LEVERAGING SBOMS THROUGHOUT THE ENTERPRISE SDLC

Drew Buttner and Robert Martin

May 21, 2026

MITRE

INTRODUCTION

A Software Bill of Materials (SBOM) is a structured document containing the details and supply chain relationships of the various components used in building a piece of software. SBOMs provide enterprises with a foundational capability for understanding, managing, and securing the software that powers modern operations. As organizations increasingly rely on complex software ecosystems composed of open-source libraries, third-party components, cloud services, and vendor-supplied applications, the ability to identify exactly what software is running in an enterprise environment has become essential for providing software supply chain assurance and performing effective cyber risk management.

CONFIDENCE IS OBTAINED VIA A SYSTEMATIC LOOK AT PROVENANCE AND PEDIGREE INFORMATION CAPTURED THROUGHOUT THE SDLC

An SBOM delivers a machine-readable inventory of software components, their origins, versions, and dependencies, enabling organizations to rapidly identify vulnerable or outdated components, accelerate incident response, improve patch and upgrade management, and strengthen software supply chain integrity [1] [2].

Beyond cybersecurity, SBOMs also support operational resilience, enterprise risk management, regulatory compliance, procurement transparency, and informed technology decision-making by reducing the uncertainty associated with opaque software supply chains.

Industry adoption continues to grow as SBOMs enable automation, improve visibility across interconnected systems, and provide organizations with actionable intelligence needed to reduce financial, operational, and reputational risk in an increasingly software-dependent world.

Today, the concept of an SBOM is widely understood, and the availability of them is becoming more prevalent. However, knowing how to make the most of the information contained within an SBOM, deciding on what is needed within an SBOM, and figuring out how to add relevant information throughout the in-house development of software remains elusive for most enterprises.

This report walks the reader through the Software Development Lifecycle (SDLC) and provides details for how an organization can create an SBOM, add information to it during each phase of the SDLC, and set up their tooling to take advantage of the SBOM. The hope is that the reader will be able to use these examples to establish their own practices to take advantage of what an SBOM can offer.

SBOM AND THE SDLC

At its core, an SBOM is a machine-readable document that captures information about the software it represents, enabling an automated understanding of the structure of the software and the 3rd party components it utilizes.

The more information presented in the SBOM, the more detailed and accurate the reasoning can be. Such reasoning is part of the Software Assurance discipline which attempts to capture the level of confidence that software is free from vulnerabilities, either intentionally designed into the

software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner, which includes 3rd party components [3].

Confidence in software is obtained via a systematic look at provenance and pedigree information captured throughout the SDLC including how the software was built, what tools were used, and what external components were included. Through automated analysis of the information in an SBOM, downstream users are able to understand the development practices followed by the software provider and use risk-based methodologies to calculate the potential harm from unseen problems (e.g., vulnerabilities) hidden in the software.

The Cybersecurity and Infrastructure Security Agency (CISA) — part of the U.S. Department of Homeland Security (DHS) — community efforts on software transparency defined six types of SBOMs to support the different information needed throughout the SDLC [4]. These types can be combined into a single SBOM document as necessary.

1) **Design** = SBOM of a planned software product and the expected 3rd party components it will depend upon (some of which may not yet exist).
2) **Source** = SBOM created directly from the development environment, source files, and included dependencies used to build a product artifact.
3) **Build** = SBOM generated during the process to create a releasable artifact (e.g., executable or package) collecting data such as source files, dependencies, build components, ephemeral data, and other SBOMs.
4) **Analyzed** = SBOM generated through analysis of build artifacts. (e.g., executables, packages, containers, and virtual machine images) Generally requires a variety of heuristics. In some contexts, this may also be referred to as a "reverse-engineered" SBOM.
5) **Deployed** = SBOM provides an inventory of software that is present on a system including dynamically linked libraries. This may be an assembly of other SBOMs that combines analysis of configuration options, and examination of execution behavior.

During development activities, tools such as Syft, Tern, ScanCode Toolkit, and the Microsoft SBOM Tool provide early visibility into components and licenses within Design and Source SBOMs.

As software is built, integrations with systems like Docker BuildKit and the Yocto Project use Build SBOMs to accurately reflect the final product.

Once deployed, management tools such as GUAC, KubeClarity, and SBOM Manager use the SBOMs to help organizations track vulnerabilities, validate deployed components, and maintain compliance in real time via Deployed SBOMs.

As a result, SBOMs are becoming essential enterprise capabilities for organizations seeking continuous visibility into software composition, operational exposure, and supply chain risk.

ENTERPRISE RISK MANAGEMENT

SBOMs not only document software composition, but also support ongoing risk-informed decision-making across the SDLC.

For software development teams, SBOMs improve software assurance and supply chain transparency. For software procurement teams, SBOMs support supplier evaluation, acquisition decisions,

and compliance verification. For software operation teams, SBOMs support vulnerability management, impact assessment, continuous monitoring, and operational risk reduction.

SBOMs provide organizations with machine-readable transparency into software dependencies, provenance, build environments, external components, and known vulnerabilities. This transparency enables organizations to make more informed risk-based decisions regarding software selection, deployment, operation, maintenance, and incident response.

While SBOMs are often discussed as technical artifacts used to describe software composition, their broader value is in enabling enterprise cyber risk management throughout the software lifecycle.

SBOM IMPLEMENTION

To demonstrate how an enterprise can leverage SBOM practices, this section will examine a hypothetical organization known as ACME. Details about the specific standards and tools used are provided to show how SBOMs are generated, verified, maintained, stored, and used at each phase in the SDLC as part of an overall risk management strategy.

Plan

The initial phase of a typical SDLC is the plan phase. The focus of this phase is the creation of detailed requirements to help ensure the software will meet expectations. These requirements play an important role later in the SDLC as they capture the intention of the software necessary for an accurate implementation and enable testing activities to know when something has gone awry. An SBOM is the perfect place to capture these requirements and make them available in a machine-readable format.

