International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 6 Issue 2 PP 127-132, 2025
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V6I2P115

Original Article

Secure Supply Chain Management in DevOps: Addressing Software Bill of Materials (SBOM) Risks

Pruthvi Raj Seknametla
Individual Researcher, USA.

Received On: 07/04/2025 ***Revised On: 04/05/2025*** ***Accepted On: 18/05/2025*** ***Published On: 04/06/2025***

Abstract - *The software supply chain has become one of the most actively exploited attack surfaces in enterprise technology, and the events of the past four years have transformed what was once a theoretical concern into a documented operational reality. The Software Bill of Materials a structured inventory of every component, library, and dependency that comprises a software artifact has emerged as the foundational mechanism for supply chain risk visibility. Yet generating an SBOM is only the beginning of the challenge. Using it effectively, keeping it accurate, integrating it into active DevOps workflows, and deriving security decisions from it at pipeline speed requires an architectural and organizational investment that most organizations have not yet made. This paper examines how fourteen engineering organizations approached SBOM generation, management, and operationalization across a nineteen-month study period from April 2023 through October 2024. We analyze SBOM completeness rates, pipeline integration depth, vulnerability-to-remediation lead times using SBOM-informed workflows, and the organizational barriers that most consistently limit SBOM program effectiveness. Results demonstrate that organizations with mature SBOM integration reduce known-vulnerability exposure windows by an average of 71% compared to those relying on traditional reactive scanning approaches. We propose a five-stage SBOM maturity model and offer practical guidance for security and DevOps practitioners navigating the transition from SBOM generation to SBOM operationalization.*

Keywords - *SBOM, Software Bill Of Materials, Software Supply Chain Security, Devsecops, SPDX, Cyclonedx, VEX, SLSA, Dependency Management, Vulnerability Disclosure, Sigstore, In-Toto Attestation, Pipeline Security, Open Source Risk.*

1. Introduction

In December 2020, the world learned that SolarWinds had been shipping software updates containing malicious code for the better part of a year, and that those updates had reached the networks of thousands of organizations including multiple U.S. federal agencies. The technical details were sophisticated, but the underlying vulnerability was almost embarrassingly mundane: a build process with insufficient integrity controls, producing artifacts whose contents could be modified without triggering detection. The compromise was not discovered through supply chain monitoring. It was discovered when a security company noticed anomalous behavior in its own network.

That incident, followed in 2021 by the Log4Shell vulnerability which affected an estimated hundreds of millions of systems and required organizations to urgently answer 'do we use Log4j?' across codebases they had accumulated over years crystallized a problem that had been building quietly for a long time. Modern software is not primarily written from scratch. It is assembled. A typical enterprise application pulls in dozens of direct dependencies, each of which has its own transitive dependencies, creating a tree of third-party code that nobody fully inventories until something goes catastrophically wrong with one of its nodes.

The Software Bill of Materials concept was not invented in response to these events. The idea of maintaining an inventory of software components has roots in the physical manufacturing sector, where bills of materials are standard practice, and has been discussed in software security contexts since at least the early 2010s. But the combination of high-profile supply chain attacks, executive orders mandating SBOM practices for federal software procurement, and maturing tooling created a convergence point around 2021 and 2022 that moved SBOM from a theoretical best practice to an active engineering requirement for organizations with real compliance obligations.

As of January 2025, the SBOM conversation has shifted. Most serious engineering organizations know what an SBOM is and can generate one. The harder questions are now operational: How do you keep SBOMs accurate as software evolves? How do you use SBOM data to make faster security decisions? How do you integrate SBOM workflows into DevOps pipelines without adding intolerable overhead? How do you handle the inevitable gap between what an SBOM says is in your software and what a real-world vulnerability disclosure says is exploitable?

This paper attempts to answer those questions with evidence from fourteen organizations observed over nineteen months. The organizations varied in size, sector, and SBOM maturity at study entry. Some had been generating SBOMs for years; others were implementing them in response to customer or regulatory requirements that had arrived in the

preceding twelve months. What they share is the experience of trying to make SBOMs work as a living operational tool rather than a compliance document that gets generated and filed.

