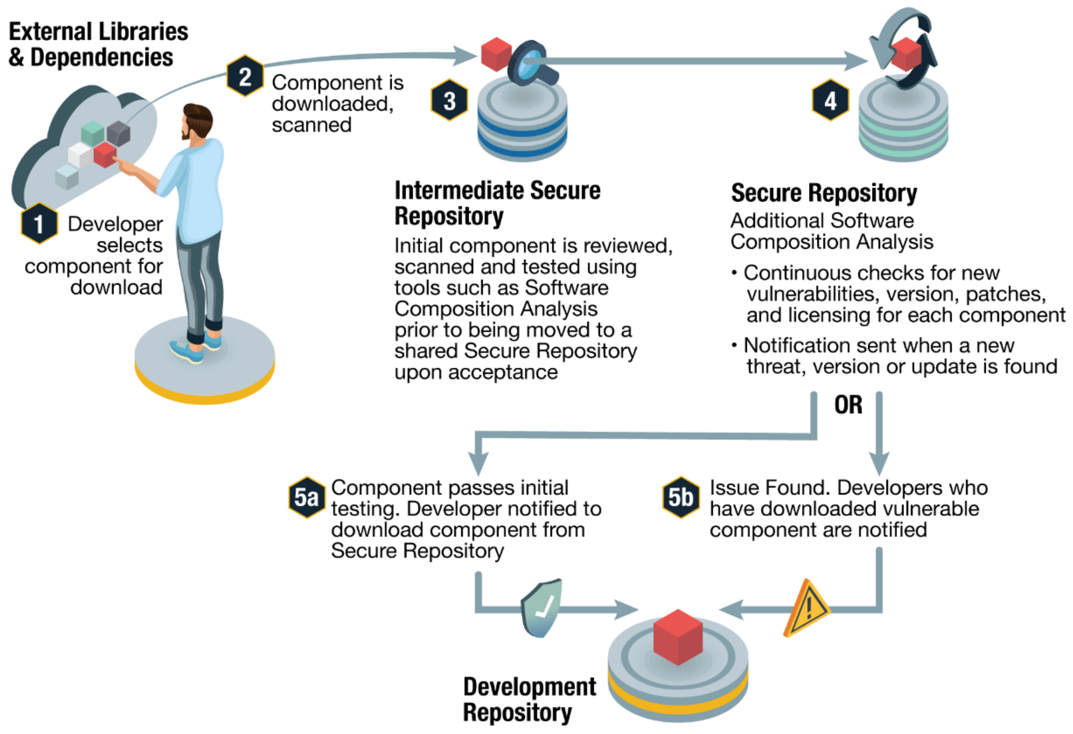

External Libraries & Dependencies
1 Developer selects component for download
2 Component is downloaded, scanned
3
Intermediate Secure Repository
Initial component is reviewed, scanned and tested using tools such as Software Composition Analysis prior to being moved to a shared Secure Repository upon acceptance
4
Secure Repository
Additional Software Composition Analysis
• Continuous checks for new vulnerabilities, version, patches, and licensing for each component
• Notification sent when a new threat, version or update is found
OR
5a Component passes initial testing. Developer notified to download component from Secure Repository
5b Issue Found. Developers who have downloaded vulnerable component are notified
Development Repository

Figure 1: Maintaining Secure Open-Source Repository

Organizations may need to select, acquire, and deploy package repository software or services to host their internal secure open-source package repository such as GitHub Packages, jFrog Artifactory and Sonatype Nexus Repository. They may already have one if they are building and publishing packages internally, and if so, they should evaluate whether this existing software or service can also meet their needs for open-source software. The main factors in selecting a package repository may be support for the types of open-source packages used by the organization (such as Maven, Node Package Manager (npm), or Docker images), as well as specific features desired by the organization (such as integration with the organization's Identity and Access Management (IAM) systems).

When implementing a package repository solution for an organization to use in open-source management, it is critical to properly define and enforce the processes for adding packages and consuming them. For organizations with extensive open-source use, these processes can have a significant impact on agility and developer satisfaction. Choosing the appropriate level of assessment for each stage of development and automating these processes can minimize this impact.

To ensure that developers can confidently consume open-source from the package repositories, appropriate controls should be put in place so packages cannot be added outside of the approved processes. These controls may include access control restrictions or policies that prevent the consumption of packages that don't meet certain criteria. To balance developer agility with risk, organizations may use multiple package repositories with differing policies. For example, one that can be used from developer local workstations and continuous integration (CI) systems (with less restrictions), and another used for more restrictive build systems used for product released (with more restrictions). These mitigations are aligned with emerging industry frameworks such as the Secure Supply Chain Consumption Framework (S2C2F)[26]. The S2C2F provides high-level practices and detailed requirements to improve how developers securely consume open-source components and organizes them into a maturity model to enable development teams and organizations to prioritize effectively (see Appendix B: Secure Supply Chain Consumption Framework (S2C2F)).

3.1 Open-Source Software Adoption Process

There are various levels of maturity of an Open-Source Software management process. For smaller organizations, the process may involve the management of a single repository where adopted third-party software is integrated after passing all risk and vulnerability assessment that may be performed manually or with tool support. The first step in the open-source adoption process (see sections 2.2 and 2.3 of the *Securing the Software Supply Chain for Developers*[27]) is the identification by the developer for the need of a specific open-source component based on product and design requirements. The adoption takes into consideration the quality of the open-source component, its adoption by others, license type, vulnerability history and the benefits of the adoption as related to time and development cost. The developer determines the delivery format of the component based on formats available, binary or source and the ability to incorporate the OSS available into the secure build environment. Source is preferred for better integration into the secure build practices of the product into which it is being adopted. The developer performs the initial vulnerability assessment by first running any security analysis tools that are available prior to download, such as Software Composition Analysis (SCA), virus scans and fuzz testing. Developers then download the component to an isolated secure environment where additional composition and security analysis is performed (refer to section 3.2 and section 5.2.1 on how to perform this analysis). Based on the size and structure of the organization, the results of the OSS vulnerability scan are provided to the suppliers and developers for further review if these groups don't already have access to the results. A Vulnerability Exploitability eXchange (VEX)[28] document associated with the software may also be an important input into the decision process. During this process, the developer also evaluates the component under consideration to ensure it provides the desired features while maintaining security and weighs the cost of integration. Once the initial evaluation is performed and the decision is made to move forward with the adoption process, larger organizations may require a formal request be generated to the development management team to complete the approval process.

For both large and small organizations, once approved, an ingestion process allows the developer to upload all required materials to a secure, protected environment, with the component being stored in an intermediate secured repository. The documents collected outline the requirements met by the component adopted, as well as artifacts that may have been obtained that describe associated information on security analysis results, risk, licensing, and export considerations.

[26] https://github.com/ossf/s2c2f

[27] https://media.defense.gov/2022/Sep/01/2003068942/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF

[28] VEX is a type of assertion allowing a supplier or other party to claim that a vulnerability does not affect a piece of software, and that the user or downstream developer does not have to take any action. It is up to the developer in this instance to determine whether to trust the VEX statement.

In large organizations, an Open-Source Review Board (OSRB) reviews all adoption requests using a team consisting of representatives from development, management, security and quality assurance teams. The team performs a security assessment identifying any known Common Vulnerabilities and Exposures (CVE) associated with the component and augments the security scanning previously performed by the developer using tools not easily available to developers due to cost or other reasons. The team also evaluates the licensing and export requirements of the component and identifies any tasks that may be required to meet those defined policies and procedures. The evaluation considers the history of the component, as it relates to any previous version, noting how the project is currently maintained and has been maintained over time. Third party analyses of the component or project, such as Open-Source Security Foundation (OpenSSF) scorecard[29] can be considered as a part of the evaluation process, requiring the developer to verify the results and assess risk. (Refer to "*Quickly Assess Open-Source Projects for Risky Practices*.")

The review process can be iterative where the OSRB team may need additional information from the developer and both the request for information and response are captured and provided as artifacts used in the final decision. Once all information is collected, the OSRB team performs risk assessment and determines an outcome for the request. The decision considers the security scanning results and may outline any exceptions that have been granted to the component, such as a known vulnerability in the component which may not be affected in the adopted product. Smaller organizations should perform a subset of the OSRB activities based on need and organization structure.

Once adopted, the component is integrated into a protected, read-only repository that is continuously scanned and monitored for vulnerabilities, with incidents reported directly to the developer groups that have adopted the component for use (see section 3.2 "*Vulnerability and Risk Assessment*").

In mature development environments, once a third-party component is adopted, it is integrated into the build process of the product, and the source or binary is pulled from the centralized, secure repository, allowing multiple products the ability to use the same vetted component for all builds. The build process may be enhanced to ensure the component is accounted for in automated vulnerability scanning using a more sophisticated set of tools not available within the day-to-day developer environment and also when generating a final SBOM.

Suppliers oversee the OSRB process and define the risk management process that includes the procedure for third-party software adoption, artifacts required for collection, the types of tools, output and formats required for SCA validation, vulnerability scanning and SBOM creation. Suppliers also define the tracking, vulnerability assessment and reporting mechanism required for both internal developers and external customers.

Suppliers collect and escrow artifacts and make selected artifacts available to customers, based on legal and security sensitive considerations. When possible, documents and artifacts related to the build process are rolled up into an SBOM describing the product and third-party components that reside within it. An SBOM generally will meet many of the requirements needed by customers of the product for risk assessment, validation and inventory. Suppliers and developers may maintain a list of third-party providers based on the evaluation process identified above and used as part of their adoption strategy. An exception process may be used to identify the third-party components which

[29] https://securityscorecards.dev/

may contain an associated risk based on security, licensing, exporting or source modification concerns. Suppliers and developers work together to manage the response to customer requests for the identification of third-party vulnerabilities within a product. Once the vulnerability has been resolved, suppliers and developers manage the availability of any updated components to the customer using a multitude of delivery mechanisms such as automatic live update, using a patch process or by providing a complete product update. The update mechanism must provide flexibility to allow customers the ability to work around the constraints of their specific deployed environments and internal update process when deploying a vulnerability resolution. They may do so using a process which may be facilitated by creating a VEX[30] report which the customer can ingest and use for tracking. A VEX document is a machine-readable security advisory in a format like the Common Security Advisory Framework (CSAF), with the notable feature that it can communicate that a vulnerability does *not* affect a product.

> NOTE: The adoption and production of VEX is an emerging framework and ecosystem as of the publication of this document. Developers and Suppliers of software should be aware that the production and maintenance of VEX documents[31] are still under development and need to monitor the CISA resource website for the latest information on VEX.