To accomplish this, the hypothetical organization ACME has mandated that all software programs create an SBOM as one of the first tasks. This initial SBOM is a foundation that is added to over the life of the software package being developed.

```
{
 "@context":"https://spdx.org/rdf/3.0.1/spdx-context.jsonld",
 "@graph":[
 {
  "type":"Organization",
  "spdxId":"urn:acme.dev:spdxdoc:org1",
  "creationInfo":"_:creationinfo-acme",
  "name":"ACME"
 },{
  "type":"CreationInfo",
  "@id":"_:creationinfo-acme",
  "specVersion":"3.0.1",
  "createdBy":["urn:acme.dev:spdxdoc:org1"],
  "created":"2025-08-13T10:28:00Z"
 },{
  "type":"SpdxDocument",
  "spdxId":" urn:acme.dev:spdxdoc",
  "creationInfo":"_:creationinfo-acme",
  "profileConformance":["core","software","build"],
  "rootElement":["urn:acme.dev:spdxdoc:sbom"]
 },{
  "type":"software_Sbom",
  "spdxId":"urn:acme.dev:spdxdoc:sbom",
  "creationInfo":"_:creationinfo-acme",
  "software_sbomType":["build"],
  "rootElement":["urn:acme.dev:spdxdoc:swpkg"]
 },{
  "type":"software_Package",
  "spdxId":"urn:acme.dev:spdxdoc:swpkg",
  "creationInfo":"_:creationinfo-acme",
  "name":"acme-sw-name",
  "software_packageVersion":"0",
  "software_primaryPurpose":"application"
 }]
}
```

The System Package Data Exchange™ (SPDX®) JSON above is the template that ACME uses. Specific guidance about the items in this SBOM is detailed in “Getting started writing SPDX 3” [5].

With the initial SBOM in place, the next step during the plan phase is to add specific requirements to it. These can be copied directly from a requirements document that is typically created for a software program.

Each individual requirement is added to the SBOM using an Annotation element with the name property set to "requirement" and the subject property set to the spdxId of the parent software package. Below is an example of such for the software that ACME is working on.

```
{
 "type":"Annotation",
 "spdxId":"urn:acme.dev:spdxdoc:req1",
 "creationInfo":"_:creationinfo-acme",
 "annotationType":"other",
 "name":"requirement",
 "contentType":"text/plain",
 "statement":"req id: descriptive text",
 "subject":"urn:acme.dev:spdxdoc:swpkg"
},
```

A new Annotation is needed for each unique requirement being captured.

In the next release of SPDX (version 3.1), a Requirement element is being proposed to replace the use of the Annotation element for this purpose.

Any applicable license information for the application should also be added at this time.

```
{
  "type":"simplelicensing_LicenseExpression",
  "spdxId":"urn:acme.dev:spdxdoc:lic1",
  "creationInfo":"_:creationinfo-acme",
  "simplelicensing_licenseExpression":"CC0-1.0"
 },{
  "type":"Relationship",
  "spdxId":"urn:acme.dev:spdxdoc:rel2",
  "creationInfo":"_:creationinfo-acme",
  "relationshipType":"hasConcludedLicense",
  "from":"urn:acme.dev:spdxdoc:swpkg",
  "to":["urn:acme.dev:spdxdoc:lic1"]
 },
```

As of this report, there are no tools available to support this phase of SBOM generation. Instead, the product team at ACME manually creates their initial SBOM by copying the template and changing a few items to represent the specific application being developed. Hopefully tools will become available, but for now all that is needed is a text editor to create a new JSON file that follows the template structure.

At the completion of this phase, the software project has a single SBOM file that states the license that guides the use of the software and contains requirements about what the software should do. However, it doesn't yet hold any information about how the software will be implemented to do those things. That will come during the design phase.

ACME stores this SBOM file alongside the Software Requirement Specification document that was produced during this phase as part of ACME's software engineering process.

Design

As summarized by Amazon, "In the design phase, software engineers analyze requirements and identify the best solutions to create the software [6]." An SBOM plays an important role in capturing these solutions and enabling automated verification during the test phase that the implementation followed this design.

The software requirements created in the preceding plan phase are used to create design specifications that specify the technical details of the software. This process starts with a high-level design of the user interface (known as storyboarding) to create a visual representation of the software and make sure it meets the user's needs. These storyboards are then used to create the technical design which will guide the engineers during the implementation phase.

The technical design of the software establishes the various modules and external components (also known as 3rd party libraries) that are necessary to implement the software. Information about these modules and components is then captured in the SBOM.

There is currently no tool available to translate a design to the SBOM snippets that describe it. To support this phase ACME manually creates the snippets and adds them to the SBOM that was generated during the plan phase.

The first snippet presented below describes one of the modules that the design defines.

```
{
 "type":"software_Package",
 "spdxId":"urn:acme.dev:spdxdoc:swmod1",
 "creationInfo":"_:creationinfo-acme",
 "software_primaryPurpose":"module",
 "name":"module-name"
},{
 "type":"Relationship",
 "spdxId":"urn:acme.dev:spdxdoc:rel3",
 "creationInfo":"_:creationinfo-acme",
 "relationshipType":"contains",
 "from":"urn:acme.dev:spdxdoc:swpkg",
 "to":["urn:acme.dev:spdxdoc:swmod1"]
},
```

A module is a self-contained part of the code that handles one or more specific tasks. The design will often break the system into multiple modules that can be worked on independently during implementation, with each potentially having different development organizations and license information. Modules themselves can also contain other modules.

