Dependency Tracking and SBOM: Strengthening Software Supply Chain Security

1. Introduction

The software ecosystem has become a complex web of interdependencies. Modern applications are rarely written from scratch; they are assembled from a vast array of third-party components, open-source libraries, and proprietary modules. While this accelerates innovation, it also introduces systemic risks.

Recent high-profile incidents such as the SolarWinds compromise, the Log4j vulnerability, and the XZ Utils/liblzma backdoor demonstrate how weaknesses in software supply chains can have global repercussions. Even seemingly small anomalies—like an SSH login delay that revealed the XZ backdoor—can signal devastating compromises.

The Software Bill of Materials (SBOM) has emerged as a critical tool to address these risks. Much like a list of ingredients in packaged food, an SBOM provides transparency into what goes into software. By enabling visibility, traceability, and accountability, SBOMs help organizations identify vulnerabilities faster, comply with regulatory frameworks, and strengthen trust across the digital supply chain.

This whitepaper explores the value of SBOMs, reviews current standards and tools, and provides best practices for effective adoption.

2. Dependency Risks in Modern Software

Dependencies are both the lifeblood and the Achilles' heel of modern software. Widely used open-source components are often maintained by small, underfunded teams, yet they become the foundation of mission-critical systems.

Common attack vectors include:

1. Dependency Confusion

Dependency confusion exploits the way package managers resolve dependencies when the same package name exists in both **private/internal repositories** and **public package registries** (e.g. npm, PyPI, Maven Central).
If resolution order is misconfigured, the build system may **pull a malicious public**

package instead of the intended internal one, executing attacker-controlled code during build or runtime.

Typical Conditions

- Internal packages use generic names (e.g. `utils`, `core-lib`)
- Build pipelines allow fallback to public registries
- No namespace scoping or package pinning
- CI/CD systems have internet access and credentials

In 2021, security researcher **Alex Birsan** demonstrated dependency confusion attacks against multiple Fortune 500 companies.
He identified internal package names from public metadata and published malicious versions to public registries with higher version numbers.
When company CI pipelines resolved dependencies, they downloaded the attacker's package, executing code that exfiltrated environment variables and credentials.

- Alex Birsan, *"Dependency Confusion"* (2021)
  https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

2. Typosquatting

Typosquatting targets human error by publishing packages with **names that closely resemble popular dependencies**, relying on misspellings or visual similarity.
Developers may accidentally install the malicious package locally, or it may be introduced via copy-paste or automated scripts.

Typical Conditions

- Manual dependency installation
- No allowlist or integrity verification
- Heavy reliance on community packages
- Lack of automated dependency review

Multiple malicious Python packages such as `reqeusts`, `urlib3`, and `django-rest-frameworks` were published to **PyPI**, mimicking widely used libraries.
Once installed, these packages executed malicious payloads including credential harvesting and backdoors.

In enterprise environments, such packages may propagate quickly via internal mirrors or shared requirements files.

- PyPI Security Reports & Malicious Package Takedowns
  https://blog.python.org/2023/09/pypi-security-malware.html

3. Poisoned Packages

Poisoned packages contain **malicious code intentionally embedded inside a legitimate-looking dependency**.
Unlike typosquatting, these packages may be:

- Popular and widely used
- Recently updated by compromised maintainers
- Modified through social engineering or long-term trust building

This class of attack is especially dangerous because it often **bypasses traditional security scanning**.

Typical Conditions

- High trust in open-source maintainers
- Automated updates without review
- No SBOM-based impact analysis
- Limited runtime behavior monitoring

In 2024, **XZ Utils (liblzma)** was compromised through a **targeted supply chain backdoor**.
An attacker spent years building trust within the project, eventually introducing obfuscated malicious code that enabled SSH authentication bypass on affected systems.

The attack was only discovered because a developer noticed abnormal SSH performance during login.