A better level of maturity automates the ingestion process used for the assessment and generation of artifacts used in the review for adoption. Once adopted, the component is stored in a secure repository where both vulnerability scanning and monitoring is regularly performed using a mix of both manual and automatic means. Procedures for best practices within development environments support automation and artifact generation to attest to the secure development of the final product. This attestation includes third-party adoption, the building, scanning, and packaging within the product. Once delivered to the customer, maintenance and response to vulnerabilities are managed and addressed. For more information on this process and the acceptance criteria for secure software development, refer to "*Securing the Software Supply Chain: Recommended Practices Guide For Developers*"[32] section 2.1, "*Secure Product Criteria and Management*," section 2.2, "*Develop Secure Code*," and section 2.3 "*Verify Third-Party Components*."

3.2 Vulnerability and Risk Assessment

This section describes the vulnerability and risk assessments that may be applied when considering open-source software before and after adoption. The process should include identification, provenance, and proposed use.

Note: The guidance for assessing risk of open-source components should be scaled based on the size of the development organization. At a minimum, developers need to perform a security assessment of software using a measurable technique that suites their environment. There are tools available to aid in this process, such as Security Scorecards, available from OpenSSF[33]. This is an

[30] https://www.cisa.gov/resources-tools/resources/minimum-requirements-vulnerability-exploitability-exchange-vex

[31] https://www.cisa.gov/sites/default/files/publications/VEX_Use_Cases_Aprill2022.pdf ; https://www.cisa.gov/sites/default/files/publications/VEX_Status_Justification_Jun22.pdf

[32] https://media.defense.gov/2022/Sep/01/2003068942/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF

[33] https://github.com/ossf/scorecard

automated system which analyzes thousands of open-source projects for conformance to a number of security practices. Open-source components which are chosen despite having a poor score should be scrutinized more carefully. In some cases, developers might consider alternatives to components with a poor score. As security issues are discovered and the adopted open-source projects are updated, developers should be vigorous in their efforts to adopt versions with security bugs fixed.

The first step in vulnerability and risk assessment is for developers to create an inventory of third-party open-source components (see section 3.1 "*Open-Source Software Adoption Process*"). Once an inventory is available, developers collect the versions of those components and verify they are up to date or, at least, have no known vulnerabilities that affect the component. Developers and/or suppliers identify any vulnerabilities within each component by initially and periodically checking for known CVEs and vulnerabilities using resources such as the National Vulnerability Database (NVD) or other community health facilities. Also addressed are third-party integration concerns such as built-in extensions for plug-ins required in the development of the component and the code interfaces used. Components are ranked based on relevant factors, such as the popularity and utility of the component, both internally and externally. Components can also be prioritized based on risk, security sensitivity, the use of encryption and community health. Maturity of the community, number of contributors, frequency of patching and the presence of an SBOM should also be considered. Code size and complexity are also a major factors. The language used to develop the software should be considered, for example selecting memory safe languages[34]. Libraries and components written in memory safe languages may reduce the risk of vulnerabilities present for classes of vulnerabilities such as buffer-overflows and memory corruption exploits. Depending on the overall assessment results from the considerations above, additional actions may be required, such as a manual review of some components, or the in-depth review of the results from automated scanners which report multiple levels of detections.

For each critical component, developers and/or suppliers apply security and threat modeling to identify any vulnerabilities and weakness in these components and their 3rd party dependencies. This process should be ongoing based on the risk assessment as discussed in section 4.1 "*Maintaining Open-Source Software*." A manual full end-to-end review can also be used. This process may isolate run time dependencies, within the parameters of how the component is used, for each third-party component and uses SCAs to identify all key aspects of the component. Each critical component is checked initially and periodically for community health and weaknesses.

Components with known vulnerabilities can check whether the vulnerable portion may be used/called or enabled within the application. If it is, then check for any compensating controls. Based on risk, additional vulnerability assessment may be applied by performing code reviews, additional static code analysis, dynamic code analysis and additional security analysis using in-house red teams, bug bounties or other third-party vulnerability detection resources. For more resources to support third-party vulnerability detection, refer to section 5.1.3 "*Software Composition Analysis and the VEX Format*."

All newly discovered vulnerabilities in the third-party component should be reported to all affected and tracked using the company bug tracking mechanism, as well as the third-party reporting

[34] https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

system, such as OSVs[35] and/or CVEs should be registered. Developers should work with the maintainers or internal stakeholders to prioritize vulnerabilities based on risk and schedule the availability of patches based on this assessment. For unpatched components, they identify the risk based on compensating controls and context of deployment. The risk ranking of the third-party component can be based on the NIST Common Vulnerability Scoring System (CVSS) or other frameworks such as CISA's Known Exploited Vulnerabilities catalog (KEV), Stakeholder-Specific Vulnerability Categorization (SSVC)[36], Exploit Prediction Scoring System (EPSS), Mend's open source database[37], OSV, and NIST's NVD, which are used to communicate the characteristics and severity of software vulnerabilities based on an associated risk score. For risk beyond a certain threshold, design an exceptions process with a defined timeline for replacing the component. The details of a solution to vulnerabilities in a third-party component should be made available using a VEX readable format for Supplier and Consumer consumption.

To augment your discovery process, perform ongoing monitoring and alerting of third-party component vulnerabilities in house by rerunning the ingestion process scanning tools or by deploying integrated re-occurring automated vulnerability scanning. Additional monitoring of security center reports provided by both internal and external researchers should be leveraged, as well as the use of threat intelligence bulletins from well-known entities, such as sponsored security announcements[38]. Further insight can be gained from automated services that track changes in OSS used within a product, providing notifications to the organization when updates to dependencies are required. Pay-per service SCA scanners or other local or cloud-based application security tools may also be leveraged where applicable. Vulnerabilities that are found should be tracked until remediated, and results recorded.

4 Open-Source Software Maintenance, Support and Crisis Management

This section describes the process used to maintain, monitor and update open-source software that has been approved for use within a company and incorporated into a product delivery. In this section, we review the mechanisms used to receive vulnerability and threat reports associated with third-party components, assess the risk of the reported vulnerability and define the type of activities associated with a crisis management process to mitigate the threat. Using the acceptance process described in section 3.2 "Vulnerability and Risk Assessment," an assessment may be conducted for any updated third-party component and then disseminate the availability of an update when all acceptance criteria is met using a secure delivery mechanism.

4.1 Maintaining Open-Source Software

Once an open-source component is adopted following the process outlined in sections 3 and 3,1 above, the third-party source is stored in a secure repository, where a continuity plan is used to define how vulnerabilities within OSS may be identified and addresses how inventory management is performed. If an SBOM has been previously created for the component, it may be used to

[35] https://osv.dev/

[36] https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=653459

[37] https://www.mend.io/vulnerability-database/

[38] https://www.cisa.gov/news-events/bulletins

automate the inventory process and scanned when vulnerabilities are reported. All third-party sources are monitored for defects and linked to ongoing vulnerability assessments in both the supplier and developer roles. Ongoing alerting and monitoring can occur in house by running the vulnerability scanning activity used within the ingestion process, or can be provided by external sources, such as reports from a security center, researcher, threat intelligence operations which look for Zero Day exploits, CISA automated notifications, security bulletins, CVE database monitoring, red teaming activities or by direct notifications from third-party source providers. Maintainers can also leverage resources such as automated notifications services and local or cloud-based scanning technologies that are described in section 3.2, "*Vulnerability and Risk Assessment*".

When vulnerable third-party software is identified, each product is assessed both manually and using an automated processes to determine what components are affected. A risk assessment for each affected product is determined using the considerations defined in section 3.2 "Vulnerability and Risk Assessment." The risk assessment takes into account the prevalence of the open-source component and its use within the product. Once identified, vulnerable products are tracked for remediation or exceptions granted. Remediation may take the form of a configuration change, a source or binary change, or may require an update of a third-party component. In the case of older sources that are no longer supported by the open-source provider, the fix may have to be backported to older source, which is maintained in the build repository or escrow. If a product is no longer being updated, strong consideration should be made to finding an alternative open-source solution.

Vulnerable products are tracked for remediation and an action plan created, identifying the actions to take and a planned timeframe for the delivery of a solution.

The plan also identifies the delivery mechanism that may be used, to include a patch, update or security bulletin, notice or advisory that can include configuration changes required, or a manual remediation step to mitigate the vulnerability. In some cases, the solution may require disabling a service, function or feature by modifying a configuration setting. The solution may also require a third-party repository update, which should use the procedures defined in the adoption process outlined above. Any fixes to the third-party component are tested and the repository is updated. All development groups that use the affected third-party component are notified through the monitoring roles listed above and all associated products are updated. Changes made to third-party components may be reflected in an updated SBOM and any associated remediation can be included in a VEX. These changes may be incorporated automatically at the completion of a product rebuild or referenced using an intermediate SBOM to the product component being updated. The updates to the vulnerable product are made available.

The fix is made available to the customer using a number of delivery mechanisms such as an automated update process, or a link to a download which can be applied at the customer's convenience. Updates can be provided using OS specific update package tools for example "apt," "yum" or a product update facility. A software update management facility can be used that allows packaging, inventory and update management functions to be performed automatically, and the update can be rolled out based on time, day, and other organization-based restrictions. Finally, updates can be provided by an on premis service engineer.

4.2 Crisis Management

The reader should familiarize themselves with NIST Special Publication (SP) 800-61, Rev 2[39], the Computer Security Incident Handling Guide. We may not supersede the NIST body of work. Rather, we may offer operational suggestions for a well-managed vulnerability response system that ties into the delivery of SBOMs and VEX components. The result of this process is a clear and timely response to customers regarding the status of a software issue. The creation and delivery of SBOM and VEX information should become an integrated step in the overall software development lifecycle.

A Crisis Management Plan defines the following:

- The nature and types of crises managed under this plan versus those managed by other functions.
- The structures that enable cross-functional information sharing, decision making, and communication.
- The individuals and teams involved in crisis response.
- The roles and responsibilities of the various teams that might be engaged in the crisis response:
    - What defines a crisis.
    - Crisis Response Concept of Operations.
    - Crisis Response Structure.
    - Definition of Product.
    - Definition of Software as a Service.

4.2.1 Crisis Definition

A crisis is defined as a situation that might or does compromise your company's reputation, products, goals, business value, ability to operate, or that of your customers. All events, even those localized to individual products or teams are expected to follow this plan, and to keep the Crisis Management Team (CMT) informed throughout the crisis to ensure timely response coordination, as required.

4.2.2 Crisis Response Concept of Operations

A highly functional team may rely on a tiered, cross-functional team structure to achieve strategic, crisis response efforts which includes a CMT and a Product Response Team (PRT).

At the onset of a crisis, the CMT may designate an incident manager who will own the crisis response. The CMT incident manager then partners with and coordinates the individual response activities of one or more PRTs to ensure a unified approach. The focus of the CMT is to facilitate effective communication, decision-making, response actions, and status between teams and, as needed with external teams. The secondary focus of the CMT is to provide visibility to outside

[39] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf

parties into progress toward resolution. The thought process used when evaluating a vulnerability response is outlined below. The incident manager will ensure full participation of each PRT.

