REVERSINGLABS

REVERSINGLABS
WHITE PAPER

# Understanding the Requirement for Software Bills of Material in Executive Order 14028

## The Attack on Software Supply Chain

In December 2020, FireEye researchers discovered “a supply chain attack trojanizing SolarWinds Orion business software updates”. The backdoor in Orion – a platform for centralized monitoring and management of IT infrastructure – allowed the attackers full administrative access to Orion customers’ infrastructure. The attack affected over 100 private sector entities and at least 9 Federal agencies, including the Departments of Defense, Commerce, Energy, Justice, Homeland Security, State, Treasure, and the National Institute of Health.

While the SolarWinds attacks involved Russian nation state actors planting malicious code in software updates, similar outcomes can result from exploiting vulnerabilities in widely used applications. Attackers infiltrated the development environment and altered the behavior of CodeCov, a software auditing tool for developers, to collect developer credentials needed for the next stage of their attack. Four zero day vulnerabilities in Microsoft Exchange Server provided attackers with the ability to steal sensitive information, install malware, and insert backdoors on thousands of organizations.

## Cybersecurity Executive Order 14028

While there have been growing concerns over the security of software used by the government for years, the SolarWinds and Microsoft attacks prompted action. In May, 2021 the President issued an Executive Order (EO) on Improving the Nation’s Cybersecurity. The order requires private sector organizations to "ensure its products are built and operate securely, and partner with the Federal Government to foster a more secure cyberspace.”

The EO requires the National Institute of Standards and Technology (NIST) to publish standards for secure software development including criteria for secure software development environments, using automated tools to identify vulnerabilities in code, maintaining accurate and up-to-date data, provenance (i.e., origin) of software code or components, and providing a Software Bill of Materials (SBOM) to purchasers of software.

## Why the EO Includes SBOM Requirements

Awareness of a risk is fundamental to mitigating that risk. A Software Bill of Materials is a listing of all components and dependencies in an application or system, including open source and commercial components. Having a list of every software component and dependency simplifies the effort of:

- Mapping known vulnerabilities to components to determine the level of risk
- Tracking down which software and suppliers are affected when a vulnerability like Log4j occurs
- Pinpointing the location of suspicious software behavior changes, injected malware and other indicators of software tampering

A 2021 study found that the average application included over 500 open source components and that 84% of the applications included at least one vulnerable open source component. The latter figure is not surprising, as thousands of new vulnerabilities are disclosed in open source components each year.

A complete and accurate Software Bill of Materials is essential to provide vendors and customers with visibility into the risk within an application or supply chain. Without insight to vulnerable, malicious or compromised components, organizations cannot defend against these attack vectors or adequately assess risk.

The Executive Order acknowledges risk in the supply chain from third-party components and requires any organization providing software to the Federal Government to also provide (or publish on a public website) a Software Bill of Materials. In response to this, other secure development standards have also included SBOM requirements, including:

- **NIST Special Publication 800-161 Rev1**: Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations. The publication instructs Federal Agencies and Departments to "***Establish a governance capability for managing and monitoring components of embedded software to manage risk across the enterprise (e.g., SBOMs paired with criticality, vulnerability, threat, and exploitability to make this more automated).***" This governance is a foundational practice, critical to successfully interacting with suppliers and improving cybersecurity supply chain risk management.
- **NIST Publication 800-218**: Secure Software Development Framework (SSDF). 800-218 requires organizations to
  - "Collect, maintain, and share provenance data for all components and other dependencies of each software release (e.g., in a software bill of materials [SBOM])." (PS.3.2)

- Obtain provenance information (e.g., SBOM, source composition analysis, binary software composition analysis) for each software component, and analyze that information to better assess the risk that the component may introduce. (PW.4.1)
- "Check code for backdoors and other malicious content" (PW.7.2)

In short, every software supplier to federal agencies now has two deliverables — the software and a Software Bill of Materials.

## How SBOM Have Evolved (Your old one may not be sufficient)

SBOMs have been around for 20 years, at first coinciding with the growth and adoption of open source software. When software developers first used open source components, the primary concern was complying with sometimes confusing or limiting license agreements that represented an intellectual property risk.

Open source can be issued under one of hundreds of licenses, or under no license at all. Some licenses require developers to link to the code and others require that proper attribution be provided. Some high-profile lawsuits and settlements related to violating open source license obligations, including Free Software Foundation (FSF) vs Cisco in 2009 and CoKinetic Systems vs Panaso nic Avionics in 2017, provided the incentive needed to adopt tools that could identify open source components from build manifests and other development tooling and enumerate the licenses associated with them – with appropriate policy definition and reporting so that legal and compliance teams could limit licensing risks.