- Sonatype, *"CVE-2024-3094: XZ Utils Backdoor"* https://www.sonatype.com/blog/cve-2024-3094-the-targeted-backdoor-supply-chain-attack-against-xz-and-liblzma

The risks are growing. In 2024, Sonatype's 10th Annual State of the Software Supply Chain Report observed a **156 % year-over-year increase in malicious open-source packages**, identifying **512,847 malicious packages** over the preceding year — a dramatic escalation in software supply chain risk across ecosystems such as npm, PyPI, and others. Surprisingly, three years after Log4Shell was exposed, 13% of downloads remain vulnerable.

The reality is that software publishers are increasingly unable to keep pace with CVE remediation, with many vulnerabilities taking more than **500 days** to resolve. We have reached a tipping point: software complexity has grown to such a degree that a lack of transparency is no longer manageable—it is a critical risk. These examples underscore the urgent need for proactive dependency tracking and SBOM adoption. Traditional security

controls are no longer sufficient; developers and automated build pipelines have become primary targets.

All figures above are here: https://www.infosecurity-magazine.com/news/156-increase-in-oss-malicious

3. SBOM: Foundations and Value

A **Software Bill of Materials (SBOM)** is a **structured, nested inventory of software components** that describes the libraries, modules, and dependencies used to build a software product. It enables software producers and consumers to identify affected components, assess exposure to vulnerabilities, and take appropriate remediation actions.

The value of a Software Bill of Materials (SBOM) rests on three foundational pillars: **transparency, traceability, and trust**. Together, they address the structural weaknesses of modern software supply chains and enable organizations to manage risk at scale.

1. Transparency

Knowing what is inside your software

Transparency is the most immediate and tangible benefit of an SBOM. It provides **visibility into the components that make up a software product**, including open-source and proprietary libraries, versions, licenses, and suppliers.

In modern software ecosystems, organizations often deploy applications without fully understanding what they contain. Transitive dependencies, auto-generated builds, and rapid CI/CD pipelines make it nearly impossible to maintain an accurate mental model of software composition without automation.

Without transparency, organizations cannot reliably answer basic questions such as:

- *Are we affected by this newly disclosed vulnerability?*
- *Which products contain this component?*
- *Do we rely on unmaintained or end-of-life libraries?*

In this sense, transparency is a **prerequisite for all downstream security actions**. You cannot secure what you cannot see.

2. Traceability

Understanding where risks come from and where they propagate

Traceability builds on transparency by allowing organizations to **map vulnerabilities, risks, and incidents to specific components and products** across the software lifecycle.

An SBOM enables traceability by linking:

- Components → versions → suppliers
- Vulnerabilities (CVEs) → affected dependencies
- Dependencies → applications, services, or devices
- Software artifacts → deployed environments

This capability is critical during security incidents. When a new vulnerability such as Log4Shell or the XZ backdoor is disclosed, the primary question is not *"Is this bad?"* but *"Where are we exposed?"*

Without traceability, incident response becomes manual, slow, and error-prone—often relying on ad-hoc searches, tribal knowledge, or incomplete inventories.

3. Trust

Enabling confidence across the supply chain

Trust is the most strategic and long-term pillar of SBOM adoption. It is not blind trust, but **verifiable trust**—grounded in evidence, provenance, and accountability.

An SBOM contributes to trust by:

- Making software composition **explicit rather than implicit**
- Enabling **supplier accountability** and contractual enforcement
- Supporting **independent verification** of security claims
- Allowing cryptographic signing and provenance validation (e.g., Sigstore)

From a governance perspective, SBOMs enable organizations to:

- Ask informed questions of suppliers
- Verify that security requirements are met
- Enforce obligations under regulations such as CRA and **NIS2**
- Build confidence with customers, partners, and regulators

In a world of distributed development and global dependencies, **trust without verification is no longer viable**. SBOMs provide the foundation for moving from implicit trust to **measurable, auditable trust**.

According to Gartner, organizations that use SBOMs to manage open-source dependencies reduce their mean time to remediate (MTTR) by 264 days compared to those that do not.