When a vulnerability is identified or announced, a senior technician should study the announcement and gather an initial understanding of the issue. The output of this step is a concise statement explaining the vulnerability and potential remediation techniques. This information is key to ensuring a consistent means of remediation across the entire enterprise, thus minimizing the chances of an incorrect resolution. A severity level should be assigned. This may dictate the timeline to remediation with high/severe threats requiring immediate attention.

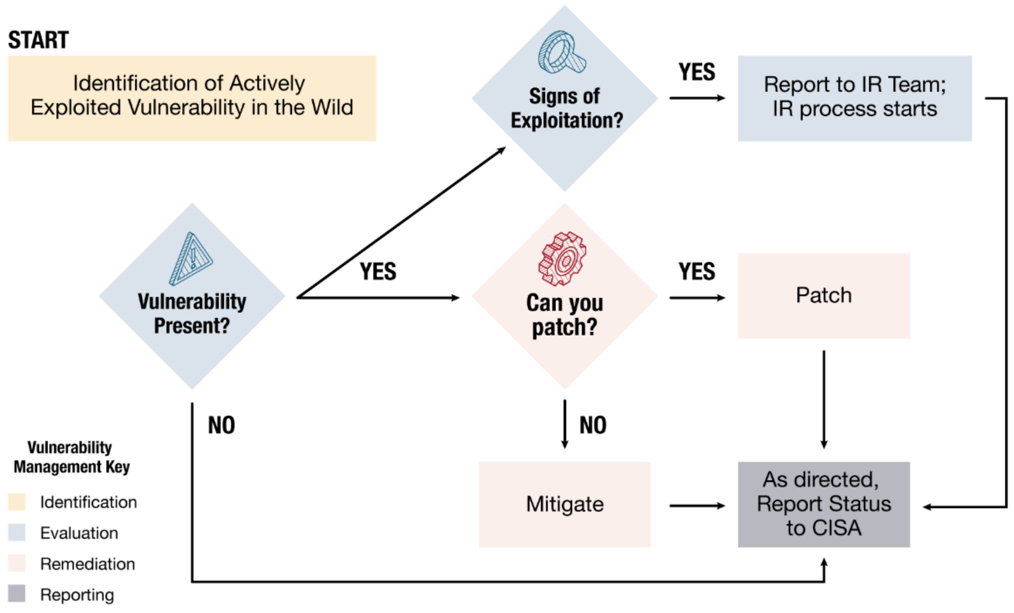

START
Identification of Actively Exploited Vulnerability in the Wild
Signs of Exploitation?
YES
Report to IR Team; IR process starts
Vulnerability Present?
YES
Can you patch?
YES
Patch
NO
NO
Mitigate
As directed, Report Status to CISA
Vulnerability Management Key
Identification
Evaluation
Remediation
Reporting

Figure 2: Vulnerability Response Process and Phases[40]

4.2.2.1 Inventory Role in Crisis Response

Having a complete understanding of the inventory of products that your organization develops is the foundation of a quality crisis response capability.

There are important steps to perform before an organization can properly respond to a crisis. A ledger of each distinct product should be maintained, regardless of the delivery mechanism for each product (e.g., on-premise versus Software as a Service (SaaS)). Each product in this inventory list should have a current owner and security champion. This inventory list cannot be a "one-and-done" list, it should be updated and verified routinely as employees may transfer or leave an organization overseen by the CMT. It is also suggested that the CMT have some type of notification from Human Resources for employees that terminate from the organization. Additionally, an email distribution list should be established to ease the communication with PRTs.

[40]https://www.cisa.gov/sites/default/files/publications/Federal_Government_Cybersecurity_Incident_and_Vulnerability_Response_Playbooks_508C.pdf

The inventory list should be current to properly facilitate efficient crisis issue research and customer communications for every product an organization develops.

Lastly, product teams should notify the CMT when a product reaches End of Support so the inventory list can remain free of errors.

4.2.2.2 Crisis Management Team Roles and Responsibilities

The job of a crisis team is to mobilize the workforce to assess and communicate on the status of a vulnerability succinctly, accurately and in a timely manner. The size of your organization, and the number of customers may dictate the sophistication of the communication. The crisis team should stay current with emerging standards for reporting and identifying the status of issues. Emerging standards like VEX are designed to facilitate digital communication with customers regarding the status of a products relationship with a vulnerability announcement. The goal of VEX is to facilitate a customer “self-service” discovery of the status of a product of interest as it relates to a vulnerability of interest.

4.2.2.3 The Crisis Management Team Process

4.2.2.3.1 Mobilize

Each product team within an enterprise should be notified and acknowledge the call to action within a specified timeframe. Each product team should update the control sheet with a status of “Under Investigation” when they begin researching the issue. This communication allows the CMT to track participation across the enterprise. The time requirement for this acknowledgement may be dictated by the severity of the issue. As research proceeds, all teams update the control list as their research is completed. The responses may vary per enterprise but should be consistent and could be similar to the following:

- Under Investigation
- Not Affected/Not applicable
- Not within execution path
- Affected, in engineering
- Work around available
- Corrected and available version number and optional build version

The result of this step may provide the CMT a clear means of keeping customers appraised in a transparent manner. Communication should consider moving toward VEX as that standard matures.

4.2.2.3.2 Resolve

If a programmatic modification is required to fix a vulnerability, care should be taken to re-deliver the product with only the required modification. Each supported repository tree should be re-built with only the required change that fixes the vulnerability, and all supported versions should be built and offered separately. Organizations should refrain from delivering the fix in an existing unreleased tree, or with components that are not already known to each customer. The patch should follow the normal development cycle in place within an organization. A label should be created designating the source tree as addressing the vulnerability. An SBOM should be generated and stored within the source tree for delivery to the customer with a patch. A VEX should also be generated and stored with the matching SBOM.

If the determination was that the software was NOT affected, the resolution is to update the VEX and store it with the current production location within each supported source tree. There is a possibility that a vulnerability can affect a customer environment adversely while the issue is under investigation. There is also a possibility that a product cannot resolve an issue at all. In either case a VEX should state affected, and a mitigation technique should be disclosed if one exists.

4.2.3 On-Premise Versus SaaS

Much of what was discussed above deals with the complexities of on-premise software. SaaS offerings should still have this type of maturity, but the number of supported software versions are likely to be considerably fewer in number. That said, there are likely to be weekly or bi-weekly releases of SaaS offerings, with roll-back versions available should there be major issues with new offerings. The same maturity is needed to ensure proper communication to customers.

4.3 Code Signing and Secure Software Delivery

Suppliers should perform code signing for all software and firmware in components that are delivered to external entities including customers and partners. Any gap in code signing processes, or in the security of the keys used in code signing operations, increases the risk of customer exposure to damage from malicious or counterfeit components, which could harm the Supplier entity's brand, reputation, liability, and future business. The purpose of this section is to define secure code signing requirements for suppliers to ensure that code signing operations and keys are secured with appropriate safeguards.

The scope of this section is inclusive of all Suppliers that provide software and firmware code for any components that are delivered to external entities. So, the suppliers in scope include Original Device Manufacturers (ODMs), Original Equipment Manufacturers (OEMs), Value-Added Resellers (VARs), Software Solution Providers, Contract Software Development Organizations, and Cloud Service Providers (CSPs).

4.3.1 Secure Code Signing Requirements

This section presents requirements for secure code signing. The requirements are organized into three activity-based security categories as follows:

- Perform Code Signing
- Use Proven Cryptography
- Secure Code Signing Infrastructure

4.3.1.1 Perform Code Signing

Supplier organizations should sign all artifacts that can be signed[41] and provided to external entities. Supplier organizations should also ensure availability of a mechanism to verify those signatures before installing or applying those components. This requirement applies to both initial

[41] This only applies to artifacts whose signature will still in a form that may be verified by the recipient of the artifact.

installation and upgrade processes. The signature validation mechanism should be documented in a security configuration guide.

For some products, the signing and/or signature validation[42] is performed by third-party components or platforms. If a third-party signature validation mechanism[43] is not available, a Supplier should provide a signature validation mechanism with any supplier proprietary code[44]. Any implemented proprietary signature validation mechanism, other than the one provided by a third-party component or platform, should be developed in accordance with Secure Development Lifecycle best practices[45] and stored/accessed in a trusted execution environment to mitigate the risk of tampering or sabotage of proprietary signature verification mechanisms.

4.3.1.2 Use Approved Cryptography

Code signing should always use a NIST approved[46] digital signature algorithm (a type of public-key cryptography, which is also known as asymmetric-key cryptography). The latest NIST guidance for code signing provides explanations of processes, techniques, and best practices. The more detailed specification for digital signatures is Federal Information Processing Standards (FIPS) 186-5[47], which approves versions of the Rivest Shamir Adleman (RSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) signature algorithms.

Public keys used for code signing should be certified by an approved trust anchor or trust path. The certificates and keys used to sign any code that is delivered to external entities should be issued from a commercial Certificate Authority (CA) entity to enable seamless operating system support for verification of the Public Key Infrastructure (PKI) chain-of-trust and trust anchors. Code signing certificates and keys issued by a public CA should have a one-year lifetime and be renewed annually. Timestamps should be applied to preserve signature validity beyond the certificate expiration date. Self-signed certificates and keys with an extended lifetime may be used in closed or proprietary systems or if otherwise dictated by technical requirements or functional specifications.

A future concern for most cryptographic processes, including code signing is the advent of quantum computing. Quantum computers can perform some computations exponentially faster than classical computers, which may put some digital signatures created with the current standard algorithms at significant risk in the future. To address this risk, new Post Quantum Cryptography (PQC) algorithms are being established via NIST's *Post-Quantum Standardization Project*[48] and NIST's *Recommendation for Stateful Hash-Based Signature Schemes*[49]. Migration to PQC may impact code signing operations, so suppliers should proactively establish a roadmap for adoption of PQC within

[42] https://www.nist.gov/publications/protecting-software-integrity-through-code-signing

[43] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-89.pdf

[44] https://csrc.nist.gov/CSRC/media/Publications/white-paper/2018/01/26/security-considerations-for-code-signing/final/documents/security-considerations-for-code-signing.pdf

[45] https://csrc.nist.gov/publications/detail/sp/800-218/final

[46] The use of non-approved cryptographic techniques, including proprietary ones, which have not been reviewed and approved by NIST is extremely risky and unlikely to meet regulatory or interoperability requirements.