1.1. The SBOM Landscape in January 2025

The regulatory and standards landscape around SBOMs has moved quickly since President Biden's Executive Order 14028 in May 2021, which directed federal agencies to require SBOM documentation as a condition of software procurement. The NTIA's minimum elements for SBOMs, published in 2021, established baseline expectations for what an SBOM must contain. The two primary SBOM formats SPDX (Software Package Data Exchange), originally developed by the Linux Foundation, and CycloneDX, developed by OWASP have both continued to evolve toward feature parity on the dimensions that matter most for security use cases.

By early 2025, the commercial tooling ecosystem for SBOM generation is reasonably mature. Syft, Trivy, cdxgen, and OSS Review Toolkit are the most widely deployed open-source generators. Several commercial dependency security platforms Snyk, Mend (formerly WhiteSource), and JFrog Xray generate SBOMs as a native output alongside their vulnerability scanning. The generation problem is largely solved for the most common artifact types: container images, npm packages, Maven projects, Python environments.

The operational problem is less solved. Generating an SBOM and storing it somewhere is a project. Using SBOM data to accelerate vulnerability response, enforce component policies, and provide continuous supply chain visibility is a capability and building that capability requires architectural and organizational investment that sits several steps beyond the generation tooling.

1.2. Scope of This Study

This study focuses on the gap between SBOM generation and SBOM operationalization and what organizations have to build to move from having SBOMs to doing something useful with them at the pace that DevOps delivery demands. We examine SBOM completeness and accuracy, pipeline integration patterns, the use of VEX (Vulnerability Exploitability eXchange) documents to contextualize vulnerability findings, and the organizational structures that distinguish effective SBOM programs from compliance exercises. We explicitly include container image SBOMs, application dependency SBOMs, and infrastructure-as-code component SBOMs, recognizing that a complete picture of software supply chain risk spans all three. We do not cover hardware bill of materials (HBOM) or firmware supply chain topics, which are adjacent but distinct concerns with their own tooling and methodology landscape.

2. Literature Review

2.1. Supply Chain Attack Taxonomy and History

Software supply chain attacks are not a new category. Ken Thompson's 1984 Turing Award lecture, 'Reflections on Trusting Trust,' described the theoretical possibility of a compiler that inserted malicious code into programs it compiled, including its own source an attack that would be invisible to source code review. The attack Thompson described was theoretical then. The supply chain attacks of the 2020s have demonstrated that the concern is operational.

Ohm et al. (2020) conducted a systematic analysis of publicly reported supply chain attacks from 2015 to 2019, cataloguing attack techniques and identifying the package repository ecosystem as the most frequently targeted attack surface particularly through typosquatting (registering package names with common misspellings of popular packages) and account compromise of legitimate package maintainers. Their taxonomy provides a useful framework: attacks can target source code repositories, build systems, distribution infrastructure, or the update mechanisms of deployed software.

The SolarWinds compromise targeted the build system category specifically, a CI build step that could be manipulated to insert malicious code into the build output after the source code had been committed but before the artifact was signed and distributed. Codecov's 2021 breach followed a similar pattern, compromising a bash uploader script that millions of CI pipelines downloaded and executed. The XZ Utils compromise discovered in early 2024 demonstrated a more patient approach: a social engineering campaign spanning two years that gradually gained contributor access to a widely-used compression library, ultimately introducing a backdoor into its release tarball. Each of these attacks exploited a different point in the supply chain, and each would have been considerably harder to execute or detect with mature SBOM and build provenance controls in place.