In contrast to what is now required under the Executive Order, organizations rarely – if ever – shared SBOMs with their customers. They viewed the composition of their software as their intellectual property.  SBOM generation methods that grew out of those early circumstances can struggle with the new shift in focus, where transparency into all parts of the software, more insight into the release and distribution processes, and data sharing across organizations are necessary for managing risk.

### SBOM Generation Methodologies

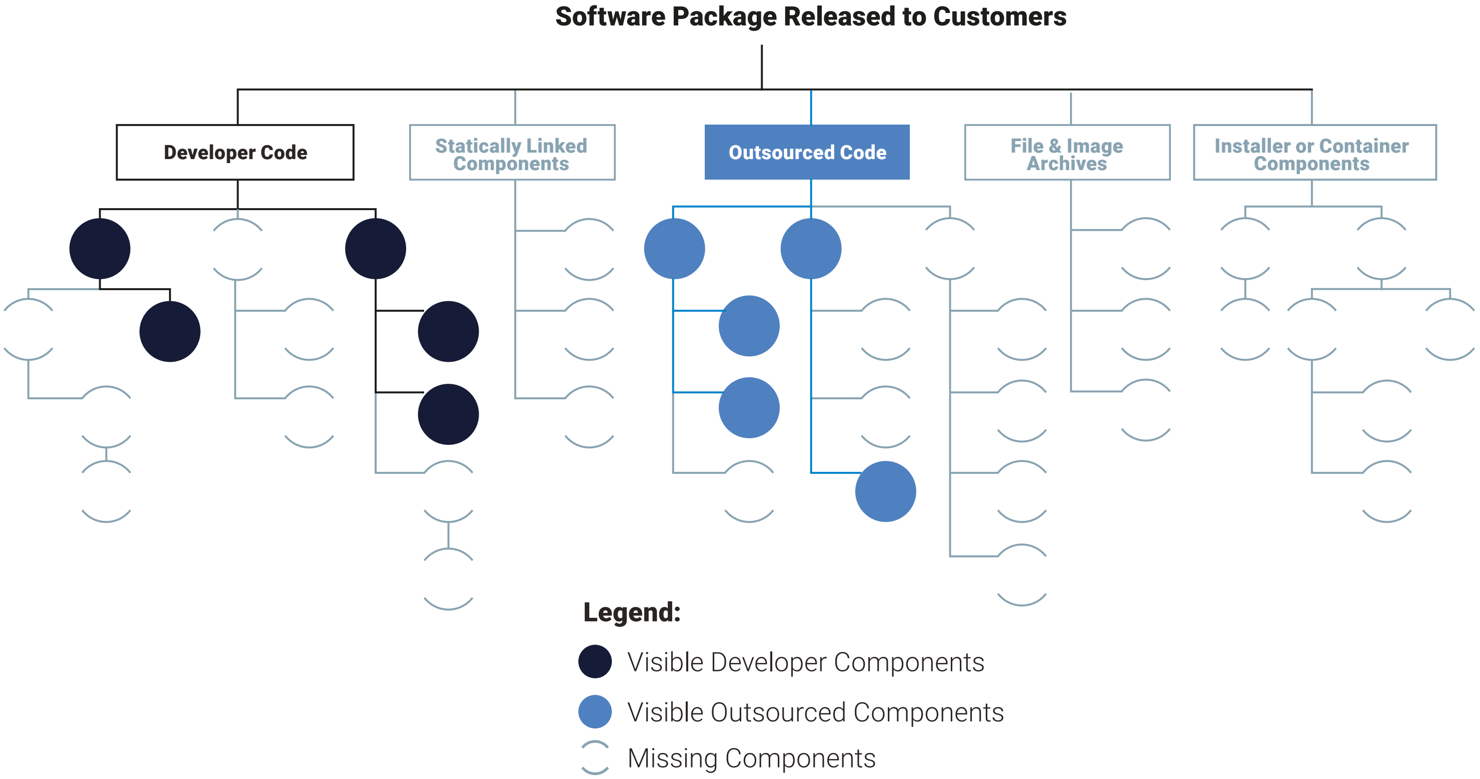
Generating an SBOM manually is time consuming and inaccurate. Software development teams using spreadsheets or shared documents rarely are aware of each component and subcomponent used over the course of the development cycle, and versions can change each day. The class of tools that automates creating an SBOM is known as Source Composition Analysis (SCA).