[47] https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf

[48] https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04282021.pdf

[49] https://csrc.nist.gov/publications/detail/sp/800-208/final

their code signing operations. The NIST Guidance document, *Getting Ready for Post-Quantum Cryptography: Exploring Challenges Associated with Adopting and Using Post-Quantum Cryptographic Algorithms*[50] is a useful reference which organizations may choose to leverage to guide this adaptation effort.

4.3.1.3 Secure Code Signing Infrastructure

The origin, authenticity, and integrity of production code signing key material should be verified and maintained to ensure that code signing keys come from a trusted source and are provisioned into a securely managed cryptographic infrastructure to support secure code signing operations. The private keys used to perform code signing operations should be secured, for example, by using a Hardware Security Module (HSM)[51]. An alternative approach for signing OCI artifacts is Sigstore, which provides key management based on single-use keys and public transparency logs. Sigstore can overcome some of the complexity and risks of traditional key management.

4.3.2 Secure Software Update Delivery

On-premises software has historically been delivered to customers in one of three methods:

- Removable media
- Digital download from an originator hosted service
- Digital download from a third-party distributor

Regardless of the delivery method, an SBOM should accompany the software, the details for generation described in Section 5.1.2. Furthermore, the SBOM should over time become available for inspection prior to, and separate from, the installation procedure for the software. Additionally, the SBOM should be signed in a manner that shows its provenance and ties it to the software package delivered.

Before shipping the software package to customers, the developer or supplier should perform binary composition analysis to verify the contents of the package and reproducible build validation when possible. This process is described in “Securing the Software Supply Chain, Recommended Practices Guide for Developers,” Section 2.5.1 “Final Package Validation” and Section 5.1.3 “Software Composition Analysis (SCA) and VEX Format” of this document. Binary SCA tools can determine what is included in the final deliverables and identify potential issues in the final packages including a range of activities from the detection of potential vulnerabilities and threats to including Software Of Unknown Provenance (SOUP) and secrets inadvertently included in the final packages. This process describes one of many ways of producing an SBOM containing the true contents of the final package being delivered, allowing customers a means to evaluate the package. For more information on the discovery, access and transporting of SBOMs refer to Software Bill of Materials (SBOM) Sharing Lifecycle Report.”

[50] https://csrc.nist.gov/publications/detail/white-paper/2021/04/28/getting-ready-for-post-quantum-cryptography/final

[51] https://csrc.nist.gov/publications/detail/fips/140/3/final

For organizations hosting a delivery service, those systems and required resources should be "right sized" to service a flood of downloads resulting from incidents like log4j. The infrastructure for these systems should also be designed with DDoS protections in place.

An automated pull model seems to work best for client software. Each client notifies the back end of the product, version and build that it is running. If a new version is available, the software is downloaded and either installed, or made ready installation. This communication should be controlled via client and server certified mutual authentication. The client needs to be assured that the communication was made with the authorized server. In some environments, the server also may need to ensure the conversation is with a known client. There are several examples of man-in-the-middle attacks that take advantage of infrastructure with insufficient security.

When automatic updates are applied to a software product, a new SBOM is required to reflect the changes within the product. This new SBOM can be delivered automatically through a notification process or provided to the customer using an agreed upon pre-established communication channel.

Updates are usually tested in a non-production environment before being rolled out during a "maintenance window." The software may undergo further testing on the customers' network for more mission-critical software, including operational technology systems. This requires other efforts to manage which versions are where. The customer software acceptance procedures should modify the SBOM inventory repository when new software gets installed in the production environment. If a version of a product exits the production environment, the repository needs to reflect that fact.

Quite often, software is delivered with content other than binary executable code, such as configuration files and data sources used in the normal operation of the system. The rampant adoption of machine learning capabilities is a modern driving force in content delivery. The software and the associated content should be signed and verified by the supplier delivered agent on the endpoint or server. Updated content within a product, such as configuration files, databases or other data resources required for the product operational environment should be reflected in an updated SBOM.

While not related to software supply chain security, an organization should provide an update paradigm that allows the consumer organization to control their own update schedule. The process should be able to download new versions of software once, then have the updates disseminate to the internal customer network as controlled by customer policy.

5 Software Bill of Materials Creation, Validation, and Artifacts

This section provides a synopsis of the SBOM creation process and its relation to the OSRB activities covered in this document. It details information, tools, processes used, and considerations required for creating an SBOM. Also included are the means to update and distribute secure SBOMs due to software updates and changes, which may result from responding to vulnerabilities in third-party components. SBOMs use information produced during the acquisition and review of open-source components and can be used as part of vulnerability management. This section may reference relevant sections elsewhere in this document for additional detail.

5.1 Software Bill of Materials Background

The EO 14028, *Improving the Nation's Cybersecurity*, 12 May 2021, directed guidance in support of software security[52]. Government and industry are collaboratively writing this guidance as a result of the EO. Highlights from the EO state that organizations may be requested to provide a SBOM directly to the purchaser or publish it on a public website and that both government and non-government parties may be required to review the SBOM to ensure that software products comply with the minimum elements for an SBOM. The EO also directed the Department of Commerce and NTIA to publish *The Minimum Elements For a Software Bill of Materials (SBOM)* which outlines activities and data required for an SBOM as well as example formats that fulfill SBOM requirements[53]. SPDX[54] and CycloneDX[55] are the two most widely used machine-readable SBOM formats.

As one of the deliverables arising from the EO, the National Institute of Standards and Technology (NIST) produced guidance[56] on the role of SBOMs within the broader Software Lifecycle. An SBOM is a formal record containing the details and supply chain relationships of various components used in building software. The goals of SBOMs are to increase software transparency and document provenance. In the context of vulnerability management, the transparency facilitated by SBOMs supports the identification and remediation of vulnerabilities. The existence of an SBOM may be indicative of a developer or suppliers' application of secure software development practices across the Software Development Life Cycle (SDLC). Figure 3 illustrates an example of how an SBOM may be assembled across the SDLC.

[52] Presidential Executive Order (10428) on *Improving the Nation's Cybersecurity*, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[53] https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom

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

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

[56] https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity/software-security-supply-chains-guidance

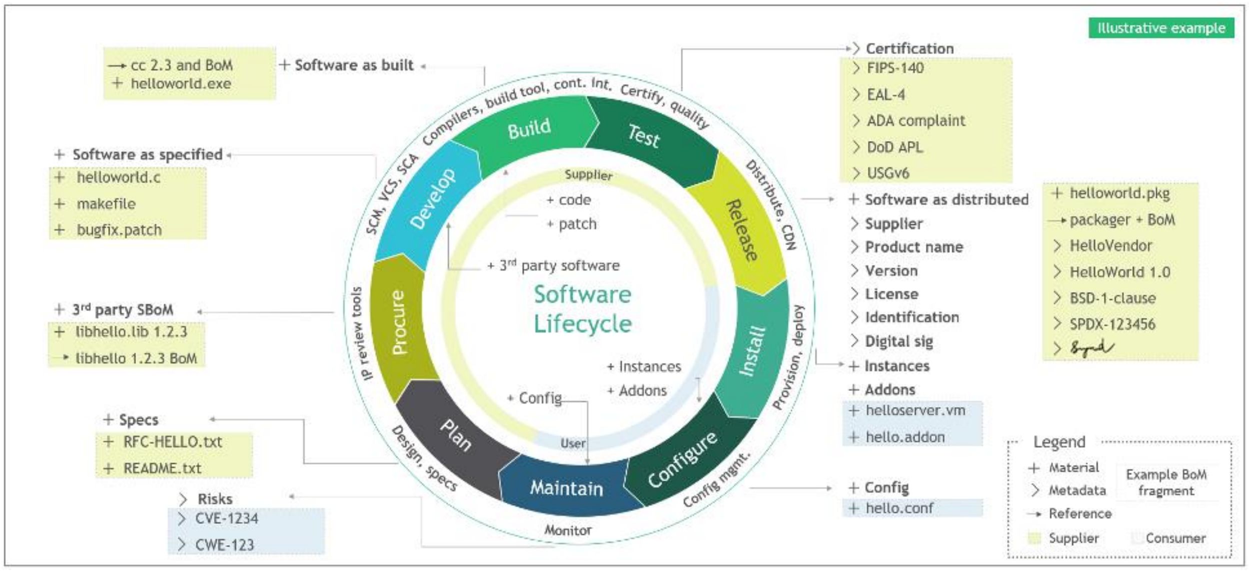


Figure 3: Software Life Cycle and Bill of Materials Assembly Line[57]

This guidance also maps SBOM-related capabilities to three maturity levels, Foundational, Sustaining, and Enhancing. The activities for these capabilities are mapped to processes within this document as follows:

- Ensure that SBOMs conform to industry standard formats to enable the automated ingestion and monitoring of versions. Acceptable standard formats currently include SPDX and CycloneDX.
    - Reference: 2.4 SBOM Overview
- Suppliers should provide SBOMs that meet the NTIA's Recommended Minimum Elements, including a catalog of the supplier's integrated open-source software and commercial components that are detectable via scans[58].
    - Reference:
        - 5.2 Supplier Activities
        - 5.1.2 SBOM Generation Tools and Training
        - Appendix B: Secure Supply Chain Consumption Framework (S2C2F)
- Map SBOM data with other data sources about risk, such as vulnerability data, supply chain information and additional data elements that inform the risk posture of the acquiring entity. Additional data elements include plug-ins, hardware components, organizational controls, and other community-provided components.[59]
    - Reference:

[57] https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity/software-security-supply-chains-software-1

[58] https://ntia.gov/sites/default/files/publications/sbom_options_and_decision_points_20210427-1_0.pdf

[59] https://www.nist.gov/system/files/documents/noindex/2021/06/08/GitLab%20-%20NIST%20Position%20Paper%20%232.pdf

        - 3.1 Open-Source Software Adoption Process
        - 3.2 Vulnerability and Risk Assessment
- Maintain vendor vulnerability disclosure report at the SBOM component level.
    - Reference:
        - 4.2 Crisis Management
        - 5.1.3 Software Composition Analysis and the VEX Format

Finally, the Office of Management and Budget (OMB) published Memorandum M-23-16 (Enhancing the Security of the Software Supply Chain through Secure Software Development Practices)[60], which allows Federal Agencies to:

- Require SBOMs based on the criticality of software or other criteria determined by the Agency.
- Require SBOMs to be in one of the formats defined in the NTIA Minimum Elements for SBOMs or subsequent guidance from CISA.
- Retain SBOMs or be able to access them from a publicly posted location.
- Consider sharing of SBOMs with other Agencies.

As such, software suppliers, especially those selling software to the Federal Government, should provide SBOMs for all components that are offered to customers. Should a supplier choose not to offer SBOMs for certain components, that fact should be clearly communicated to all parties creating and consuming the SBOM.