The next snippet describes a 3rd party library/component that a specific module will depend on and is to be included with the software. Information about the version being used, where it was downloaded from, and where the source code can be found is critical in support of future analysis and testing. It is also important to include information about the license associated with the library.

```
{
 "type":"software_Package",
 "spdxId":"urn:acme.dev:spdxdoc:swlib1",
 "creationInfo":"_:creationinfo-acme",
 "name":"library-name",
 "software_primaryPurpose":"library",
 "software_packageVersion":"3.4",
 "software_downloadLocation":"https://www.store.com",
 "software_homePage":"https://www.library-name.org",
 "software_sourceInfo":"https://github.com/lib"
},{
 "type":"Relationship",
 "spdxId":"urn:acme.dev:spdxdoc:rel4",
 "creationInfo":"_:creationinfo-acme",
 "relationshipType":"dependsOn",
 "from":"urn:acme.dev:spdxdoc:swmod1",
 "to":["urn:acme.dev:spdxdoc:swlib1"]
},{
 "type":"Relationship",
 "spdxId":"urn:acme.dev:spdxdoc:rel5",
 "creationInfo":"_:creationinfo-acme",
 "relationshipType":"hasConcludedLicense",
 "from":"urn:acme.dev:spdxdoc:swlib1",
 "to":["urn:acme.dev:spdxdoc:lic1"]
},{
 "type":"simplelicensing_LicenseExpression",
 "spdxId":"urn:acme.dev:spdxdoc:lic1",
 "creationInfo":"_:creationinfo-acme",
 "simplelicensing_licenseExpression":"CC0-1.0"
},
```

Ideally, the 3rd party libraries included in the software design will already have their own SBOMs made available by the vendor with all the information above included. In these situations, it is recommended to reference the library's SBOM file instead of recreating it.

To do this, an ExternalMap is added to an import property that is added to the SpdxDocument element that was already created during the plan phase. The following example shows an import property ACME added to the initial template within the element defined by `"type":"SpdxDocument"`. Multiple ExternalMaps can be added within the import, one for each library being referenced.

```
"import":[
{
 "type":"ExternalMap",
 "externalSpdxId":"urn:xyz.com:spdxdoc:lib_a",
 "locationHint":"http://downloads.xyz.com/lib_a.spdx.json",
 "verifiedUsing":[
 {
  "type":"Hash",
  "algorithm":"sha256",
  "hashValue":"3ba8c25d449c1b7ca1b6ff4eb149"
 }]
}]
```

Once imported, the SBOM can then have a Relationship element to link the module being developed to the 3rd party library. Below is an example of a relationship based on the library imported above. A new relationship is needed for each unique link being established.

```
{
 "type":"Relationship",
 "spdxId":"urn:acme.dev:spdxdoc:rel6",
 "creationInfo":"_:creationinfo-acme",
 "relationshipType":"dependsOn",
 "from":"urn:acme.dev:spdxdoc:swmod1",
 "to":["urn:acme.dev:spdxdoc:lib_a"]
},
```

At this point in the design phase, ACME passes the SBOM to its Software Composition Analysis (SCA) tool to identify any publicly known vulnerabilities in the 3rd party libraries that have been selected. The SCA tool adds information about known vulnerabilities to the SBOM and returns it to the development team. ACME uses this information to make risk-informed decisions regarding whether the design phase should be revisited to change the list of 3rd party libraries, compensating controls should be introduced, or the identified risk should be formally accepted as part of the design.

The test phase section of this report explains the SPDX structure used to record this information about known vulnerabilities in the SBOM, it is not repeated here.

Implement

With the design phase complete, the development team enters the next phase in the lifecycle. The implement phase is where the source code is written, and where the required external libraries are pulled in during the build process to support the code.

Both the “source” and “build” SBOM types as defined by CISA [4] are relevant to the implement phase. The SBOM is used to capture information about the development environment leveraged, the source code written, and the external components that are pulled into the project, including any licensing requirements that must be followed. Additional information related to the build process is also captured - including the parameters used to guide the compilation, a map of environment variables and values that are set during a build session, the time at which a build is triggered and finished, and the tooling used (e.g., GitHub action workflow, invocation of a compiler).

The first information to add to the SBOM being created by ACME represents the source code being written. An initial software_Package was already part of the SBOM created during the plan phase. ACME manually adds a new sourceInfo property to that existing package. In this example, the sourceInfo property holds the location of the Git repository being used to maintain the source code. ACME also updates the version of the software to reflect what is being worked on.

```
{
 "type":"software_Package",
 "spdxId":"urn:acme.dev:spdxdoc:swpkg",
 "creationInfo":"_:creationinfo-acme",
 "name":"acme-sw-name"
 "software_packageVersion":"1.0",
 "software_primaryPurpose":"application",
 "software_sourceInfo":"https://github.com/acme/repo.git"
}
```

The SBOM being created by ACME is then added to the top level of this GIT repository and will be maintained from there throughout the implement phase.

Once coding work has completed, or at least progressed enough, the next step is to capture information related to the build of the software package. Only the build aligned with the software version being described by the SBOM is necessary. Prior build information can be overwritten during each

development sprint. Below is an example of such information that the hypothetical organization ACME captured.

```
{
 "type":"build_Build",
 "spdxId":"urn:acme.dev:spdxdoc:b1",
 "creationInfo":"_:creationinfo-acme",
 "build_buildType":"https://acme.dev/build_workflow/v1",
 "build_buildStartTime":"2025-08-19T01:00:00Z",
 "build_buildEndTime":"2025-08-19T02:00:00Z",
 "build_environment":"urn:acme.dev:spdxdoc:de1",
 "build_parameter":"urn:acme.dev:spdxdoc:de2"
},{
 "type":"DictionaryEntry",
 "spdxId":"urn:acme.dev:spdxdoc:de1",
 "creationInfo":"_:creationinfo-acme",
 "key":"NODE_ENV",
 "value":"production"
},{
 "type":"DictionaryEntry",
 "spdxId":"urn:acme.dev:spdxdoc:de2",
 "creationInfo":"_:creationinfo-acme",
 "key":"branch.default",
 "value":"refs/head/main"
},
```

To start, a Build element is needed. This will be the root that additional information is related to. The above example includes an environment variable used by the build and a parameter that was passed to the build. Ideally all the environment variables and parameters used to guide the build would be listed.