The most common approach used by SCA to identify components is Manifest Parsing. This works by interrogating build manifests and package managers during the build process to identify which open source components have been declared by development as required ("declared dependencies"). It then determines other "undeclared dependencies" are required by listed components. Once the SBOM is complete, it maps the components to databases of the licenses under which each component was published.

While this approach appears accurate (the components listed as required by development are those listed in the SBOM), Manifest Parsing has several technical shortcomings

- Components statically linked or added directly to the code by developers will not be declared by the manifest
- Imprecise declarations (e.g., "version 2 or higher", "latest") may generate the wrong version number in the SBOM. Since vulnerabilities affect only specific versions, this can lead to false positives and false negatives
- Components embedded within software containers are not part of a build manifest
- Manifest parsing is unusable with programming languages such as C and C++. This can also make it impossible to automatically generate reliable SBOM for legacy software.

## Typical SCA solutions cannot generate comprehensive SBOMs


Software Package Released to Customers
Developer Code
Statically Linked Components
Outsourced Code
File & Image Archives
Installer or Container Components
Legend:
● Visible Developer Components
● Visible Outsourced Components
◌ Missing Components

# The Future of SBOMs

As required by the Executive Order, the National Telecommunications and Information Administration (NTIA) published The Minimum Elements For a Software Bill of Materials in 2021. These include:

## Minimum Elements

**Data Fields**

Document baseline information about each component that should be tracked: Supplier, Component Name, Version of the Component, Other Unique Identifiers, Dependency Relationship, Author of SBOM Data, and Timestamp.

**Automation Support**

Automatic generation and machine-readability to allow for scaling across the software ecosystem. Data formats used to generate and consume SBOMs include SPDX, CycloneDX, and SWID tags.

**Practices and Processes**

A number of items that focus on the mechanics of SBOM use should be addressed in any policy, contract, or arrangement to ask for or provide SBOMs, including Frequency, Depth, Known Unknowns, Distribution and Delivery, Access Control, and Accommodation of Mistakes

## Future Requirements

Additionally, the document provides a preview of how these "minimum requirements" may evolve. This includes the potential future SBOM requirements for SBOM Integrity and Authenticity, SBOM for Cloud-based Software and Software-as-a-Service, Component Relationships, and Vulnerability and Exploitability in Dependencies.

**Legacy Software and Binary Analysis**

The NTIA document also specifically notes that source code may not be available for all legacy software. In those cases, only the object code available for SBOM generation and binary analysis tools can be used to better understand the components and dependencies in the software in question.

**Emerging Software Supply Chain Concepts**

NIST continues work on cybersecurity supply chain risk management and to update its standards concerning the EO and the Secure Software Development Framework. Security and development teams should expect NIST to build off the "minimum requirements" and develop "Foundational, Sustaining, and Enhancing" recommendations for SBOM, Vendor Risk Assessments, Open Source Software Controls, and Vulnerability Management Practices. The requirements are expected to be "risk based", with different requirements based on the criticality of the software, the information it manages, and the threat environment.

> Legacy software's older code base, and its frequent use in important parts of critical infrastructure, often makes transparency more important, especially for assessing risk from known vulnerabilities."
>
> **Department of Commerce**
> "The Minimum Elements for an SBOM"

# Operationalizing SBOMs

As previously noted, automation is required to produce accurate SBOM at scale. Early in the SDLC this can include integration with development tools and issue tracking software. Later in the development process, integration is required with build systems; vulnerability management, and incident response. The most critical point of integration is just prior to release or deployment of software and containers. This requires binary analysis.