SBOMs are a key enabler of both regulatory and contractual compliance. Under the **EU Cyber Resilience Act (CRA)**, manufacturers are required to identify, document, and manage vulnerabilities throughout the entire lifecycle of a product, including obligations for **coordinated vulnerability disclosure and timely remediation**. SBOMs provide the necessary visibility to meet these requirements in practice.

In parallel, the **NIS2 Directive** reinforces these expectations by requiring suppliers to implement cybersecurity risk management measures, grant audit rights, notify incidents without undue delay, and handle vulnerabilities proactively. Without an SBOM, meeting these obligations in a consistent and verifiable manner becomes unrealistic.

4. SBOM Standards and Formats

Multiple SBOM standards have emerged, each serving different use cases. Below is a comparison of their applicability to IoT apps (embedded, long lifecycle) and classic SaaS apps (cloud-native, web).

Standard	IoT/embedded App (embedded, long lifecycle)	Classic SaaS App (cloud-native, web)
SPDX (ISO/IEC 5962:2021)	Strong for compliance and archival; fits regulated IoT (medical devices, automotive). Ensures SBOM longevity across lifecycle.	Good for SaaS compliance reporting and procurement. Provides ISO-recognized format for audits under NIS2.
CycloneDX (OWASP)	Useful for firmware + hardware BOMs; supports security attestations. Fits IoT where hardware/software mix matters.	Highly suited for SaaS dependency tracking in CI/CD. Integrates well with DevSecOps pipelines and vulnerability management.
SWID (ISO/IEC 19770-2)	Lightweight tagging of installed software. Practical for constrained devices, but limited for full IoT firmware transparency.	Less relevant for SaaS—more for asset management than dependency tracking.

This comparison shows that **no single SBOM standard is universally optimal**; the right choice depends on the system architecture, lifecycle constraints, and regulatory context.

SPDX is particularly well suited for **IoT and embedded products operating in regulated environments**, where long lifecycles, formal compliance, and auditability are key. Its ISO standardization and stability make it a strong fit for products that must be maintained, supported, and evidenced over many years.

CycloneDX, by contrast, aligns best with **cloud-native SaaS applications and software-centric components**, including the backend or frontend layers of IoT systems. Its tight integration with CI/CD pipelines and vulnerability management tooling makes it highly effective for fast-moving development environments where continuous dependency tracking and rapid remediation are required.

SWID plays a complementary role, offering lightweight software identification for asset management, but it is insufficient on its own for dependency-level security or vulnerability analysis.

In practice, **hybrid approaches are often the most effective**: SPDX for product-level compliance and lifecycle management, and CycloneDX for operational security, DevSecOps, and vulnerability response. This layered strategy reflects the reality of modern systems, where regulated devices and agile software components coexist within the same product ecosystem.

5. SBOM Tools Comparison (IoT vs SaaS Context)

Comparison Criteria

- **Type**: Open source / Commercial
- **IoT / Embedded suited**: Firmware, long lifecycle, OT/ICS, constrained environments
- **Classic SaaS suited**: Cloud-native apps, CI/CD, fast iteration
- **Core capabilities**:
    - SBOM generation
    - Dependency management
    - Vulnerability triage / prioritization
    - Reachability / exploitability analysis
    - Code-level remediation guidance
    - Asset / product lifecycle management

Tool	Type	IoT / Embedded suited	Classic SaaS suited	Core Capabilities
CycloneDX	Open source (standard)	⚠️ Partial (firmware + HW/SW BOM support)	✅ Yes	SBOM specification, dependency metadata, vulnerability references (no scanning itself)
SPDX Tools	Open source (standard)	✅ Strong (regulated, long lifecycle)	⚠️ Moderate	SBOM creation & exchange, compliance-focused, limited vulnerability triage
OWASP Dependency-Track	Open source	⚠️ Possible (with SBOM input)	✅ Strong	Dependency inventory, CVE mapping, risk scoring, dashboards (no code changes)
Anchore Syft	Open source	⚠️ Partial (firmware/container images)	✅ Strong	SBOM generation from images, containers, filesystems
OWASP EMBA (FSTM)	Open source	✅ Strong	❌ Not suited	Firmware SBOM extraction, embedded Linux analysis, vulnerability detection

Takeaway (Open Source)

- **Best for IoT/embedded**: EMBA + SPDX
- **Best for SaaS**: CycloneDX + Syft + Dependency-Track

Open-source tools provide strong transparency, but weaker remediation & governance without commercial layers.