The ability to share SBOMs across organizational boundaries is crucial. Solutions such as the Digital Bill of Materials (DBoM)[61]. managed by the Linux Foundation provide a digital common where SBOM and other BOM information can be stored in defined taxonomies and accessed using defined policies. An individual DBoM[62] is the collection of records stored in the digital commons that are associated with an individual artifact (a piece of software, hardware, device, virtual artifact).

More details on each of the SBOM formats listed by NIST, and the minimum elements required for an SBOM are provided in section 2.4 "SBOM Overview." Furthering this line of thought, the sections below contain links to those formats' as well as respective training and tools in the areas of SBOM training, generation, license verification, conversion and validation.

https://spdx.dev/resources/tools

https://cyclonedx.org/tool-center/

Developers should complete a set of activities to ensure a full-spectrum coverage of SBOM which include:

- **Inventory Management (section 5.1.1)** – Helps understand what libraries and components are included in a software package. It also includes the types of scanning tools

[60] https://www.whitehouse.gov/wp-content/uploads/2023/06/M-23-16-Update-to-M-22-18-Enhancing-Software-Security.pdf

[61] https://www.ntia.doc.gov/files/ntia/publications/ntia_sbom_framing_sharing_july9.pdf

[62] https://dbom.io/ and https://dbom-project.readthedocs.io/en/2.0.0-alpha-1/

available for both developers and suppliers to support finding vulnerabilities and maliciously hidden insertion of components during development and when building the product (see section *4.3.2 "Secure Software Delivery"*).

- **Creating an SBOM (section 5.1.2)** – Lists and helps track libraries and components in a software deliverable.
- **Analyzing OSS and Utilizing a VEX document (section 5.1.3)** – Identifies known vulnerable libraries or components and potential mitigations to libraries or components.
- **License and Export Control (section 5.1.4) –** Provides information to verify software components for license incompatibilities which can cause liability and distribution issues for projects using OSS.
- **SBOM Validation (section 5.1.5) –** Validates the created SBOM.

5.1.1 Software Management and SBOMs

One of the most critical aspects of creating a comprehensive SBOM is software management. Understanding how a product is built and what software components it is built from is essential to producing accurate, complete, and up-to-date SBOMs. Software is complicated by the fact that a component may, itself, contain other components. Each of the components in a product may depend on specific versions of different components within the product.[63]Developers should collect all these components, versions, and dependencies (internal and external) to produce a useful SBOM.

Many developers are familiar with the techniques (including SCA, static, runtime, and vulnerability assessment scanners) used in the creation of an SBOM. SBOM creation tools can be incorporated at product build time, in final packaging or after product deployment in a secure developer environment. SBOM extraction tools can be broken down into four major categories, source, binary, package, and runtime extractors. Each extraction type has both benefits and drawbacks with respect to their availability, adoption, and performance. Also, many tools incorporate two or more of the extraction techniques to enhance the fidelity of the SBOM results.

SBOM source scanners are used during the creation of the product to identify the interdependencies between cooperating components. They provide the ability to construct dependency trees to fully understand the relationship between a software component and each library or other component of its dependent libraries. This information includes version information, as well as any licensing, and cryptographic capabilities/dependencies that may be provided within the source files. When used, configuration information such as file names and default settings may also be made available. Since source compilation is architecture specific, source extractors may not yield the precise dependencies for all delivered product architectures. Also, source extractors do not reflect the runtime environment of the system the software is deployed within, such as changes to dynamically linked libraries from system updates or other product installations that may use the same libraries but different versions. Source scanners also cannot identify differences in library implementations that may exist due to the use of multiple paths for which differing versions of a library have been deployed within a system environment. System path precedence may lead to variations in which dynamic library is being used.

[63] A product may also have operational dependencies, such as run time libraries. These dependencies are generally seen as beyond the scope of SBOMs. For example, the CycloneDX effort has defined an Operations Bill of Materials (OBOM) to meet this need.

SBOM binary scanners are deployed on components or products after they have been built. In this environment, much of the specific information about the components are available in the executable header of each component, to include the versioning information, library dependencies and architecture used. Binary scanners might not be able to obtain license, cryptographic and export restrictions as readily as source and package scanners and suffer from the same deployment fidelity as source scanners.

SBOM package scanners are used to extract information contained within an installable bundle of software. These packages can come in the form of executables or well-known product bundles used by system installation software or containers. Each package identifies the component information it contains, and in many cases, the licensing, cryptographic and export control restrictions which may apply to the component, or product. Dependencies of all components of the product, to include the minimum versions acceptable, are generally included within the software package. However, the actual version being used by the component cannot readily be derived unless specifically restricted by the product, and updates to required components may happen automatically for many software products so the deploy system may deviate from the initial SBOM. Package scanners can suffer from the same deployment fidelity as source and binary scanners with respect to system environment changes.

Runtime scanners can provide the best fidelity of a specific code path when being analyzed and creating an SBOM. They run on a deployed system and can accurately detect all components and track each dependency as it is currently deployed, with awareness to path precedence. They can also record product default component configuration state after installation. Runtime scanners are a more complex solution and not readily available for all environments.

Consideration on which extractor technique to use should be based on the environment the tool may be used in. In some situations, one or more scanning techniques can be combined to ensure that a comprehensive SBOM has been created. Then a separate tool can be used to validate the newly created SBOM. Care should be taken to ensure false positives are not introduced within an SBOM result due to the by-product of scanning tools that install additional packages, components and libraries that are used solely for the creation of the SBOM and not included in the actual product the SBOM describes.

Prior to signing an SBOM, the contents of an SBOM should be verified by either Quality Assurance or as part of the development process to ensure malicious content has not been added. When possible, reproducible builds can be used within a verification process to ensure the build environment has not been compromised. Reproducible builds require the creation of two or more SBOMs, built from segmented and secure independent build environments. The results of these builds are then compared for consistency. For more information on reproducible builds, refer to "*Securing the Software Supply Chain: Recommended Practices Guide For Developers,"* section 2.4, and "*Harden the Build Environment*" as well as the Linux Foundation's "*Core Infrastructure Initiative*"[64] and "*Reproducible Builds.*"[65]

[64] https://www.coreinfrastructure.org/

[65] https://reproducible-builds.org

When open-source software (OSS) is being included in a product, and the OSS is accompanied by an SBOM from the distributor of the OSS, that OSS SBOM should be validated before being bound to the incorporating product and the product's SBOM.

When creating an SBOM, consideration should be given on who is going to consume the SBOM and what formats are acceptable. Many tools are capable of producing the common formats listed in section 5.1.2 "*SBOM Generation Tools and Training*." When needed, translation tools are available to convert from one SBOM format to another, for example, SPDX to CycloneDX; this allows the correct format to be delivered to the consumer.

Once created and validated, SBOMs are signed to provide integrity and attestation for the contents of a product being delivered to a customer. In cases where translations are not performed by the producer of the SBOM, the provenance of the SBOM may suffer due to the lack of signing by the initial originator.

SBOMs and all associated artifacts created to attest to the validity of an SBOM are considered sensitive documents and need to be treated as such and stored in a secure repository with restricted access and version control.

An SBOM can be accompanied by additional vulnerability data, such as VEX, to provide additional information on vulnerabilities found during vulnerability scanning of a product prior to release and during the lifetime of the product. In many instances, these vulnerabilities are evaluated for the associated risk to the operational requirements of the product and required to be addressed prior to release. One means of identifying, addressing and tracking these concerns is by using a VEX, which can describe the vulnerability and its status. See section 5.1.3 "*Software Composition Analysis (SCA) and the VEX Format"* for more details on VEX and how it might be applied.

5.1.2 Software Bill of Materials Generation Tools and Training

The information and tools listed below provide examples of the support available for the automation of SBOMs and are used in the generation and validation of the two data formats that are widely used for SBOM, SPDX and CycloneDX. To understand how this procedure maps into the software life cycle, refer to Figure 3: "*Software Life Cycle and Bill of Materials Assembly Line.*" The tool examples described support the verification of software components for license incompatibilities, which can cause liability and distribution issues for projects using open-source software (OSS). Note that the example tools themselves are OSS and so should be placed in configuration management and tested as if they were internally developed code. Tools available for SWID, a format used to define software products and components that may be included in SPDX and CycloneDX SBOMS are also detailed below.

SPDX

- Tools - https://spdx.dev/use/tools/
- Information and Training
  - OpenSSF SPDX Tutorial – https://github.com/david-a-wheeler/spdx-tutorial
  - ISO/IEC 5962 SPDX – https://www.iso.org/standard/81870.html

- NTIA's How-To Guide for SBOM Generation provides the tag format (pg. 11) and examples (pgs. 24-31)[66]
- ISO/IEC 5230:2020(en) Information technology – OpenChain Specification
- NTIA's Tooling Ecosystem working with SPDX[67]

CycloneDX

- Tools - https://cyclonedx.org/tool-center/
- Information and Training
  - OWASP – https://owasp.org/www-project-cyclonedx/
  - OWASP – CycloneDX Learning Series[68]
  - NTIA –Tooling Ecosystem working with CycloneDX[69]

SWID

- Tools
  - NIST SWID Tools – https://pages.nist.gov/swid-tools.
  - NTIA – Tooling Ecosystem working with SWID[70].
- Information and Training
  - NIST provides resources on Software Identification (SWID) tagging[71] and tools[72] to build and validate the SWID tags and post them to the NVD site such as swid-builder, swid-maven-plugin, swidval and swid-repo-client.
  - NTIA's How – To Guide for SBOM Generation[73] – SWID tag format (pg 17) and examples (pgs. 24-25, 32-34)[74].
  - DevConf – Minting and Collecting SID Tags[75].

General SBOM Information and Training

- Linux Foundation
  - Linux Foundation Announces Software Bill of Materials (SBOM) Industry Standard, Research, Training, and Tools to Improve Cybersecurity Practices[76].
  - Linux Foundation - Generating a Software Bill of Materials (LFC192)[77].