Next, three relationships specific to the build are required. One is about the tool used to compile the executable.

```
{
 "type":"LifecycleScopedRelationship",
 "spdxId":"urn:acme.dev:spdxdoc:lsr1",
 "creationInfo":"_:creationinfo-acme",
 "scope":"build",
 "relationshipType":"usesTool",
 "from":"urn:acme.dev:spdxdoc:b1",
 "to":["urn:acme.dev:spdxdoc:tool1"]
},{
 "type":"Tool",
 "spdxId":"urn:acme.dev:spdxdoc:tool1",
 "creationInfo":"_:creationinfo-acme",
 "name":"compiler_tool_name"
},
```

The second required relationship is used to record the person, organization, or software agent that invoked the build. ACME leverages a GitHub Action to automate their build. A relationship is needed to reflect the specific software agent that invoked the build.

```
{
 "type":"LifecycleScopedRelationship",
 "spdxId":"urn:acme.dev:spdxdoc:lsr3",
 "creationInfo":"_:creationinfo-acme",
 "scope":"build",
 "relationshipType":"invokedBy",
 "from":"urn:acme.dev:spdxdoc:b1",
 "to":["urn:acme.dev:spdxdoc:sa1"]
},{
 "type":"SoftwareAgent",
 "spdxId":"urn:acme.dev:spdxdoc:sa1",
 "creationInfo":"_:creationinfo-acme",
 "name":"acme_build_agent",
 "comment":"agent triggered by GitHub Action"
},
```

Finally, a relationship is required to specify the file that was produced by the build.

```
{
 "type":"LifecycleScopedRelationship",
 "spdxId":"urn:acme.dev:spdxdoc:lsr2",
 "creationInfo":"_:creationinfo-acme",
 "scope":"build",
 "relationshipType":"hasOutput",
 "from":"urn:acme.dev:spdxdoc:b1",
 "to":["urn:acme.dev:spdxdoc:swf1"]
},{
 "type":"software_File",
 "spdxId":"urn:acme.dev:spdxdoc:swf1",
 "creationInfo":"_:creationinfo-acme",
 "createdUsing":"urn:acme.dev:spdxdoc:tool1",
 "name":"acme-sw-name.exe",
 "verifiedUsing":[
 {
  "type":"Hash",
  "algorithm":"sha256",
  "hashValue":"ee4f966d7d149d69ec33a90e78f45"
 }],
 "builtTime":"2025-08-19T02:00:00Z",
 "originatedBy":["urn:acme.dev:spdxdoc:org"],
 "software_primaryPurpose":"executable",
 "software_additionalPurpose":["application"],
 "software_copyrightText":"Copyright 2025, ACME"
},
```

By including verification information such as a file hash enables downstream users of the ACME software to confirm that what they received is exactly what ACME produced and has not been altered during the supply chain. Additional information about when the file was built, what its intended purpose is, and any associated copyright

further supports the confidence that one can establish in the file before it is leveraged by a downstream user.

Now that the executable file has been defined, a new relationship should also be present that links this file to the software package being built.

```
{
 "type":"Relationship",
 "spdxId":"urn:acme.dev:spdxdoc:rel7",
 "creationInfo":"_:creationinfo-acme",
 "relationshipType":"contains",
 "from":"urn:acme.dev:spdxdoc:swpkg",
 "to":["urn:acme.dev:spdxdoc:swfl"]
},
```

The final step within the implement phase is to revisit the 3rd party components added during the design phase and update the information in the SBOM as necessary. For example, ACME compares the version included by the build and makes sure that the packageVersion property is accurate, that the downloadLocation is correct, and that license information is accurate.

Ideally this information is added to the SBOM by the build tool being used. The build tool should grab the latest version of the SBOM from the GIT repository and add information as necessary before checking the updated copy back into the repository. Current build tools don't support this so ACME adds this information to the SBOM manually.

If entering this information like ACME does through manual means is not possible, then analysis tools exist that can produce some of the desired information. Using these tools sacrifices accuracy for time, but this trade-off may be appropriate in some situations. These analysis tools will scan the project, source, and build-related files (e.g., a .csproj file) and programmatically identify external components that have been included. Some tools may even visit component source repositories to gather more detailed information, or to determine if a vendor supplied SBOM is available to be referenced.

Test

The test phase is used to analyze the software that was produced during the implement phase and identify any bugs or vulnerabilities that may be present. Ideally all identified issues would then be remediated; however, operational constraints may require organizations to evaluate and formally accept residual risk associated with unresolved vulnerabilities. An SBOM is used to relay information about these remaining issues to the downstream user.

ACME uses a Software Composition Analysis (SCA) tool to help with this analysis. The SBOM produced as a product of the implement phase is passed to the SCA tool, which in turn parses the SBOM to recognize all 3rd party libraries that are being leveraged. The SCA then searches databases of known vulnerabilities and relays matches back to the ACME test team.

```
{
 "type":"security_Vulnerability",
 "spdxId":"urn:acme.dev:spdxdoc:vul1",
 "creationInfo":"_:creationinfo-acme",
 "summary":"Known vuln in lib.",
 "description":"lib v2 is vulnerable.",
 "security_publishedTime":"2020-06-10T00:00:00Z",
 "security_modifiedTime":"2020-06-10T00:00:00Z",
 "externalIdentifier":[
 {
  "type":"ExternalIdentifier",
  "externalIdentifierType":"cve",
  "identifier":"CVE-2020-2028",
  "identifierLocator":[
    "https://www.cve.org/CVERecord?id=CVE-2020-2028"
  ],
  "issuingAuthority":"urn:acme.dev:spdxdoc:org2"
 }],
 "externalRef":[
 {
  "type":"ExternalRef",
  "externalRefType":"securityAdvisory",
  "locator":"https://xyz.com/security"
 }]
},
```

The example snippet presented above shows

what is added to the SBOM regarding a vulnerability discovered during the test phase.

