SBOM: How a Software Bill of Materials Strengthens Security

Learn how a Software Bill of Materials (SBOM) strengthens security by tracking components, identifying vulnerabilities, and ensuring compliance.

 Swaroop Sham November 26, 2024 4 minutes read

Main takeaways from this article:

- A Software Bill of Materials (SBOM) lists every component in your software, ensuring clarity and control over supply chains.
- SBOMs enable fast responses to vulnerabilities, as seen with Log4j and SolarWinds, strengthening supply chain defenses.
- Regulatory mandates like PCI DSS and government requirements tie security directly to SBOM adoption.
- Automating SBOM generation eliminates errors and ensures up-to-date vulnerability tracking.
- Wiz's agentless SBOM scanning offers real-time insights, helping teams stay on top of changing software environments.

What is an SBOM?

A Software Bill of Material (SBOM) is a comprehensive inventory that details every software component that makes up an application. It includes open-source and commercial third-party libraries, API calls, versions, and licenses.

Applications used in the supply chain ecosystem are an amalgam of elements from several sources. These sources may contain vulnerabilities that cybercriminals could exploit during supply chain attacks. SBOMs ease vulnerability management by providing information about these elements.

When incorporated into the software delivery life cycle (SDLC), SBOMs enhance visibility into third-party software patch status and licenses, and provide other security-related information that can bolster code integrity.

SBOM's role in software supply chain security

With rising supply chain attacks, SBOMs have become essential tools for tracking and securing software components. Gartner estimates that by 2025, **60% of organizations will require SBOMs** as part of their cybersecurity practices, a significant increase from 20% in 2022. Government mandates, such as the White House's **Executive Order on Improving the Nation's Cybersecurity**, also now require SBOMs for organizations working with government agencies.

The Log4j and SolarWinds incidents highlight the importance of SBOMs:

- **Log4j Vulnerability (CVE-2021-44228)**: This widely used Java library created a security nightmare for

organizations worldwide. Without SBOMs, many companies were left scrambling, piecing together which components were at risk. Those with SBOMs, though? They pinpointed affected components fast, dodging massive fallout.

- **SolarWinds Attack**: In 2020, attackers compromised SolarWinds Orion software, impacting around 18,000 customers, including top-tier companies and government agencies. An SBOM could've cut through the chaos, helping victims trace the malware quickly and respond effectively.

Why SBOMs are important

SBOMs provide a detailed list of all the components in a software application, helping organizations identify and manage security risks. They also improve transparency, make it easier to track and update software dependencies, and more:

1. Transparency and visibility

Think of SBOMs as your software's blueprint. They give developers a clear view of all third-party software components—like open-source libraries—used in their applications. This transparency helps teams weigh the risks before adding a library and stay on top of vulnerabilities after deployment.

2. Regulatory compliance

With governments and industry standards cracking down on software security, SBOMs have become a compliance essential. From PCI DSS to HIPAA, many regulations now demand a clear record of software components. An SBOM not only helps meet these requirements but also keeps your organization out of trouble, whether it's fines or reputation damage from licensing mishaps.

3. Incident response and forensics

When something goes wrong, an SBOM can be a lifesaver. It pinpoints exactly which component is vulnerable, helping teams zero in on the problem area, **prioritize their response**, and assess the broader impact.

What should an SBOM include?

An SBOM should include details about all open-source and proprietary software components used in a product, including their names, versions, and licenses. It should also specify the relationships between components and their dependencies.

The following key elements should be present in an SBOM:

1. Component identifiers: This includes metadata such as supplier and component names, component origins, description and maintainers, artifact ID, timestamp, version number, and specific references, such as Git commit IDs or a SHA-1 hashes for every component.