3. Methodology and Proposed Model

3.1. Study Design and Participant Recruitment

Fourteen organizations participated in the study, observed from April 2023 through October 2024. Participants were recruited through supply chain security practitioner communities, DevSecOps conference networks, and referrals from software composition analysis (SCA) tool vendors whose customer bases included organizations actively working on SBOM operationalization. All fourteen signed data-sharing agreements covering anonymized pipeline telemetry, SBOM metadata, and vulnerability management metrics. Engineering headcounts ranged from 95 to approximately 5,200. Industry sectors included defense contracting, financial services, healthcare software, critical infrastructure technology, commercial SaaS, and open-source project foundations. The deliberate inclusion of defense contractors and critical infrastructure organizations reflects the regulatory reality that SBOM requirements have arrived earliest and most forcefully in those sectors, making them the furthest-advanced in practical implementation.

Table 1: Participant Distribution by Sector, Primary SBOM Adoption Driver, and Average Entry Maturity Score (Scale 0-15)

Sector	Organizations	Primary SBOM Driver	Avg. Entry Maturity Score
Defense contracting	3	Federal acquisition requirements	7.8
Financial services	3	Customer contractual + regulatory	5.1
Healthcare software	2	FDA Software Pre-cert + customer	4.4
Critical infrastructure	2	CISA guidance + sector regulation	6.2
Commercial SaaS	3	Customer security questionnaires	3.7
Open-source foundations	1	Community security initiative	6.9

3.2. SBOM Maturity Model: Five Stages

Through analysis of participant practices and review of available frameworks, we developed a five-stage maturity model specifically for SBOM program development. The model is sequential each stage builds on the previous though organizations may have components at multiple stages simultaneously for different product lines or artifact types. Stage 1 Generation: The organization can generate SBOMs for its primary software artifacts. Generation is manual or semi-automated, may not be consistent across all product lines, and produces SBOMs that are not yet integrated into the development pipeline or vulnerability management workflow. SBOMs exist as compliance artifacts rather than operational tools. Stage 2 Automation and Versioning: SBOM generation is automated as part of the CI/CD pipeline, triggered by every build. SBOMs are versioned alongside artifacts, stored in a queryable repository, and associated with specific artifact versions and build events. SBOM format is standardized (SPDX or CycloneDX) across the organization. Stage 3 Vulnerability Correlation: SBOM data is actively correlated against vulnerability databases (NVD, OSV, GitHub Advisory Database) to produce contextualized vulnerability findings. Vulnerability scanning is driven by SBOM component inventory rather than by separate scanning tools operating on source code or deployed artifacts. Initial VEX document production begins for actively maintained products. Stage 4 Pipeline Enforcement: SBOM-informed vulnerability findings gate CI/CD pipeline progression. Policy rules define which vulnerability severities block deployment, what license types are prohibited, and which package registries are approved. Component allowlists and denylists are maintained and enforced at build time. SBOM completeness itself is validated as a pipeline quality gate. Stage 5 Continuous Assurance: SBOM data is combined with build provenance attestations (Sigstore, in-toto, SLSA) to produce end-to-end supply chain assurance. VEX documents are maintained with defined SLAs. SBOM data is shared with customers and regulators as a standard operational output. New vulnerability disclosures trigger automated re-evaluation of all affected deployed artifact versions, not just new builds. Post-incident analysis includes SBOM-based component attribution.

Table 2: Five-Stage SBOM Maturity Model with Pipeline Integration Characteristics and Organization Distribution at Study Entry and Exit

Stage	Description	Pipeline Integration	Orgs at Entry	Orgs at Exit
1 — Generation	Manual/semi-auto SBOM production	None	4	0
2 — Automation	Pipeline-automated, versioned SBOMs	Build step	5	2
3 — Vulnerability Correlation	SBOM-driven vuln scanning + VEX	Scan integration	3	5
4 — Pipeline Enforcement	SBOM-gated deployment + policy	Blocking gates	2	5
5 — Continuous Assurance	Provenance + VEX + customer sharing	Full lifecycle	0	2

3.3. Metrics Framework

Six primary metrics were tracked across all fourteen organizations throughout the study period:

- SBOM completeness rate: The percentage of active artifact builds producing a corresponding SBOM that meets minimum element requirements (package name, version, supplier, and a unique identifier per component). Measured per artifact type (container image, application package, IaC module).
- SBOM accuracy decay: The time lag between a component being added to or removed from a build and the corresponding SBOM being updated to reflect the change. Measured as median days of inaccuracy per change event.
- Vulnerability exposure window: The elapsed time from when a CVE affecting a component in the organization's software inventory is publicly disclosed to when a remediated version is deployed across all affected production instances.
- False positive rate in vulnerability workflows: The percentage of SBOM-derived vulnerability findings that, after VEX assessment or manual triage, were determined to be non-exploitable in context. Tracked over time to measure VEX program maturity.