Relationships are then created to link the identified vulnerability to the component that it applies to and the organization that is reporting it.

```
{
 "type":"Relationship",
 "spdxId":"urn:acme.dev:spdxdoc:rel8",
 "relationshipType":"hasAssociatedVulnerability",
 "from":"urn:xyz.com:spdxdoc:lib_a",
 "to":["urn:acme.dev:spdxdoc:vul1"]
},{
 "type":"Relationship",
 "spdxId":"urn:acme.dev:spdxdoc:rel9",
 "relationshipType":"publishedBy",
 "from":"urn:acme.dev:spdxdoc:vul1",
 "to":["urn:acme.dev:spdxdoc:org2"]
},{
 "type":"Organization",
 "spdxId":"urn:acme.dev:spdxdoc:org2",
 "creationInfo":"_:creationinfo-acme",
 "name":"XYZ"
},
```

The SBOM is then returned to the development team so it can make the decision to either return the software to the design phase to address the discovered issues or advance the software to the deploy phase with the outstanding issues part of it.

This decision is part of ACME's security control gate. To better understand the potential risk associated with each vulnerability identified by the SCA tool, ACME uses a Common Vulnerability Scoring System (CVSS) calculator [7] to determine the characteristics and severity of the software vulnerability. CVSS consists of four metric groups (Base, Threat, Environmental, and Supplemental) with each group contributing to a final severity score. The results of this calculation are added to the SBOM in order to communicate this score to the risk assessment team.

CVSS scores alone do not fully represent operational risk. ACME also considers exploitability within the operational environment, mission impact, exposure pathways, compensating controls, and available threat intelligence when prioritizing remediation activities and deployment decisions.

```
{
 "type":"security_CvssV4VulnAssessmentRelationship",
 "spdxId":"urn:acme.dev:spdxdoc:cvss1",
 "relationshipType":"hasAssessmentFor",
 "security_severity":"critical",
 "security_score":"10.0",
 "security_vectorString":"CVSS:4.0/AV:N/AC:L/AT:N",
 "from":"urn:acme.dev:spdxdoc:vul1",
 "to":["urn:xyz.com:spdxdoc:lib_a"]
},{
 "type":"Relationship",
 "spdxId":"urn:acme.dev:spdxdoc:rel10",
 "relationshipType":"generates",
 "from":"urn:first.org:cvssv4_calculator",
 "to":["urn:acme.dev:spdxdoc:cvss1"]
},{
 "type":"Tool",
 "spdxId":"urn:acme.dev:spdxdoc:tool1",
 "creationInfo":"_:creationinfo-acme",
 "name":"CVSS v4 Calculator",
 "externalRef":[
 {
  "type":"ExternalRef",
  "externalRefType":"other",
  "locator":"https://www.first.org/cvss/calculator/4-0"
 }]
}
```

The SBOM is also leveraged by the ACME security control gate to determine which external components may be too risky for the operational environment. The lack of a vendor supplied SBOM, multiple known vulnerabilities, or download locations that are untrusted could all be reasons for recommending the selection of a different component.

In many instances, the cycle of design-implement-test is repeated across multiple short sprints that are part of iterative and incremental development methodologies such as "spiral" and "agile."

At the end of the test phase the software now has a complete SBOM that captures the requirements, the location of source code, the external libraries included, any and all

applicable licenses, how the software was built, and any outstanding issues that have not been resolved. This SBOM is a complete picture of the software and is tracked by ACME as a Configuration Item (CI). Organizational CIs are tracked within the Configuration Management (CM) processes enabling the status of each CI to be monitored (i.e., traced, versioned, approved, baselined) throughout the development effort.

Deploy

The focus of the deploy phase is on moving the software that was just implemented and tested into its operational environment. As required by NIST Special Publication 800-53 revision 5 control CM-2(6): Development And Test Environments [8], and mentioned by Amazon, "Having separate build and production environments ensures that customers can continue to use the software even while it is being changed or upgraded [6]." Once implementation and testing is complete, it is time to deploy it to its production environment.

The SBOM plays an important role in this phase by enabling downstream organizations to independently evaluate software supply chain risk, licensing exposure, provenance integrity, and outstanding vulnerabilities before deployment.

Attempting to gather build and component information after the software has passed the test phase using composition analysis tools is challenging. SCA tools attempt to curate this information but suffer from a number of issues, including [9]:

- False Negatives
- "Out of Scope" Dependencies
- Transitive Dependencies
- False Positives

To mitigate these challenges, the deployment team at ACME uses the previously generated SBOM to obtain the necessary assessment information without having to perform expensive and inaccurate composition analysis.

ACME uses an SCA tool during the deploy phase, but its use is limited to the assessment steps. Instead of asking the SCA tool to generate an SBOM based on binaries and limited information sources, ACME passes the SBOM that was generated throughout the SDLC and expects the SCA tool to use that as the basis of its assessment. The reports back from the SCA tool are then used by the ACME team to educate its deployment decision.

Maintain

The last phase of the SDLC is maintain, which can last for an undefined amount of time from the end of the deployment through the decommissioning of the software.

Throughout this phase it is important to continuously monitor the software and its dependencies for changes that may increase operational or cybersecurity risk. Of primary concern is the monitoring of threat information sources for new vulnerability and exploit developments associated with external components included within the software.

Similar to the deploy phase, ACME uses their complete and accurate SBOM to drive the assessment of external components and flag any that have released a newer version, to determine if any new vulnerability has been discovered and made public, and to identify components that are no longer maintained or are abandoned. ACME also feeds the information about external components into its intrusion detection

system to establish real-time monitoring of traffic that may be targeting one of the components.

The results of this ongoing assessment have the potential to kick off a new development cycle to address the issues.

Continuous monitoring activities driven by the information in an SBOM enable ACME to prioritize remediation efforts, accelerate incident response activities, and maintain awareness of evolving software supply chain risk throughout the operational life of the software.