2. Dependencies: The relationship between each component and its dependencies must be clearly documented.

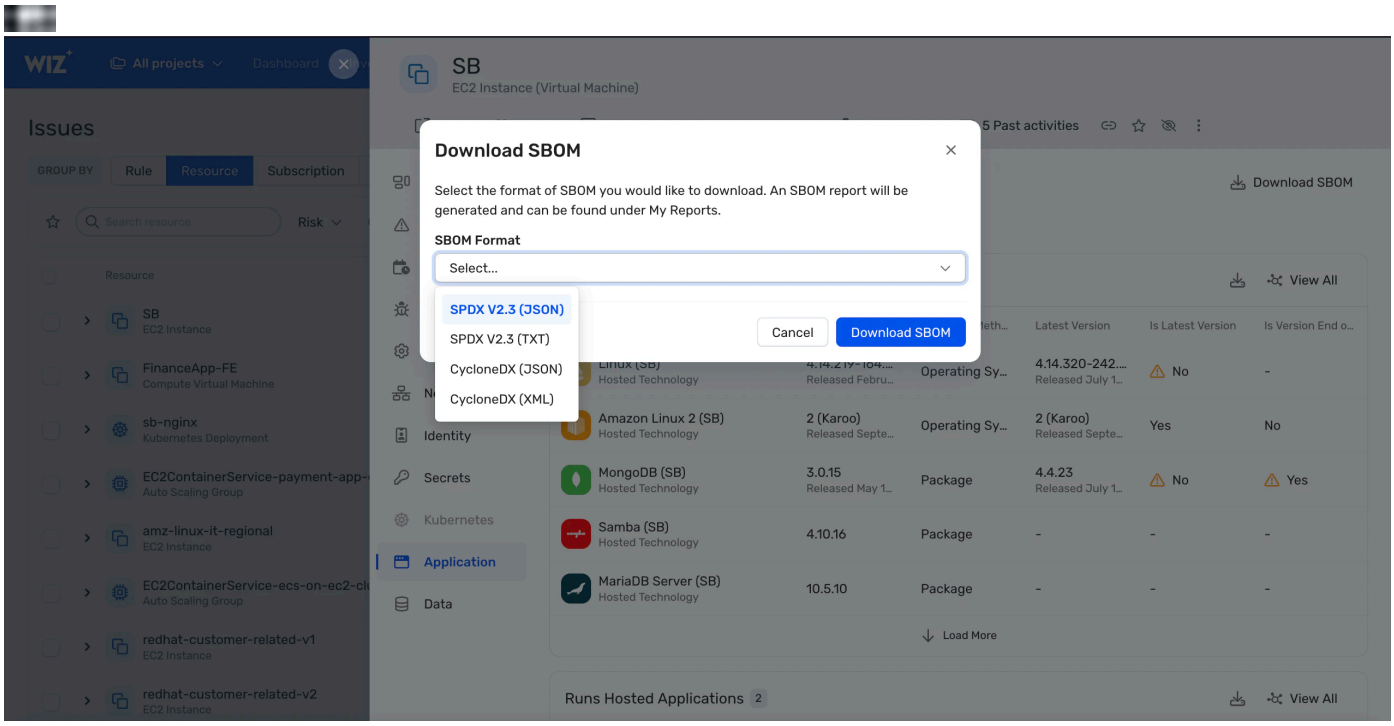
3. Version information: This includes software version number, file name, and operating system to enable easy installation and prevent compatibility issues. Version information enables you to track necessary updates or patches for each component.

4. Vulnerability data: It is essential to include information about known vulnerabilities associated with each component. This data can be obtained from vulnerability catalogs such as the National Vulnerability Database (NVD) or CVE (Common Vulnerabilities and Exposures).

5. Licensing information: Each component has licensing terms (e.g., MIT, Apache, and BSD Licenses). The SBOM must contain these terms to ensure compliance with license obligations.

7. External references: These include URLs or documentation related to each component. They provide additional context on the functions of the components.

Common SBOM formats



SBOMs can be generated manually or automatically.