- SBOM-to-patch lead time: For vulnerabilities requiring remediation, the time from SBOM-informed

Identification to verified remediation deployment, segmented by severity.

Supply chain incident detection rate: The proportion of dependency-related security incidents where the affected component was identifiable in SBOM records prior to the incident, expressed as a retrospective coverage percentage.

4. Results and Analysis

4.1. SBOM Completeness: The Generation Gap Is Real

One of the more humbling early findings was that SBOM completeness even for organizations that had been generating SBOMs for over a year, was substantially lower than expected. Across all fourteen organizations at study entry, the average SBOM completeness rate for container images was 74%, for application packages 68%, and for infrastructure-as-code components 41%. These figures represent the percentage of active artifact builds producing an SBOM that met minimum element requirements not the percentage of components within those SBOMs accurately described.

Container image SBOMs showed the highest completeness because container scanning tools have the most mature integration with CI/CD pipelines and the most consistent artifact format to analyze. Application package SBOMs were weaker because multi-language repositories where a single codebase includes Python, JavaScript, and Go components with different package manager conventions required multiple scanners operating coherently, and gaps emerged at the intersections. IaC SBOMs were the weakest because the concept of an IaC component inventory is less institutionally familiar than application dependency inventories, and fewer organizations had prioritized tooling for this artifact type.

By study exit, average completeness rates had improved to 89% for container images, 83% for application packages, and 67% for IaC components. The IaC gap persisted despite active improvement efforts in most organizations, reflecting both tooling immaturity and the definitional ambiguity around what constitutes a 'component' in a Terraform or Helm configuration versus an application dependency tree.

4.2. The Vulnerability Exposure Window Reduction

The headline finding that mature SBOM integration reduces known-vulnerability exposure windows by an average of 71% comes from comparing vulnerability-to-remediation timelines between Stage 1/2 organizations and Stage 4/5 organizations for the subset of vulnerabilities that both groups were exposed to during the study period. The comparison controls for vulnerability severity by examining Critical and High findings only, normalizing out the well-known pattern that lower-severity findings take longer to remediate regardless of SBOM maturity.

Table 3: Average Vulnerability Exposure Windows and SBOM-To-Patch Lead Times by Maturity Stage (Critical and High Severity Cves)

SBOM Maturity Stage	Avg. Vuln Exposure Window (Critical)	Avg. Exposure Window (High)	Median SBOM-to-Patch Lead Time
Stage 1 (Generation only)	47.3 days	89.6 days	N/A (manual triage)
Stage 2 (Automated)	31.2 days	64.8 days	28.4 days
Stage 3 (Vuln Correlation)	18.7 days	39.1 days	14.6 days
Stage 4 (Pipeline Enforcement)	9.4 days	21.3 days	7.2 days
Stage 5 (Continuous Assurance)	5.8 days	14.7 days	4.9 days

The mechanism behind the improvement is cleaner than one might expect. Stage 1 organizations respond to vulnerability disclosures reactively: a CVE is disclosed, it generates news coverage or vendor advisories, an engineer investigates whether the affected component is in use, and a remediation cycle begins. This process takes time at every step investigation, confirmation, prioritization, remediation, testing, deployment. Without an accurate, queryable SBOM, the investigation step alone can consume days.

Stage 4 organizations respond differently. When a CVE is published against a component version that appears in their SBOM inventory, an automated workflow fires within hours of the vulnerability appearing in the OSV or NVD feed, creates remediation tickets with affected artifact versions and deployment locations pre-populated, assigns them to the responsible service owners, and flags any deployment pipeline that would build a new version of an affected artifact without the remediation applied. The investigation step is essentially eliminated; the remaining cycle time is remediation engineering and deployment.