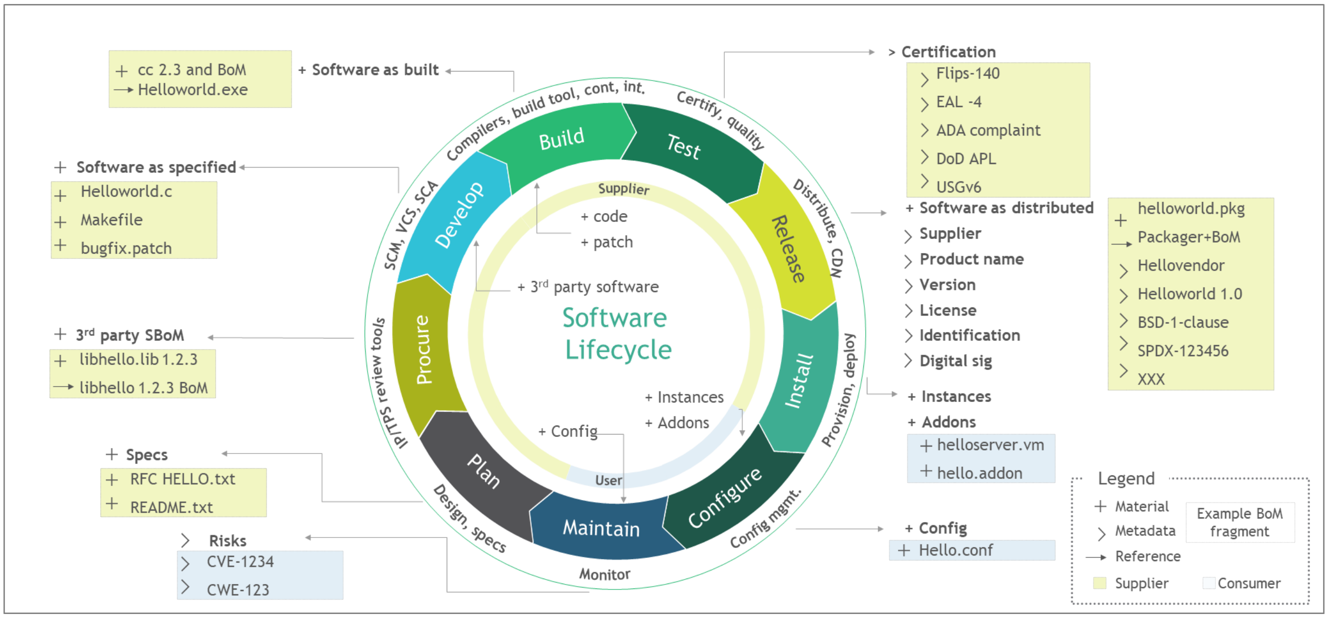
NIST Software Supply Chain Security Guidance website illustrates how a SBOM illustrates an example of how an SBOM may be assembled across the SDLC, shown below. The pale green  boxes follow software and components as they change over time. The SBOM components  of “Software as specified” and “Software as built” (where open-source SCA tools have traditionally focused) can differ significantly from the SBOM components of “Software as distributed”.

For example, a number of other items are often packaged with the software to assist with installation: installer software, separate installation libraries, or even an entire container. Vulnerabilities or tampering within these items can introduce the same security risks. The SBOM created during the final build (i.e.“Software as built”) will not list these components, leaving organizations blind to the risks.

“Software as distributed” – the binary and package –  is the release that matters from a security standpoint.  Only binary analysis solutions can produce an accurate and complete “as distributed” SBOM.

Additionally, complete and accurate SBOMs are  valuable tools for determining the impact of attacks like Solarwinds or components with newly discovered supply chain risks such as the Apache log4j vulnerability.  Binary analysis simplifies the effort in identifying all of the transitive dependencies included, regardless of how deeply the components are layered within the application.

### Software Life Cycle & Bill of Materials Assembly Line



Source: Software Security in Supply Chains: Software Bill of Materials (SBOM)

## Making SBOMs Part of Daily Activity

Complying with the EO at scale requires organizations to operationalize generation and review of SBOM information. At scale, this includes changes to people, processes, and technology.

- **People** – The requirement for greater security (delivered in part by the SBOM) and faster delivery cycles requires changes in team make up. DevSecOps adoption integrates multiple technical teams to meet delivery schedules and security/compliance requirements. Security no longer operates as a separate entity. Instead, they assist product engineering by providing the appropriate information in a timely manner.
- **Processes** – The EO is explicit in the need to track software components as they change and vulnerabilities affecting them. Creating and maintaining an SBOM is now a requirement. However, developers are still under great pressure to deliver required functionality by a specific date and cannot be expected to manually track every component in every application. At the same time, manual processes for creating the SBOM and tracking vulnerabilities as they are disclosed are unreliable.
- **Technology** – Traditional security testing tools are unable to accurately identify vulnerabilities in open source – even those that have been known about for years. Using the development team's list of "declared dependencies" in a build manifest is simply automating an unreliable process. Signature-based solutions providing "ground truth" SBOMs based on compiler output and continuous monitoring of newly disclosed vulnerabilities and compromises mapped to production software are better at meeting the requirements outlined in the EO.