Tool	Type	IoT / Embedded suited	Classic SaaS suited	Core Capabilities
Black Duck (Synopsys)	Commercial	⚠️ Partial	✅ Strong	SCA, SBOMs, license compliance, vulnerability triage, policy enforcement
FOSSA	Commercial	❌ Limited	✅ Strong	Dependency management, SBOMs, license & vulnerability analysis, CI/CD
Endor Labs SBOM Hub	Commercial	❌ Limited	✅ Strong	SBOM mgmt, exploitability analysis, reachability, prioritization

Tool	Type	IoT / Embedded suited	Classic SaaS suited	Core Capabilities
Socket.dev	Commercial	❌ No	✅ Strong	Dependency behavior analysis, reachability, supply-chain attack detection
Aikido Security	Commercial	❌ No	✅ Strong	SCA, runtime insights, developer-centric vulnerability triage
Sonatype SBOM Manager	Commercial	⚠️ Partial	✅ Strong	SBOM mgmt, OSS governance, vulnerability intelligence, policy enforcement
RunSafe Security	Commercial	✅ Strong	❌ No	Binary hardening, firmware protection, exploit prevention (no classic SCA)
Cybeats SBOM Studio	Commercial	✅ Very strong	⚠️ Moderate	SBOM mgmt, OT/ICS asset inventory, lifecycle mgmt, CRA/NIS2 alignment

Takeaway (Commercial)

- **Best for SaaS**: Sonatype, Endor Labs, Socket, Aikido
- **Best for IoT / OT**: Cybeats, RunSafe

Commercial tools excel at **triage, prioritization, governance, and reporting.**

No single SBOM tool covers all needs.
Open-source tools are excellent for **generating SBOMs and establishing transparency**, while commercial platforms are typically required for **vulnerability triage, reachability analysis, governance, and regulatory reporting**.

- **IoT / Embedded products** benefit from **SPDX + EMBA + Cybeats / RunSafe**
- **Classic SaaS applications** benefit from **CycloneDX + Dependency-Track + Sonatype / Endor Labs**
- **Hybrid architectures** (IoT backend + SaaS frontend) should adopt **multiple SBOM formats and toolchains**

6. Best Practices for SBOM Adoption

To unlock the full value of SBOMs, organizations must treat them as **living security artifacts**, not static compliance documents. The following best practices reflect industry guidance from **Sonatype's SBOM Cheat Sheet** and the **OWASP Software Component Verification Standard (SCVS)**, and align with emerging regulatory expectations under NIS2 and the Cyber Resilience Act.

1. Generate SBOMs Automatically and Early

Make SBOMs a default output of your build process

SBOMs should be generated **automatically within CI/CD pipelines**, not manually or as an afterthought. Automation ensures consistency, reduces human error, and aligns with SCVS requirements for repeatable and verifiable software composition analysis.

2. Keep SBOMs Continuously in Sync

An outdated SBOM is a false sense of security

SBOMs must reflect the **exact software artifact deployed in production**. Any mismatch between code, dependencies, and SBOM undermines traceability and compliance.

3. Actively Correlate SBOMs with Vulnerability Intelligence

Visibility without action has no security value

SBOMs should be continuously correlated with vulnerability sources such as **NVD, OSV, and commercial threat intelligence feeds**. This enables rapid identification of affected components when new CVEs are disclosed.