When the time comes to decommission the software, ACME uses the information in the SBOM to update the intrusion detection and other security management systems to reflect what sub-components of the decommissioned software no longer need to be monitored for attacks against them or new vulnerabilities within them.

EXPANDING TO THE LARGE ENTERPRISE

As SBOM practices mature, their significance and complexity increase dramatically when adopted across large enterprises. Unlike smaller organizations or single-product teams, large enterprises face unique challenges; they manage diverse portfolios, coordinate distributed development teams, operate across multiple technology stacks, and navigate intricate supply chains. In fact, large enterprises can often rely on hundreds of unique software programs, each requiring its own up-to-date SBOM. The ability to maintain and leverage these SBOMs is foundational to automation goals, especially for continuous monitoring and risk management practices. However, managing this vast collection of SBOMs is far from trivial.

Within a large enterprise, SBOMs evolve from being a project-level artifact to a foundational enterprise capability supporting software governance, cyber risk management, operational resilience, and compliance. For organizations like ACME, this means establishing centralized policies that require SBOM creation and maintenance at every stage of the SDLC, spanning all business units and product lines. Standardization becomes essential; selecting a primary SBOM format—such as SPDX or CycloneDX—and enforcing its use across the organization ensures interoperability between teams and external partners. This consistency simplifies downstream analysis and reporting, making it easier to respond to regulatory requirements and security incidents.

Centralized Repository

Centralized discovery of software and its corresponding SBOMs is a critical enterprise capability. It is important to maintain awareness of, and access to, all SBOMs relevant to an enterprise. Using a central repository that is accessible throughout the enterprise is one of the potential ways to accomplish this discovery task [10]. The enterprise's central repository is used to look up which software programs are present and where each SBOM can be found.

This repository is not expected to hold the physical SBOM files as those should be held in the program's source code repository—or in a program's information directory that hosts the design and tooling leveraged at the start of a project—where the SBOM can be maintained by the software development team. Rather, this central repository is expected to have a link to each official SBOM. This can include links to external

sources for SBOMs pertaining to 3rd party software in use withing the enterprise.

ACME uses an enterprise tracking application to record all the software programs relied on throughout the enterprise. The information gathered for each program includes:

- Software Name
- Purpose
- SBOM Location
- Owning Organization
- Admin Contact

This tracker is accessed by ACME's continuous monitoring tool to pull in the SBOM locations and to document any issues identified.

Continuous Monitoring

Software and the issues to be aware of is always evolving. New vulnerabilities are made public every day. Malicious actors are constantly being discovered working within the 3rd party dependencies being leveraged. The evaluation performed during software testing must be repeated on an ongoing basis throughout the life of the software. This is where continuous monitoring comes into play.

ACME's continuous monitoring tool queries the central repository daily, importing SBOMs from their recorded locations and parsing them to extract information about software packages and dependencies. This information is then used to perform lookups against external threat and vulnerability data sources, as recommended by the NSA [11]. The tool also checks package sites to determine if newer versions of components are available, ensuring the enterprise can proactively address updates. When vulnerabilities or updates are identified, the monitoring tool issues tickets in the central repository, assigning them to the appropriate admin contact for resolution.

In parallel to the above monitoring, ACME leverages a second tool to pull the SBOM information and perform continuous license checks to make sure the enterprise is abiding to legal use of the software.

Additional activities that the SBOM enables are incident response, threat management, and end of life alerting [12].

Organizational Alignment

Organizational alignment is another key factor in successful enterprise-wide SBOM adoption. Executive leadership, security, risk management, and engineering teams should work together to develop clear policies governing SBOM generation, validation, storage, and usage. These policies should be embedded in software engineering standards, procurement requirements, and vendor management processes. Assigning responsibility for SBOM management to specific roles—such as product owners, release managers, or dedicated SBOM coordinators—ensures accountability and consistency.

ACME has established enterprise-level security and compliance teams and tasked them to analyze SBOMs and act on findings. At the same time, ACME has required developers and DevOps engineers to enroll in ongoing training on SBOM standards, tooling, and best practices. These responsibilities within ACME have fostered a culture of transparency and accountability in software supply chain management which is essential for long-term success.

SBOM STATUS

In its beginning, an SBOM was used to collect information about the components

and files present within a system or a piece of software. This has since grown to include information collected across every aspect of the software development lifecycle.

SBOM standards are evolving rapidly to meet the growing demand for transparency in the software supply chain [14], with System Package Data Exchange™ (SPDX®) and CycloneDX emerging as the leading frameworks.

SPDX, formally recognized as ISO/IEC 5962:2021, has become the primary open standard for SBOMs. The recent release, SPDX 3.0.1 from December 2024 and in process as ISO/IEC 5962:2026, was jointly published by the Linux Foundation and the Object Management Group (OMG) and introduces highly detailed metadata capabilities. This includes comprehensive security information such as present vulnerabilities represented by Common Vulnerabilities and Exposures (CVE™) identifiers, vulnerability status assertions via the Vulnerability Exploitability eXchange (VEX), and vulnerability severity scores. SPDX 3.0.1 also includes support for build data, AI models, datasets, and licensing details, with supply chain and hardware being supported shortly.

CycloneDX, developed by OWASP and standardized by ECMA International as ECMA-424, focuses on lightweight automation and security, offering native support for bill of material details, VEX assertions, pedigree, and attestation features.

Together, these standards are laying the groundwork for comprehensive and interoperable SBOM practices across industries.

This report used the SPDX 3.0.1 standard for its examples, but the concepts are equally applicable to other SBOM formats including CycloneDX.

Momentum around the benefits of SBOMs has been building throughout the software community [1] [2]. On February 3, 2022 the National Institute of Standards and Technology (NIST) issued guidance as part of the Secure Software Development Framework (SSDF) encouraging organizations to protect their software by collecting, maintaining, and sharing SBOMs, and analyzing provenance data for all software components identified within the SBOM [15]. Meanwhile, CISA is advancing the SBOM adoption and practices by facilitating community-led work, with a focus on scaling and operationalization, as well as tools, new technologies, and new use cases [2] [16].