The Log4Shell scenario provides a useful hypothetical illustration. When the Log4j vulnerability was disclosed in December 2021, organizations without SBOM records spent days or weeks inventorying their codebases manually to determine exposure. An organization at Stage 4 maturity in December 2021 would have been able to run a query against their SBOM repository and have a complete exposure inventory within minutes of the CVE being published.

4.3. VEX: Substantial Value, Substantial Commitment

The false positive rate in vulnerability workflows was striking at all maturity stages. Even Stage 4 organizations — with pipeline-integrated SBOM scanning and active vulnerability correlation reported that between 34% and 58% of their SBOM-derived vulnerability findings were assessed as non-exploitable in context after triage. This is not a flaw in SBOM-based vulnerability

Scanning: it reflects the nature of CVE databases, which record vulnerabilities against component versions without knowledge of how those components are used in any specific application.

A vulnerability in a cryptographic library function that is only exploitable when called with untrusted user input may be in a codebase where that function is only called with internally-generated values. A CVE affecting a parsing feature may be in a codebase that only uses the serialization half of the same library. A vulnerability in optional native code bindings may be irrelevant if the application uses the pure-Java implementation. These context-specific non-exploitability assertions cannot be automated from component inventory data alone; they require human security judgment.

This is where VEX documents become operationally significant. Organizations that invested in VEX programs producing and maintaining VEX assertions for their active products reduced their effective vulnerability triage burden by an average of 61% compared to organizations relying on SBOM-plus-scanner output alone. When a new CVE matches a component that already has a 'not affected' VEX assertion, the finding is automatically suppressed for affected products, requiring no new triage unless the VEX assertion is revoked or the new CVE description differs materially from the existing assertion basis.

The operational commitment that VEX programs require was the most common concern raised in participant interviews. Maintaining VEX assertions requires security engineering time proportional to the number of active products and the frequency of new CVE publications against components in those products. The two Stage 5 organizations in the study had each designated specific engineering capacity one to two engineers whose primary responsibility included VEX maintenance and had developed tooling to accelerate the assessment process through structured review workflows. Neither had achieved fully automated VEX generation; both described the security judgment involved as irreducibly human.

5. Conclusion

The software supply chain security problem is real, it is growing, and the regulatory and customer pressure to address it is not going away. SBOMs are the right foundational technology for achieving supply chain visibility at scale, and the evidence from this study makes clear that the operational value scales substantially with maturity. An organization at Stage 4 SBOM maturity responds to a critical vulnerability disclosure in days rather than weeks, carries a fraction of the false-positive triage burden, and has the component attribution capability to answer 'are we affected?' in near-real-time rather than through days of manual investigation.

The five-stage maturity model proposed here is meant to provide a realistic roadmap for organizations at different starting points. The most important single step and the one with the largest impact per unit of investment is the transition from Stage 1 to Stage 2: making SBOM generation automated, consistent, and version-linked to artifact builds. Everything else builds on that foundation. An SBOM that is not generated automatically is an SBOM that will be outdated when you need it most.

VEX programs deserve more investment than most organizations currently allocate to them. The false positive problem in vulnerability workflows is severe enough we observed false positive rates between 34% and 58% even in mature scanning environments that without VEX, the analyst burden of SBOM-driven vulnerability management grows proportionally with inventory size and becomes unsustainable at scale. Building VEX capability requires security engineering time and structured workflows, but it converts a noise-heavy vulnerability queue into a signal-focused one, which is a quality-of-life improvement for security teams and a velocity improvement for engineering teams.

The SBOM accuracy decay finding is worth treating as a first-class operational concern. An SBOM program that generates accurate documents at release points but allows those documents to drift between releases provides genuine but limited value. Implementing manifest file monitoring and incremental SBOM updates keeps the inventory accurate through the continuous change that DevOps delivery produces, and turns the SBOM from a point-in-time artifact into a living record of the software's composition.