---

[66] https://www.ntia.gov/files/ntia/publications/howto_guide_for_SBOM_generation_v1.pdf

[67] https://docs.google.com/document/d/1A1jFIYihB-IyT0gv7E_KoSjLbwNGmu_wOXBs6siemXA/edit

[68] https://www.youtube.com/playlist?list=PLqjEqUxHjy1X9nGMcjS1ikwxFMZAB2uea

[69] https://docs.google.com/document/d/1biwYXrtoRc_LF7Pw10TO2TGIhlM6jwkDG23nc9M_RiE/edit

[70] https://docs.google.com/document/d/1oebYvHcOhtMG8Uhnd5he0l_vhty7MsTjp6fYCOwUmwM/edit

[71] NIST Software Identification (SWID) Tagging, https://csrc.nist.gov/projects/Software-Identification-SWID

[72] NIST Software Identification (SWID) Tools, https://pages.nist.gov/swid-tools/

[73] https://docs.google.com/document/d/1oebYvHcOhtMG8Uhnd5he0l_vhty7MsTjp6fYCOwUmwM/edit

[74] https://www.ntia.gov/files/ntia/publications/howto_guide_for_sbom_generation_v1.pdf,

[75] https://www.youtube.com/watch?v=x86v5brZDfI

[76] Linux Foundation Announces Software Bill of Materials (SBOM) Industry Standard, Research, Training, and Tools to Improve Cybersecurity Practices https://linuxfoundation.org/press-release/linux-foundation-announces-software-bill-of-materials-SBOM-industry-standard-research-training-and-tools-to-improve-cybersecurity-practices/

[77] https://training.linuxfoundation.org/training/generating-a-software-bill-of-materials-SBOM-lfc192/

        - Overview of "*Using Open-Source Code.*"[78]
        - Top Level – "*Open-Source Best Practices for the Enterprise.*"[79]
        - Tools for Managing Open-Source Programs[80].
    - NTIA Software Bill of Materials listing of resources[81].
    - CISA Software Bill of Materials[82].
    - OpenChain Security Assurance Reference Guide[83].
    - OWASP Software Component Verification Standard (SCVS)[84].

5.1.3 Software Composition Analysis and the VEX Format

Generating or obtaining a SBOM for your application or software package is an important step toward improving software security. Based on recent attacks, Tenable, a well-known security company, has stated that once a vulnerability is found and disclosed, it only takes attackers 5.5 days to exploit it. Therefore, quickly identifying new vulnerable software components in a software package is essential, and automation is key to doing so quickly. SBOM generation tools should be used within automation pipelines where possible. SBOM generation tools can work with software security tools that can intake or generate SBOMs to support software component vulnerability analysis or custom vulnerability database mapping techniques. The type of generation tools called SCA tools can also support pipeline automation. Depending on the tool, SCA tools will try to analyze an application's dependencies for vulnerabilities and open-source license violations against software vulnerability databases, such as the NVD, public bug trackers, security advisories, and other sources. A best practice is to store SBOMs and verify them frequently, such as in each iteration of the build cycle (see section 2.1.2 Product Evaluation[85] subsection *Recommended mitigations*).

In addition to off-the-shelf software, there are tools which may be used in specialized environments, such as container images and applications. These tools can be used to process, integrate, and evaluate SBOMs. For the latest information on SBOM and VEX, refer to the "Software Bill of Materials | CISA"[86]" webpage. The "Featured Content" section highlights information on available SBOM-related concepts and their sharing lifecycle as well as information on the minimum

[78] https://www.linuxfoundation.org/tools/using-open-source-code/#policy

[79] https://www.linuxfoundation.org/resources/open-source-guides/

[80] https://www.linuxfoundation.org/tools/tools-managing-open-source-programs/#why-special-tools

[81] https://www.ntia.gov/SBOM

[82] https://www.cisa.gov/sbom

[83] https://www.openchainproject.org/security-guide

[84] https://owasp-scvs.gitbook.io/scvs/

[85] https://media.defense.gov/2022/Nov/17/2003116445/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_CUSTOMER.PDF

[86] CISA SBOM Workstreams, https://www.cisa.gov/sbom

requirements for a VEX[87]. Workstreams are also available as part of this resource to assist in the evolution and refinement of SBOM and VEX.[88]

In addition to the SBOM directive, the Executive Order directed the NIST to publish the *Guidelines on Minimum Standards for Developer Verification of Software*[89]. The minimum standards include SCA. The NIST guidelines reference a Dependency-Check[90], an SCA tool that attempts to detect publicly disclosed vulnerabilities within a project's dependencies. Other tools include Maven, a build automation and management tool that can provide a Project Object Model (POM) to help Dependency-Check identify known vulnerable software components[91].

Syft is a Command Line Interface (CLI) tool and library for generating a SBOM. Tools such as OWASP Dependency-Track supports SBOM continuous monitoring to include the NVD, the Sonatype OSS Index, NPM Advisories, and VulnDB from Risk Based Security as well as the VEX content in security advisories.[92]

A VEX document provides a method to provide clarifying security information for components in a specific software package's SBOM which are marked as vulnerable. For example, if the vulnerable part of an open-source component is inaccessible to attackers or removed and therefore cannot be exploited, this could be listed in the VEX document that accompanies an SBOM. In this manner, VEX provides additional context which may reduce the number of "false positive" vulnerabilities that a vulnerability scanning tools would report against an SBOM due to the specific package instance. A vendor could provide a package's SBOM and VEX document online to support customer requests.

[87] https://www.cisa.gov/resources-tools/resources/minimum-requirements-vulnerability-exploitability-exchange-vex

[88] https://www.cisa.gov/news-events/alerts/2023/11/06/cisa-published-when-issue-vex-information

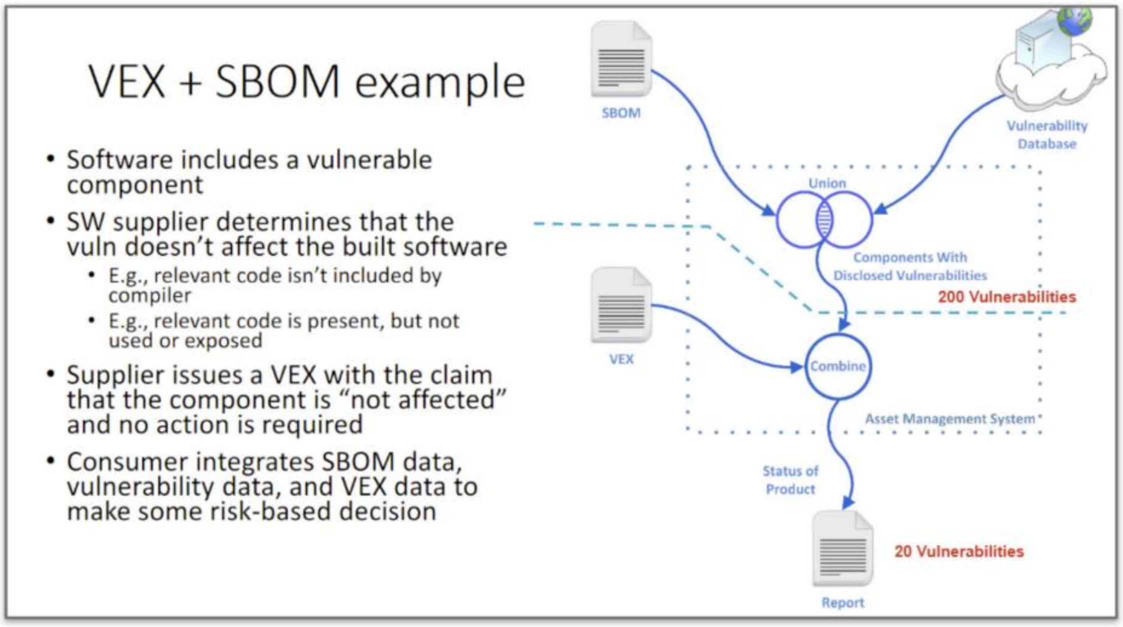
[89] https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8397.pdf

[90] https://owasp.org/www-project-dependency-check/

[91] Dependency Check & Maven https://jeremylong.github.io/DependencyCheck/dependency-check-maven/check-mojo.html ; Dependency Check & Maven https://mvnrepository.com/artifact/org.owasp/dependency-check-maven ; *Dependency-Check Comparison* (to Dependency-Track), https://docs.dependencytrack.org/odt-odc-comparison/

[92] NTIA, *Vulnerability-Exploitability eXchange (VEX) - An Overview*, https://ntia.gov/files/ntia/publications/vex_one-page_summary.pdf

CISA, *Vulnerability Exploitability eXchange (VEX) - Use Cases*, https://www.cisa.gov/sites/default/files/publications/VEX_Use_Cases_April2022.pdf



VEX + SBOM example

- Software includes a vulnerable component
- SW supplier determines that the vuln doesn't affect the built software
  - E.g., relevant code isn't included by compiler
  - E.g., relevant code is present, but not used or exposed
- Supplier issues a VEX with the claim that the component is "not affected" and no action is required
- Consumer integrates SBOM data, vulnerability data, and VEX data to make some risk-based decision

Figure 4: NTIA Framing Group – Framing NTIA Software Supply Chain Transparency[93]

Tools exist and continue to emerge that integrate data with a VEX to produce more intelligence about the software and its risks. For example, CycloneDX and SPDX work with Dependency Check and VEX, but the reader should be aware that feature support between various tools may vary.[94]

The SBOM community continues to enhance and refine SBOM implementation, with innovation from across the software ecosystem and marketplace. There are also efforts to address gaps and offer community guidance collectively. Check the CISA website dedicated to SBOM for the latest information.[95]

5.1.4 License and Export Control

Many of the use cases around SBOM predate security applications and were initially advocated to better track and comply with the complexities around OSS licenses. A comprehensive and widely used list of OSS licenses is maintained by the SPDX community, which supports machine-readability and automation[96]. The information and tools listed below support automation of SBOM formats for

[93] NTIA Framing Group – Framing NTIA Software Supply Chain Transparency, https://www.ntia.doc.gov/files/ntia/publications/framing_2021-04-29_002.pdf

Adolus, What is VEX and What Does it Have to Do with SBOMS?, https://blog.adolus.com/what-is-vex-and-what-does-it-have-to-do-with-sboms

[94] https://github.com/CycloneDX/cyclonedx-maven-plugin and https://cyclonedx.org/capabilities/vex/

[95] CISA, *Software Bill of Materials,* https://www.cisa.gov/SBOM

[96] https://spdx.org/licenses/

license and export compliance assessment for projects using OSS. The Linux Foundation maintains the OpenChain[97] ISO/IEC 5260 International Standard for open-source license compliance[98].

SPDX

There are several tools to help with license and export control verification for SPDX.

- The Linux Foundation provides an automated license compliance tool, FOSSology[99], which itself is an open-source tool and should be managed as internally developed code. FOSSology is a license compliance software system and toolkit for license, copyright and export control scans. The *SPDX Online Tool*[100] validates SPDX formatted SBOM file types and can verify licenses are compatible with an organization's policy.
- Tools such as Cybeats SBOM Studio include license verification and security assessment.

CycloneDX

CycloneDX provides a variety of tools for managing and using CycloneDX SBOMs, including license verification[101].

- Cybeats SBOM Studio[102].
- NTIA, Tooling Ecosystem working with CycloneDX[103].
- CycloneDX SBOM's can also be converted into SPDX-compatible SBOMs with the CycloneDX CLI[104] and the cdx2spdx tool[105] so that the SPDX FOSSology[106] tool can automate license compliance.

SWID

The NIST provides resources on SWID tagging and tools to build and validate SWID tags such as swid-builder, swid-maven-plugin, swidval and swid-repo-client[107].