On August 25, 2025, CISA released a draft of the update to the minimal suggested elements contained with an SBOM [17]. This new document proposes minor refactoring of the original while adding additional elements for licensing, hashes, lifecycle phase, describing relationships between components, and the tooling used to generate the SBOM. Throughout this paper we show the use of many elements of an SBOM that go beyond depiction of this minimal element set and hopefully show the utility and value these can bring to an enterprise.

The adoption of SBOMs is accelerating worldwide, propelled by a wave of emerging industry efforts and best practices for SBOMs. In the United States, government and industry regulations are leading the way. The U.S. Food and Drug Administration requires (as part of Section 524B(b)(3) of the Federal Food, Drug, and Cosmetic Act as amended by Section 3305 of the Food and Drug Omnibus Reform Act of 2022) manufacturers of medical devices to provide SBOMs that list included commercial, open-source, and off-the-shelf software

components [18]. The U.S. Department of the Army issued a memorandum on August 16, 2024 mandating the use of SBOMs to enhance software supply chain risk management practices and effectively mitigate software supply chain risks [19]. The Payment Card Industry (PCI) Data Security Standard (DSS) v4.0.1 requires organizations to maintain living, searchable inventory of software and their third-party components [20]. An SBOM is directly aligned with this requirement.

Globally, the European Union (EU) Cyber Resilience Act [21] and Germany's BSI TR 03183 [22] are setting SBOM requirements for digitally connected products, and countries such as Japan, the United Kingdom, and Australia are also advancing SBOM initiatives or offering strong guidance for adoption. These regulatory and industry forces are making SBOMs a foundational element of software supply chain security and compliance worldwide.

In response to these trends, the SBOM tooling ecosystem has matured rapidly, offering a wide range of open-source and commercial solutions for generating, managing, and analyzing SBOMs. Tools now support the use of SBOM standards to record information throughout the full software lifecycle, from static code analysis and build-time integration to runtime orchestration and vulnerability management. Importantly, SBOMs are increasingly recognized as living documents that evolve throughout the software lifecycle.

SOFTWARE SYSTEMS USING AI

The example presented in this paper does not have artificial intelligence (AI) elements embedded in the software system. However, as more systems include AI capabilities, organizations should look at the "Software Bill of Materials (SBOM) for AI – Minimum Elements" [13]**.** Developed by CISA together with its international co-creators in the Group of Seven (G7) Cybersecurity Working Group, it identifies a minimum baseline of transparency, provenance, and accountability information necessary to describe AI systems and their supply chains. SPDX 3.1 RC1 and future SPDX releases provide a strong technical foundation for implementing many of the document's recommended minimum elements. SPDX is working on adding representations of AI models, datasets, training artifacts, build environments, inference services, and lifecycle relationships as interconnected graph objects.

In the context of the G7 document, SPDX can serve as the interoperable machine-readable layer that links the minimum elements together across organizational and technical boundaries. The document's emphasis on traceability for datasets, fine-tuning activities, software dependencies, model derivation, and operational provenance aligns closely with the expanded capabilities introduced in SPDX 3.x profiles, particularly the AI, Dataset, Build, and Security profiles. In this way, SPDX provides the structured semantic representation needed to exchange and automate the AI transparency objectives described in the BSI guidance.

At the same time, the "SBOM for AI – Minimum Elements" document also recognizes that AI transparency and trustworthiness require more than a component inventory alone, which is where SPDX must operate in conjunction with complementary governance and assurance capabilities. SPDX can encode references and relationships to Model Cards, Datasheets for Datasets, VEX documents,

Supply-chain Levels for Software Artifacts (SLSA) attestations, safety evaluations, compliance evidence, and organizational governance artifacts required by frameworks such as the NIST AI Risk Management Framework (RMF), ISO/IEC 42001, and emerging EU AI Act obligations. This combined ecosystem approach directly reflects the intent of the CISA and co-creator organizations, which frame AI SBOMs not as isolated documents but as part of a broader transparency and assurance architecture for AI systems. Under this model, SPDX 3.1 RC1 becomes the interoperable evidence graph that connects software components, AI artifacts, operational processes, security attestations, and governance documentation into a unified supply chain representation capable of supporting both technical automation and regulatory accountability.

CONCLUSION

SBOMs are becoming essential capabilities for enterprises seeking continuous visibility into software composition, operational exposure, and software supply chain risk. The more information presented in the SBOM, the more detailed and accurate the reasoning about software can be.

When SBOMs are leveraged across the enterprise, organizations unlock powerful capabilities. Security teams gain the ability to rapidly identify which products are affected by newly disclosed vulnerabilities, enabling targeted patching and risk mitigation. Automated SBOM analysis ensures that all deployed software complies with licensing obligations, reducing legal risk and simplifying audits. Enterprises can assess the provenance and security posture of all third-party components used across their software portfolio, informing procurement and vendor management decisions. SBOMs also support compliance with global regulations, such as PCI DSS, FDA requirements, and the EU Cyber Resilience Act, by providing auditable records of software composition and lifecycle events.

Despite these benefits, expanding SBOM practices to the enterprise level presents challenges. Legacy systems and shadow IT may lack SBOMs or use outdated formats, requiring strategies for retroactive SBOM generation and ongoing coverage. SBOMs may contain sensitive information about proprietary code or third-party relationships, so enterprises must balance transparency with confidentiality through access controls and data protection measures. Integrating SBOM tooling with existing enterprise platforms—such as SIEM, CMDB, and vulnerability management systems—demands careful planning and technical expertise.

This report looked at SBOMs in the context of SPDX 3.0.1 and showed how the hypothetical organization ACME uses the SBOM to assess and monitor its software. Knowing how to make the most of the information contained within an SBOM, and how to add relevant information throughout the in-house development of software enables enterprises like ACME to achieve a higher level of software assurance, operational resilience, and risk-informed decision-making throughout the software lifecycle.