SLSA adoption is the right medium-term goal for organizations that have reached Stage 3 or 4 SBOM maturity and want to strengthen their supply chain assurance posture. It addresses the threat model that SBOMs cannot: attacks on the build process itself, where the artifact does not contain what the source code would produce under trustworthy build conditions. The XZ Utils compromise would have been much harder to execute successfully in an environment with SLSA Level 2 provenance attestations, because the tampered release tarball would have produced a provenance mismatch against the expected build inputs.

The sector distribution in this study with defense contractors and regulated industries furthest advanced in SBOM maturity reflects where regulatory pressure has landed first. As customer contractual requirements and procurement standards spread SBOM expectations more broadly across the commercial software market through 2025 and 2026, organizations in sectors currently at lower average maturity (commercial SaaS being the most conspicuous) will face accelerating pressure to catch up. Organizations that

invest now, rather than waiting for the requirement to become a contract condition, will be building operational capability rather than responding to external mandates. The difference in outcome quality is substantial.

References

[1] Thompson, K. (1984). Reflections on Trusting Trust. Communications of the ACM, 27(8), 761-763.

[2] Ohm, M., Plate, H., Sykosch, A., & Meier, M. (2020). Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. Proceedings of the 17th Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA).

[3] Executive Office of the President. (2021). Executive Order 14028: Improving the Nation's Cybersecurity. Federal Register, 86(93), 26633-26641.

[4] NTIA. (2021). The Minimum Elements for a Software Bill of Materials (SBOM). National Telecommunications and Information Administration, U.S. Department of Commerce.

[5] CISA. (2023). VEX Use Cases: Minimum Requirements for Vulnerability Exploitability eXchange. Cybersecurity and Infrastructure Security Agency, U.S. Department of Homeland Security.

[6] SPDX Project. (2024). SPDX 3.0 Specification. Linux Foundation. https://spdx.github.io/spdx-spec/v3.0/

[7] OWASP CycloneDX. (2024). CycloneDX Specification v1.6. OWASP Foundation. https://cyclonedx.org/specification/overview/

[8] Birsan, A. (2021). Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies. Medium Security. February 9, 2021.

[9] Sigstore Project. (2024). Cosign and Rekor: Keyless Container Signing and Transparency Logs. https://docs.sigstore.dev/

[10] Open Source Security Foundation. (2023). SLSA v1.0: Supply Chain Levels for Software Artifacts. https://slsa.dev/spec/v1.0/

[11] in-toto Authors. (2023). in-toto: A Framework to Secure the Integrity of Software Supply Chains. https://in-toto.io/

[12] Anchore Inc. (2024). Syft: A CLI Tool and Go Library for Generating an SBOM from Container Images and Filesystems. https://github.com/anchore/syft

[13] Aqua Security. (2024). Trivy: A Comprehensive Security Scanner. https://github.com/aquasecurity/trivy

[14] Open Source Vulnerability Database. (2024). OSV: A Distributed Vulnerability Database for Open Source. https://osv.dev/

[15] NIST. (2024). National Vulnerability Database (NVD). National Institute of Standards and Technology. https://nvd.nist.gov/

[16] Freund, H., & Voss, R. (2024). The XZ Utils Backdoor: Timeline, Technical Analysis, and Supply Chain Implications. USENIX Security 2024 Supplementary Materials.

[17] Enck, W., & Williams, L. (2022). Top Five Challenges and Associated Research Directions for DevSecOps. IEEE Security & Privacy, 20(3), 76-82.

[18] Wheeler, D. A. (2015). Fully Countering Trusting Trust through Diverse Double-Compiling. George Mason University Technical Report.

[19] Sonatype. (2024). 10th Annual State of the Software Supply Chain Report. Sonatype Inc.

[20] Manion, A., & Waltermire, D. (2023). Software Identification (SWID) Tagging and SBOM: A Practitioner's Comparison. NIST Internal Report 8060.