SWID is supported by the Internet Engineering Task Force (IETF) who provides the Concise Software Identification Tags guidance.[108]

[97] https://www.openchainproject.org ; https://www.openchainproject.org/security-guide

[98] https://www.linuxfoundation.org/resources/publications/understanding-us-export-controls-with-open-source-projects

[99] https://www.fossology.org

[100] https://www.tools.spdx.org/app/validate/

[101] https://www.cyclonedx.org/tool-center/

[102] https://www.cybeats.com/sbom-studio

[103] NTIA, https://docs.google.com/document/d/1biwYXrtoRc_LF7Pw10TO2TGIhlM6jwkDG23nc9M_RiE/edit

[104] CycloneDX CLI https://github.com/CycloneDX/cyclonedx-cli

[105] Cdx2spdx tool https://github.com/spdx/cdx2spdx

[106] https://www.fossology.org

[107] https://csrc.nist.gov/projects/Software-Identification-SWID ; https://pages.nist.gov/swid-tools/ ; https://nvd.nist.gov/

[108] https://datatracker.ietf.org/doc/draft-ietf-sacm-coswid/

The SPDX tool page[109] lists the CyberProtek tool[110] to convert the SWID format to SPDX. Once in the SPDX format, the FOSSology tool[111] can be leveraged for automation.

The below tools will translate between SBOM formats.[112]

- SwiftBOM - SPDX (.spdx), SWID (Extensible Markup Language (.xml)), CycloneDX (.xml, JavaScript Object Notation (.json))
  - Demo at https://democert.org/sbom/
  - Source code at https://github.com/CERTCC/SBOM/tree/master/sbom-demo
- DecoderRing - SPDX (.spdx), SWID(.xml)
  - Source code at https://github.com/DanBeard/DecoderRing
- SPDX tools - SPDX (.spdx, json, yaml, rdf, xml, xls)
  - Demo at https://tools.spdx.org/app/
  - Source code at https://github.com/spdx/spdx-online-tools
- CycloneDX CLI - CycloneDX (.xml, .json), SPDX (.spdx)
  - Source code at https://github.com/CylconeDx/cyclonedx-cli

5.1.5 Software Bill of Materials Validation

Examples of SBOM Validation & Verification tools can be found in NTIA's "*Software Suppliers Playbook: SBOM Production and Provision* guidance[113] (page 8):

- Validation of SBOM Format
  - SPDX Online Tool validates SPDX format SBOMs and converts between SPDX SBOM file types and checks licenses - https://tools.spdx.org/app/validate/
  - SWID Tools - https://pages.nist.gov/swid-tools/swidval/.
  - CycloneDX CLI Tool and Web Tool validates CycloneDX format SBOMs - https://github.com/CycloneDX/cyclonedx-cli/; https://cyclonedx.github.io/cyclonedx-web-tool/.

5.2 Supplier Activities

Suppliers define policy and validate the integrity of the product using an SBOM that follows one of the standard formats defined in section 2.4 "SBOM Overview" above. A supplier, as defined in section 2.0 of the *Securing the Software Supply Chain for Suppliers*[114], provides a software package, whether it is a development group in smaller companies, or higher-level management that oversees development teams within a larger company, as defined in section 2 of the *Securing the Software*

[109] https://spdx.dev/tools-commercial/

[110] https://cyberprotek.com

[111] https://www.fossology.org

[112] https://www.ntia.doc.gov/files/ntia/publications/ntia_sbom_formats_energy_brief_2021.pdf

[113] NTIA, Software Suppliers Playbook: SBOM Production and Provision, https://www.ntia.gov/files/ntia/publications/software_suppliers_sbom_production_and_provision_-_final.pdf

[114] https://media.defense.gov/2022/Oct/31/2003105368/-1/-1/0/SECURING_THE_SOFTWARE_SUPPLY_CHAIN_SUPPLIERS.PDF

Supply Chain for Developers[115], who delivers their own product or a repackaged product. Third-party software information is also included, as described in section 3.1 "Open-Source Software Adoption Process," this may include the origin of the software to include the company/organization and possibly the country when meaningful. Additional information that may also be included within an SBOM, but not necessarily required today, are licensing and export information. The validation process should ensure the minimum elements, as defined in the NTIA "*The Minimum Elements For a Software Bill of Materials (SBOM)"* or successor CISA documents are met. Once validated using the techniques defined in section 5.1.5 "SBOM Validation," the SBOM is made available to customers with the shipping products using various methods listed in section 4.3.2 "Secure Software Delivery." Suppliers should continuously scan for vulnerabilities within their products which includes third-party components. When vulnerabilities are found they should be addressed and the associated SBOM and VEX should be updated and provided to customers, refer to section 5.1.3 "*Software Composition Analysis and the VEX Format*" for more details. Suppliers also need to be aware of and track the latest developments for open-source management and the use of SBOMs using the guidelines given by the Cybersecurity and Infrastructure Security Agency (CISA), refer to "*Appendix A: Ongoing Efforts*."

5.2.1 Software Bill of Materials Validation and Verification Tools

The SBOM and its contents must be validated and verified. Validation assures that the SBOM data is appropriately formatted and can be integrated into various tools and automation. Verification ensures the content within the SBOM is accurately described and all components and related information on a product for licensing and exporting are represented.

Many organizations are increasingly incorporating tools into the build and source repository facility to automate this process and provide artifacts which can attest to the verification of the SBOM being delivered. Both the content of the package, the executables, libraries and configuration files, and the actual format of the SBOM, should be validated. Any open-source software components should be verified for license or export restrictions. In some organizations, validation is performed first by the developer during build/packing of the product and then by the developer/supplier before customer delivery to verify the integrity of the SBOM being delivered. For more information on the formats and tools available for validation, refer to section 5.1.5 of this document "*SBOM Validation.*"

A good reference on guidance for the SBOM process can be found in NTIA's publication "*Software Suppliers Playbook: SBOM Production and Provision"*[116] guidance. It is important that developers understand the end-user requirements for SBOM generation and how this information might be used by both suppliers and customers. Additional process information relating to SBOMs and acquisitions can be found in the "*Software Consumers Playbook: SBOM Acquisition, Management, and Use"*[117].

---

[115] https://media.defense.gov/2022/Sep/01/2003068942/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF

[116] https://www.ntia.gov/files/ntia/publications/software_suppliers_sbom_production_and_provision_-_final.pdf

[117] https://www.ntia.gov/files/ntia/publications/software_consumers_sbom_acquisition_management_and_use_-_final.pdf

Appendix A: Ongoing Efforts

Suppliers and developers should be aware of and track the latest developments for open-source management and the use of SBOMs using the guidelines from the Cybersecurity and Infrastructure Security Agency (CISA). The NTIA's multi-stakeholder working groups facilitated the development of foundational SBOM-related documents, many of which are referenced in this guidance and can be found at https://ntia.gov/SoftwareTransparency. The NTIA's work continues through a series of community-led workstreams facilitated by CISA focused on the following topics :

- Cloud and Online Applications – integrating current understanding around SBOM into the context of online applications and modern infrastructure.
- On-Ramps and Adoption – promoting education and awareness to help lower the costs and complexities of adoption, allowing newer or less mature organizations to provide, request, and use SBOMs to secure and understand their organization's risk.
- Sharing and Exchanging – concepts related to moving SBOMs, and related metadata, across the software supply chain.
- Tooling and Implementation – opportunities and challenges for automating the SBOM ecosystem.

CISA also facilitates VEX-related discussions with the SBOM community and has published white papers on defining VEX minimum elements, use cases, and status justifications.

Additional information, including schedules and how to participate in the workstreams and the production of VEX white papers, visit https://www.cisa.gov/sbom.

Correspondingly, the information in this document will continually evolve due to the complexity and urgency of securing the software supply chain for a vast number of stakeholders and environments. Some of the areas to monitor the CISA website for updated guidance and clarification are the following:

- Validating an SBOM is complex. Developers, suppliers and consumers may not have the same ability to generate and compare SBOMs delivered for a product. Indeed, for complex software, the "true" SBOM may vary across the lifecycle of development, build, deployment, and run. A base set of common criteria to include a minimum set of elements within an SBOM, the formats used to describe information and the "definitions" used in identifying the meaning of the elements may evolve into a more meaningful data set as the integration of the current standards are adopted by a wide variety of consumers and their unique environments.
- The frequency of SBOM updates and the life cycle of an SBOM is evolving over time. While it is envisioned that SBOMs may be used to evaluate the risk associated with a product as input to future adoption decisions, it is also envisioned that they may provide valuable threat management capabilities over the life cycle of the products they describe. The frequency of SBOM updates needs to be considered in this discussion as cloud and automated product update procedures may change product composition and these changes need to be reflected in "updated" SBOMs. Many modern organizations have daily or hourly builds. The update paradigm of both the software update process and its

associated SBOM needs to be clearly understood, specifically by the customers who may receive them. How component modifications are reflected in an SBOM, and whether these changes are reflected by supplying a completely new, all-inclusive SBOM or just a partial SBOM update for the component modified needs to be clearly understood. In addition, a means to determine the differences between two or more SBOMs may be required to allow the ability to calculate the risk associated with the specific changes within a product due to the update, if any.

- Care should be taken to ensure SBOMs are created that reflect the composition of the product they describe. Depending on the environment the SBOM is created in, for example a software appliance, the necessary tools required to automatically generate an SBOM may require additional packages to be installed to perform this operation. Packages and tools used solely for the support of SBOM creation, should only be reflected in the document creation section of an SBOM, but not included as contents/package information in the final SBOM produced. In addition, the SBOM tools incorporated and used within a product environment should undergo the same risk assessment and adoption process as detailed for OSS components.
- Finally, strategies outlined in the SSDF should be used to manage identity and access control with respect to the generation, signing, updating and distribution of SBOMS for specific products. Some of the key considerations identified are:
    - Implement Roles and Responsibilities (PO.2)
    - Implement and Maintain Secure Environments for Software Development (PO.5)
    - Protect All Forms of Code from Unauthorized Access and Tampering (PS.1)

Appendix B: Secure Supply Chain Consumption Framework (S2C2F)

A **Secure Supply Chain Consumption Framework** (S2C2F) is a threat-based risk reduction framework that is focused on securing how developers consume open-source software into the developer's workflow. Microsoft has been implementing the framework since 2019 and continues to lead the S2C2F SIG within the OpenSSF (https://github.com/ossf/s2c2f).

S2C2F Requirements

Below is a table of the requirements mapped to the 8 different S2C2F practices. Two of the requirements have prerequisites identified that are outside the scope of the S2C2F.