REFERENCES

[1] D. Buttner and R. Martin, "The Cybersecurity Benefits of Leveraging a Software Bill of Materials," September 2022. [Online]. Available: https://www.mitre.org/sites/default/files/2022-11/PR-22-01488-20-cybersecurity-benefits-of-sbom-september-2022.pdf.

[2] U.S. Cybersecurity and Infrastructure Security Agency, "A Shared Vision of Software Bill of Materials (SBOM) for Cybersecurity," 3 September 2025. [Online]. Available: https://media.defense.gov/2025/Sep/03/2003791481/-1/-1/0/JOINT-GUIDANCE-A-SHARED-VISION-OF-SOFTWARE-BILL-OF-MATERIALS-FOR-CYBERSECURITY.PDF.

[3] Committee on National Security Systems, "National Information Assurance Glossary - CNSS Instruction No. 4009," 26 April 2010. [Online]. Available: https://www.dni.gov/files/NCSC/documents/nittf/CNSSI-4009_National_Information_Assurance.pdf.

[4] CISA, "Types of Software Bill of Material Documents," April 2023. [Online]. Available: https://www.cisa.gov/sites/default/files/2023-04/sbom-types-document-508c.pdf.

[5] SPDX, "Getting started writing SPDX 3," 25 April 2025. [Online]. Available: https://github.com/spdx/using/blob/main/docs/getting-started.md.

[6] Amazon Web Services, "What is SDLC (Software Development Lifecycle)?," [Online]. Available: https://aws.amazon.com/what-is/sdlc/. [Accessed 13 March 2026].

[7] Forum of Incident Response and Security Teams, "Common Vulnerability Scoring System Version 4.0 Calculator," [Online]. Available: https://www.first.org/cvss/calculator/4-0. [Accessed 13 March 2026].

[8] NIST, "Security and Privacy Controls for Information Systems and Organizations - NIST SP 800-53 Rev 5," 10 December 2020. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final.

[9] D. Crane, "Why Software Composition Analysis Tools Fail," 10 November 2023. [Online]. Available: https://danacrane.medium.com/why-software-composition-analysis-sca-tools-fail-320c1deee20e.

[10] CISA, "Software Bill of Materials Sharing Lifecycle Report," April 2023. [Online]. Available: https://www.cisa.gov/sites/default/files/2023-04/sbom-sharing-lifecycle-report_508.pdf.

[11] NSA, "Recommendations for Software Bill of Materials Management," January 2024. [Online]. Available: https://media.defense.gov/2023/Dec/14/2003359097/-1/-1/0/CSI-SCRM-SBOM-MANAGEMENT.PDF.

[12] OpenSSF, "Improving Risk Management Decisions with SBOM Data," 18 September 2025. [Online]. Available: https://openssf.org/wp-content/uploads/2025/09/Improving_Risk_Management_Decisions_with_SBOM_Data.pdf.

[13] G7 Cybersecurity Working Group, "Software Bill of Materials for AI," 12 May 2026. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/SBOM-for-AI_minimum-elements.pdf?__blob=publicationFile&v=4.

[14] G. Sullivan, "Why every CISO should demand a comprehensive Software Bill of Materials (SBOM)," 12 November 2025. [Online]. Available: https://www.techradar.com/pro/why-every-ciso-should-demand-a-comprehensive-software-bill-of-materials-sbom.

[15] NIST, "Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities - NIST SP 800-218," 3 February 2022. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf.

[16] U.S. Cybersecurity and Infrastructure Security Agency, "CISA SBOM," [Online]. Available: https://www.cisa.gov/sbom. [Accessed 13 March 2026].

[17] CISA, "2025 Minimum Elements for a Software Bill of Materials (SBOM)," August 2025. [Online]. Available: https://www.cisa.gov/sites/default/files/2025-08/2025_CISA_SBOM_Minimum_Elements.pdf.

[18] 117th Congress, "Consolidated Appropriations Act, 2023," 29 December 2022. [Online]. Available: https://www.congress.gov/117/plaws/publ328/PLAW-117publ328.pdf.

[19] Department of the Army, "Software Bill of Materials Policy Memorandum," 16 August 2024. [Online]. Available: https://api.army.mil/e2/c/downloads/2024/10/17/4072ab1e/asaalt-software-bill-of-materials-policy-signed.pdf.

[20] PCI Security Standards Council, "PCI DSS: v4.0.1," June 2024. [Online]. Available: https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0_1.pdf.

[21] The European Parliament and the Council of the European Union, "Cyber Resilience Act," 23 October 2024. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32024R2847.

[22] Federal Office for Information Security, "Technical Guideline TR-03183: Cyber Resilience Requirements for Manufacturers and Products," 20 September 2024. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03183/BSI-TR-03183-2-2_0_0.pdf.

ACKNOWLEDGEMENTS

The authors extend thanks to their MITRE, government, and industry colleagues for their advice, review, and expertise.

About the Authors

Drew Buttner *leads the software assurance technical capability area within MITRE. Mr. Buttner created the internal secure code review program at MITRE and draws on over 20 years of experience across static code analysis, software security weaknesses, and secure coding practices.*

Robert Martin *leads software supply chain security efforts within MITRE and with industry and is a member of the OMG Steering Committee. Mr. Martin created the community standard for software security weaknesses (CWE™) and MITRE's System of Trust™ catalog of supply chain risks, as well as over 60 global standards addressing the interplay of enterprise risk management, cybersecurity, critical infrastructure protection, and supply chain integrity.*

About MITRE

MITRE's mission-driven teams are dedicated to driving solutions to our nation's most pressing challenges. As a not-for-profit research and development organization, MITRE's staff leverage our unique multi-sponsor vantage point, systems expertise, and innovative solutions to ensure the health, prosperity, and security of our nation.

The views, opinions, and/or findings contained herein are those of the author(s) and should not be construed as an official government position, policy, or decision unless designated by other documentation.