- The **manual method** involves listing all software components and their respective versions, licenses and dependencies in spreadsheets. It is only suited to small-scale deployments and is prone to human error.
- The **automatic method** involves integrating SBOM tools into the Continuous Integration/Continuous Deployment (CI/CD) pipeline.

After generation, SBOMs are structured in two major formats: CycloneDX and Software Package Data Exchange (SPDX). Below is a brief description of each.

Format	Description
SPDX	SPDX supports representation of SBOM information, such as component identification and licensing information, alongside the relationship between the components and the application. SPDX enables information gathering and sharing in various file formats, including human-readable and machine-parsable formats such as JSON, XML, and YAML. It enhances transparency, and facilitates license compliance.
CycloneDX	CycloneDX supports listing internal and external components/services that make up applications alongside their interrelationships, patch status, and variants. It structures the data as an XML or JSON file, and enables you to add details such as Common Vulnerability Scoring System (CVSS) scores and descriptions to the SBOM. CycloneDX is highly extensible, allowing developers to add new capabilities as required.

To give you a better understanding of the SBOM formats, consider this example of the CycloneDX inventory in JSON format:

```
{
"bomFormat": "CycloneDX",
"specVersion": "1.4",
"serialNumber": "urn:uuid:3e673487-395b-41h8-a30f-a58468a69b79",
"version": 1,
"components": [
{
"type": "library",
"name": "nacl-library",
"version": "1.0.0"
}
]
}
```

In addition to the two formats covered above, organizations can also use Software Identification (SWID) tags, which are usually installed during deployment. SWID tags provide SBOM information such as software component release dates and licenses. Regulatory bodies and the US government consider SWID tags, CycloneDX, and SPDX as acceptable SBOM formats.

SBOM implementation: a step-by-step guide

Creating an SBOM might sound daunting, but breaking it into manageable steps can make the process easier. Here's how to get started:

Step	Process
1. Choose SBOM tools	Start with tools that fit your workflow. Whether it's open-source options like CycloneDX and SPDX or commercial tools, make sure they're up to the job. Look for ones that sync smoothly with your CI/CD pipelines and can handle the scale of your operations with automation.
2. Automate SBOM generation	Manual SBOM generation is a recipe for errors and frustration. Automate it instead. Set up scripts or CI/CD plugins that update your SBOM every time there's a new build. It keeps things current and saves your team time and effort.
3. Track and update software components	Software isn't static—it evolves. Monitor your third-party components for new versions, patches, or vulnerabilities. Make reviewing and updating your SBOM a regular habit. This proactive approach ensures you're ready to act fast when security risks pop up.
4. Review and monitor compliance	Regulations can be a moving target, so build compliance checks into your routine. Does every component meet licensing and security standards? Conduct audits to double-check. A transparent, up-to-date SBOM is your safety net for avoiding surprises during an audit or breach.

Wiz's approach to SBOM security

Wiz Code enhances SBOM (Software Bill of Materials) security by providing comprehensive visibility into the components and dependencies within your software. With real-time scanning and automated security checks, Wiz Code ensures that vulnerabilities within third-party libraries and open-source components are identified and mitigated early.

Some advantages of agentless SBOM deployment include:

- **Flexibility and simplicity:** Dedicated agents are add-on services that consume resources and need ongoing maintenance, adding to a sky-high maintenance overhead. Agentless SBOM deployment also lowers cost and eliminates agent-to-OS compatibility concerns..
- **Instant and complete visibility:** Agents must be installed on each subsystem in the software stack. An agentless SBOM gives you a complete view of your applications' components—from the open-source libraries in use to the package and nested dependencies—within minutes, without blind spots.
- **SBOM search:** Search and quickly locate specific OS and open-source packages across cloud environments. This capabiliity is particularly timely given recent critical vulnerabilities found in widely used libraries like xz-utils.
- **Always up to date:** Agents require manual installation which can be error-prone, while an agentless approach allows you to generate up-to-date SBOMs without manual intervention.