Practice	Requirement ID	Maturity Level	Requirement Title	Benefit
Ingest it	ING-1	L1	Use package managers trusted by your organization[118]	Your organization benefits from the inherent security provided by the package manager
	ING-2	L1	Use an OSS binary repository manager solution	Caches a local copy of the OSS artifact and protects against left-pad incidents, enabling developers to continue to build even if upstream resources are unavailable
	ING-3	L3	Have a Deny List capability to block known malicious OSS from being consumed	Prevents ingestion of known malware by blocking ingestion as soon as a critically vulnerable OSS component is identified, such as colors v 1.4.1, or if an OSS component is deemed malicious
	ING-4	L3	Mirror a copy of all OSS source code to an internal location	Supports Business Continuity and Disaster Recovery (BCDR) scenarios. Also enables proactive security scanning, fix it scenarios, and the ability to rebuild OSS in a trusted build environment

[118] https://opensource.com/article/20/11/trust-package

Practice	Requirement ID	Maturity Level	Requirement Title	Benefit
Scan It	SCA-1	L1	Scan OSS for known vulnerabilities (i.e., CVEs, GitHub Advisories, etc.)	Able to update OSS to reduce risks
	SCA-2	L1	Scan OSS for licenses	Ensures your organization remains in compliance with the software license
	SCA-3	L2	Scan OSS to determine if its end-of-life	For security purposes, no organization should take a dependency on software that is no longer receiving updates
	SCA-4	L3	Scan OSS for malware	Able to prevent ingestion of malware into your CI/CD environment
	SCA-5	L3	Perform proactive security review of OSS	Identify zero-day vulnerabilities and confidentially contribute fixes back to the upstream maintainer
Inventory It	INV-1	L1	Maintain an automated inventory of all OSS used in development	Able to respond to incidents by knowing who is using what OSS where. This can also be accomplished by generating SBOMs for your software
	INV-2	L2	Have an OSS Incident Response Plan	This is a defined, repeatable process that enables your organization to quickly respond to reported OSS incidents
Update It	UPD-1	L1	Update vulnerable OSS manually	Ability to resolve vulnerabilities
	UPD-2	L2	Enable automated OSS updates	Improve MTTR to patch faster than adversaries can operate

Practice	Requirement ID	Maturity Level	Requirement Title	Benefit
	UPD-3	L2	Display OSS vulnerabilities as comments in Pull Requests (PRs) • **Prerequisite**: Two-person PR reviews are enforced.	PR reviewer doesn't want to approve knowing that there are unaddressed vulnerabilities
Audit It	AUD-1	L3	Verify the provenance of your OSS	Able to track that a given OSS package traces back to a repo
	AUD-2	L2	Audit that developers are consuming OSS through the approved ingestion method	Detect when developers consume OSS that isn't detected by your inventory or scan tools
	AUD-3	L2	Validate integrity of the OSS that you consume into your build	Validate digital signature or hash match for each component
	AUD-4	L4	Validate SBOMs of OSS that you consume into your build	Validate SBOM for provenance data, dependencies, and its digital signature for SBOM integrity
Enforce It	ENF-1	L2	Securely configure your package source files (i.e., nuget.config, npmrc, pip.conf, pom.xml, etc.)	By using NuGet package source mapping, or a single upstream feed, or using version pinning and lock files, you can protect yourself from race conditions and Dependency Confusion attacks
	ENF-2	L3	Enforce usage of a curated OSS feed that enhances the trust of your OSS	Curated OSS feeds can be systems that scan OSS for malware, validate claims-metadata about the component, or systems that enforce an allow/deny list. Developers should not be allowed to consume OSS outside of the curated OSS feed

Practice	Requirement ID	Maturity Level	Requirement Title	Benefit
Rebuild It	REB-1	L4	Rebuild the OSS in a trusted build environment, or validate that it is reproducibly built • **Prerequisite**: Sufficient build integrity measures are in place to establish a trusted build environment	Mitigates against build-time attacks such as those seen on CCleaner and SolarWinds. Open-Source developers could introduce scripts or code that aren't present in the repository into the build process or be building in a compromised environment
	REB-2	L4	Digitally sign the OSS you rebuild	Protects the integrity of the OSS you use
	REB-3	L4	Generate SBOMs for OSS that you rebuild	Captures the supply chain information for each package to enable you to better maintain your dependencies, auditability, and blast radius assessments
	REB-4	L4	Digitally sign the SBOMs you produce	Ensures that consumers of your SBOMs can trust that the contents have not been tampered with
Fix It + Upstream	FIX-1	L4	Implement a change in the code to address a zero-day vulnerability, rebuild, deploy to your organization, and confidentially contribute the fix to the upstream maintainer	To be used only in extreme circumstances when the risk is too great and to be used temporarily until the upstream maintainer issues a fix

Appendix C: References

1. Presidential Executive Order (10428) on *Improving the Nation's Cybersecurity*, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/
2. NTIA *The Minimum Elements For a Software Bill of Materials (SBOM)* https://www.ntia.doc.gov/files/ntia/publications/SBOM_minimum_elements_report.pdf
3. SPDX, https://spdx.dev/ (last visited May 18, 2021).
4. CycloneDX, https://cyclonedx.org/ (last visited May 18, 2021).
5. 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).
6. Linux Foundation Announces Software Bill of Materials (SBOM) Industry Standard, Research, Training, and Tools to Improve Cybersecurity Practices https://linuxfoundation.org/press-release/linux-foundation-announces-software-bill-of-materials-SBOM-industry-standard-research-training-and-tools-to-improve-cybersecurity-practices/
7. https://training.linuxfoundation.org/training/generating-a-software-bill-of-materials-SBOM-lfc192/
8. Linux Foundation, *Overview of "Using Open Source Code"*, https://www.linuxfoundation.org/tools/using-open-source-code/#policy
9. Linux Foundation, *Top Level - "Open Source Best Practices for the Enterprise"*, https://www.linuxfoundation.org/resources/open-source-guides/
10. Linux Foundation, *Tools for Managing Open Source Programs* - https://www.linuxfoundation.org/tools/tools-managing-open-source-programs/#why-special-tools
11. NTIA, *Software Suppliers Playbook: SBOM Production and Provision*, https://www.ntia.gov/files/ntia/publications/software_suppliers_SBOM_production_and_provision_-_final.pdf
12. *Software Consumers Playbook: SBOM Acquisition, Management, and Use*, https://docs.google.com/document/d/1Ae0l1MDS8m1on58e8mdVIA9NujzPD0k5j352VlDZr9I/edit#heading=h.xgt811s3780y
13. Tenable, https://www.wsj.com/articles/cyber-matters-heed-the-window-of-opportunity-1527115463
14. NIST Guidelines on Minimum Standards for Developer Verification of Software https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8397.pdf
15. OWASP, Dependency Check - https://owasp.org/www-project-dependency-check/
16. Dependency Check & Maven https://jeremylong.github.io/DependencyCheck/dependency-check-maven/check-mojo.html

17. Dependency Check & Maven https://mvnrepository.com/artifact/org.owasp/dependency-check-maven
18. *Dependency-Check Comparison* (to Dependency-Track), https://docs.dependencytrack.org/odt-odc-comparison/
19. NTIA, *Vulnerability-Exploitability eXchange (VEX) - An Overview*, https://ntia.gov/files/ntia/publications/vex_one-page_summary.pdf
20. CISA, *Vulnerability Exploitability eXchange (VEX) - Use Cases,* https://www.cisa.gov/sites/default/files/publications/VEX_Use_Cases_April2022.pdf
21. NTIA Framing Group – Framing NTIA Software Supply Chain Transparency, https://www.ntia.doc.gov/files/ntia/publications/framing_2021-04-29_002.pdf
22. Digital Bill of Materials (DBoM) Consortium is a Linux Foundation project that maintains a digital common place where SBOM and other BOM information can be stored in defined taxonomies and accessed using defined policies. https://dbom.io/
23. CISA, *Software Bill of Materials,* https://www.cisa.gov/SBOM
24. NIST Security Considerations for Code Signing, https://csrc.nist.gov/publications/detail/white-paper/2018/01/26/security-considerations-for-code-signing/final
25. CA Security Code Signing Whitepaper, https://casecurity.org/wp-content/uploads/2016/12/CASC-Code-Signing.pdf
26. Linux Foundation, 2022. "Secure Supply Chain Consumption Framework (S2C2F)." Available at (https://github.com/ossf/s2c2f)

Appendix D: Acronym List

Acronym	Expansion
CA	Certificate Authority
CCL	Commercial Control List
CI	Continuous Integration
CIPAC	Critical Infrastructure Partnership Advisory Council
CISA	Cybersecurity and Infrastructure Security Agency
CLI	Command-Line Interface
CMT	Crisis Management Team
CNSS	Committee on National Security Systems
CSAF	Common Security Advisory Framework
CSP	Cloud Service Provider
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DBOM	Digital Bill of Materials
EAR	Export Administration Regulations
ECDSA	Elliptic Curve Digital Signature Algorithm
EO	Executive Order
EPSS	Exploit Prediction Scoring System
ESF	Enduring Security Framework
EU	European Union
FIPS	Federal Information Processing Standards
HSM	Hardware Security Module
IAM	Identity and Access Management
IETF	Internet Engineering Task Force
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
json	JavaScript Object Notation
KEV	Known Exploited Vulnerabilities
NIST	National Institute of Standards and Technology
npm	Node Package Manager
NSA	National Security Agency
NTIA	National Telecommunications and Information Administration

Acronym	Expansion
NVD	National Vulnerability Database
ODM	Original Device Manufacturers
ODNI	Office of the Director of National Intelligence
OEM	Original Equipment Manufacturers
ONCD	Office of the National Cyber Director
OMB	Office of Management and Budget
OpenSSF	Open-Source Security Foundation
OSRB	Open-Source Review Board
OSS	Open-Source Software
OWASP	Open Web Application Security Project
PII	Personally Identifiable Information
PKI	Public Key Infrastructure
PO	Prepare the Organization
POM	Project Object Model
PQC	Post Quantum Cryptography
PRT	Product Response Team
PS	Protect the Software
PW	Produce Well-Secured Software
RSA	Rivest-Shamir-Adleman
RV	Respond to Vulnerabilities
S2C2F	Secure Supply Chain Consumption Framework
SAST	Static Analysis
SaaS	Software as a Service
SaaSBOM	Software as a Service Bill of Materials
SBOM	Software Bill of Materials
SCA	Software Composition Analysis
SDLC	Software Development Life Cycle
SOUP	Software of Unknown Provenance
SP	Special Publication
SSDF	Secure Software Development Framework
SSVC	Specific Vulnerability Categorization
SWID	Software Identification
TTPs	Tactics, Techniques, And Procedures

Acronym	Expansion
U.S.	United States
VAR	Value-Added Reseller
VEX	Vulnerability Exploitability eXchange
XML	Extensible Markup Language