## What's Needed Now

To meet the Executive Order and the demands of today's high-velocity development environments, organizations need a solution that integrates into the development team's existing tools; that creates accurate and complete SBOMs at scale; and that provides rapid feedback on both security vulnerabilities and malicious code. It must also be capable of supporting modern programming languages and older but still actively used server applications built with legacy code .

ReversingLabs' binary analysis technology provides scalability and speed across the development lifecycle. ReversingLabs analyzes actual build output, providing "ground truth" SBOMs including code added directly by developers rather than relying on "declared dependencies" to identify components. The table below compares ReversingLabs' approach to commercial SCA technologies across key factors.

| **Value Category** | **Commercial SCA** | **ReversingLabs** |
|---|---|---|
| SBOM Completeness | Uncovers primary (top level) open source dependencies | Uncovers primary and subcomponent (nth level) open source, third-party and statically linked dependencies |
| SDLC Integration | Analysis during build phase<br>(SBOM will have long list of known unknowns) | Analysis during build, release and deployment phases provides a more complete picture of the SBOM and software risk |
| SBOM Accuracy | Reports component data as captured | Checks whether collected component data matches actual objects in the binary |
| Integrated Risk Analysis | Identifies known vulnerabilities in primary (top level) open source components | Identifies known vulnerabilities in primary and subcomponent (nth level) opensource, third-party and statically linked dependencies |
| Malicious Code Detection (Counterfeit components, Hidden functionality) | NA | Integrated component risk scoring based static analysis of components to identify known vulnerabilities, malware, backdoors, and counterfeit software |

## ReversingLabs Fills the Security Testing Gap

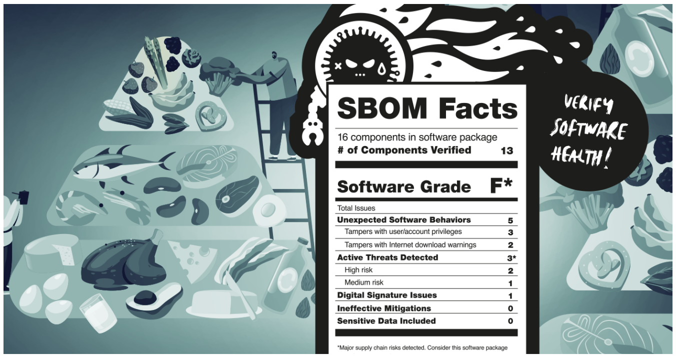
ReversingLabs fills the gap left by traditional security testing tools to secure the software supply chain, reduce brand risk, and comply with the Executive Order on Cybersecurity. ReversingLabs works on the compiled application, without needing source code access, to find vulnerabilities, backdoors, suspicious behaviors, and malware that can be introduced after the development process where static analysis scanners and Source Composition Analysis tools operate. It quickly and carefully inspects every file in every component to discover malicious code, Indicators of Compromise, and invalid or compromised certificates. ReversingLabs provides:

- Independent verification of the software, including fourth (and nth)- party code
- An accurate Software Bill of Materials, including the presence of legitimate but vulnerable components and counterfeit components
- Assurance that backdoors and suspicious behaviors have not been introduced by compromised build servers within the vendor's software engineering processes.
- Software quality assessment to identify issues such as exposed secrets, incorrectly mitigated vulnerabilities, digital signing issues, etc.

# Additional Resources:

REVERSINGLABS
SOFTWARE PACKAGE DECONSTRUCTION
HOW TO ASSESS RISKS TO YOUR SOFTWARE SUPPLY CHAIN
with Tim Stahl

Watch our educational series Software Package Deconstruction. Each episode will unpack, analyze, and expose hidden risks inside SBOM components in some of the largest most complex software packages.



**SBOM Facts**

16 components in software package

| | |
|---|---|
| **# of Components Verified** | **13** |
| **Software Grade** | **F*** |
| Total Issues | |
| **Unexpected Software Behaviors** | **5** |
| Tampers with user/account privileges | 3 |
| Tampers with Internet download warnings | 2 |
| **Active Threats Detected** | **3*** |
| High risk | 2 |
| Medium risk | 1 |
| **Digital Signature Issues** | **1** |
| **Ineffective Mitigations** | **0** |
| **Sensitive Data Included** | **0** |

*Major supply chain risks detected. Consider this software package

Learn how ReversingLabs helps you see software components and risks that others miss: Learn more

**Understand what is in your software,
get a free Software Bill of Materials
(SBOM) report & risk analysis now.**

GET A FREE SBOM REPORT & ANALYSIS

Worldwide Sale:
+1.617.250.7518
sales@reversinglabs.com

REVERSINGLABS