4. Enforce SBOM Requirements Across the Supply Chain

SBOMs are as much a procurement tool as a security tool

Organizations should require SBOMs from suppliers and integrate them into **vendor risk management and procurement processes**, as recommended by both SCVS and NIS2.

5. Ensure Integrity and Provenance of SBOMs

Trust must be verifiable, not assumed

SBOMs themselves are security-critical artifacts and must be protected against tampering. Signing and verifying SBOMs ensures authenticity and supports non-repudiation.

6. Manage SBOMs Across the Full Product Lifecycle

SBOMs do not end at release

SBOM obligations extend throughout the **entire software lifecycle**, including maintenance, patching, and end-of-life—explicitly required under CRA vulnerability handling obligations.

7. Regulatory and Industry Drivers

SBOM adoption is no longer driven solely by best practice; it is increasingly **mandated by regulation**, with the **EU Cyber Resilience Act (CRA)** acting as a primary catalyst.

Under **Article 10 of the Cyber Resilience Act**, manufacturers are explicitly required to **identify, document, and address vulnerabilities throughout the entire lifecycle of a product**, including obligations for **coordinated vulnerability disclosure, timely remediation, and communication with users**. In practice, these obligations cannot be fulfilled without a clear and continuously maintained inventory of software components. SBOMs therefore become a foundational mechanism for CRA compliance, enabling manufacturers to determine whether disclosed vulnerabilities affect their products and to act accordingly.

Complementing the CRA, the **EU NIS2 Directive** reinforces supply chain accountability by requiring suppliers and service providers to implement cybersecurity risk management measures, notify significant incidents without undue delay, and grant audit rights. SBOMs support these obligations by providing traceability, auditability, and evidence of proactive vulnerability handling.

Beyond the EU, regulatory momentum is global. The **US Executive Order 14028** mandates SBOMs for federal software procurement, while the **US Food and Drug Administration (FDA)** requires SBOMs for medical devices as part of premarket and postmarket cybersecurity controls.

Together, these frameworks signal a clear shift: **transparency of software composition is becoming a regulatory expectation rather than an optional security enhancement**.

8. Conclusion

SBOMs are no longer optional—they have become a cornerstone of modern cybersecurity and a prerequisite for regulatory compliance. By adopting SBOMs, organizations gain essential visibility into their software supply chains, improve their ability to assess and remediate vulnerabilities, and build verifiable trust with customers, partners, and regulators.

Looking ahead, SBOMs will continue to evolve beyond static documentation. Advances such as automated and AI-assisted SBOM generation, continuous synchronization with CI/CD and runtime environments, improved interoperability between standards, and deeper integration with vulnerability intelligence will increasingly transform SBOMs into **active security controls**. In this model, SBOMs support real-time risk assessment, informed decision-making, and enforceable supply chain governance across the full product lifecycle.

The path forward is therefore both clear and unavoidable: integrate SBOMs into DevSecOps workflows, require them systematically from suppliers, and operationalize them as part of vulnerability management and compliance programs. In an environment of growing software complexity and regulatory scrutiny, **transparency, traceability, and trust are no longer aspirational principles—they are operational necessities**.

9. References

- Sysdig – SBOM 101: Software Bill of Materials: https://www.sysdig.com/blog/sbom-101-software-bill-of-materials
- Sonatype – CVE-2024-3094 Analysis: https://www.sysdig.com/blog/cve-2024-3094-detecting-the-sshd-backdoor-in-xz-utils
- OWASP – CycloneDX: https://owasp.org/www-project-cyclonedx
- Dependency Track: https://dependencytrack.org/
- Sigstore Project – https://www.sigstore.dev/
- Gartner – SBOM MTTR Reduction Report https://anchore.com/sbom/gartner-innovation-insights-sboms
- The New Stack – The 3S of Software Supply Chain Security https://thenewstack.io/the-3-ss-of-software-supply-chain-security-sboms-signing-slimming
- https://spdx.dev/tools/
- https://github.com/scriptingxss/owasp-fstm