Realities of SBOM:
What is Under the Hood of SBOM

N O V 0 2, 2 0 2 3

Hasan Yasar
Technical Director and Faculty Member
Why does it matter?

97% of commercial code contains at least some open source codes\(^1\)

81% of codebases contain an outdated version of open source\(^2\)

62% of breaches originated from a compromised software component\(^3\)

Software Supply Chain Attacks\(^4\) (2015-2020)

Open Source Vulns per Year\(^5\) (2009-2020)

\(^1\) Sonatype Software Supply Chain Attack report 2020
\(^2\) Mend Annual Report, Open Source Vulnerabilities 2021
\(^3\) Synopsys OSSRA report 2022
\(^4\) Verizon Data Breach Investigations Report 2022
Why does it matter? (continued $$\text{\$}$")

When breached, a single exploitable software component can compromise countless services.

**Use Case:** Log4j – a software component embedded in Java-based products and web services.

**Exemplary ROI**
- ≈10,000 person-hours with an estimated ad-hoc response cost equal to $400K – $900K
- A breach incident could have resulted in an average financial risk range of $141K – $5.7M
Targeting Software Development

Adversaries target software development (source, build, and deployment systems): SolarWinds, ASUS Live/Shadowhammer, MEDoc/Not Petya, NPM packages, docker hub and others.

*Reflections on Trusting Trust*, Ken Thompson, 1984:

“No amount of source-level verification or scrutiny will protect you from using untrusted code. In demonstrating the possibility of this kind of attack, I picked on the C compiler. I could have picked on any program-handling program such as an assembler, a loader, or even hardware microcode. As the level of program gets lower, these bugs will be harder and harder to detect. A well-installed microcode bug will be almost impossible to detect.”

Software supply chain for assembled software

- Expanding the scope and complexity of acquisition and deployment
- Visibility and direct controls are limited (only in shaded area)

But SW Development became like a College Party

Code we wrote

Code someone else wrote

Do you know what's in your app?
Is SBOM all about creating the FILE! (with tools)
But How to Start?
Common Format for SBOM

**SPDX** is an open standard for communicating software bill of material information (including components, licenses, copyrights, and security references). The SPDX specification is developed by the SPDX workgroup, which is hosted by the Linux Foundation. The grass-roots effort includes representatives from more than 20 organizations—software, systems and tool vendors, foundations and systems integrators.

**CycloneDX** is a software bill of materials (SBOM) standard, purpose-built for software security contexts and supply chain component analysis. The specification is maintained by the CycloneDX Core working group, with origins in the OWASP community.

**SWID** tags record unique information about an installed software application, including its name, edition, version, whether it is part of a bundle and more. SWID tags support software inventory and asset management initiatives. The structure of SWID tags is specified in international standard ISO/IEC 19770-2:2015.

- All has common SB elements.
- A 'multilingual' ecosystem -machine readable format-
- Rather than pick a winner, develop guidance to support all formats with effective interoperability.
DevSecOps

- **“DevOps is a set of principles and practices which enable better communication and collaboration between relevant stakeholders for the purpose of specifying, developing, continuously improving, and operating software and systems products and services.”** [1]

- **“DevSecOps is a cultural and engineering practice that breaks down barriers and opens collaboration between development, security, and operations organizations using automation to focus on rapid, frequent delivery of secure infrastructure and software to production. It encompasses intake to release of software and manages those flows predictably, transparently, and with minimal human intervention/effort.”** [2]

---


## Minimum elements for SBOM

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Description</th>
<th>How/Where with DevSecOps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier Name</td>
<td>The name of an entity that creates, defines, and identifies components</td>
<td>Issue Tracking/VCS/IaC</td>
</tr>
<tr>
<td>Component Name</td>
<td>Designation assigned to a unit of software defined by the original supplier</td>
<td>Issue Tracking/VCS/IaC</td>
</tr>
<tr>
<td>Version of the Component</td>
<td>Identifier used by the supplier to specify a change in software from a previously identified version.</td>
<td>IaC/Build/Packaging</td>
</tr>
<tr>
<td>Other Unique Identifiers</td>
<td>Other identifiers that are used to identify a component, or serve as a look-up key for relevant databases.</td>
<td>VCS</td>
</tr>
<tr>
<td>Dependency Relationship</td>
<td>Characterizing the relationship that an upstream component X is included in software</td>
<td>Project/VCS/IaC</td>
</tr>
<tr>
<td>Author of SBOM Data</td>
<td>The name of the entity that creates the SBOM data for this component</td>
<td>Issue Tracking/VCS</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Record of the date and time of the SBOM data assembly</td>
<td>VCS</td>
</tr>
</tbody>
</table>

*VCS: Version Control System, IaC: Infrastructure as Code*

*https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf*
Minimum practices for SBOM

<table>
<thead>
<tr>
<th>Practice/Use Cases</th>
<th>Description</th>
<th>How/Where with DevSecOps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Fields</td>
<td>Document baseline information about each component that should be tracked: Supplier, Component Name, Version of the Component, Other Unique Identifiers, Dependency Relationship, Author of SBOM Data, and Timestamp</td>
<td>SBOM files (Design, Source, Build, Analyzed, Deployed, Runtime)</td>
</tr>
<tr>
<td>Automation Support</td>
<td>Support automation, including via automatic generation and machine-readability to allow for scaling across the software ecosystem. Data formats used to generate and consume SBOMs include SPDX, CycloneDX, and SWID tags.</td>
<td>Overall DevSecOps platform – VCS, IaC, CI, CD, CM</td>
</tr>
<tr>
<td>Practices and Processes</td>
<td>Define the operations of SBOM requests, generation and use including: Frequency, Depth, Known Unknowns, Distribution and Delivery, Access Control, and Accommodation of Mistakes</td>
<td>Policy, Governance, Education, Organization wide Common Artifactory,</td>
</tr>
</tbody>
</table>

**VCS:** Version Control System, **IaC:** Infrastructure as Code, **CI:** Continuous Integration, **CD:** Continuous Delivery/Deployment, **CM:** Continuous Monitoring

*https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf*
# Implementing core SBOM fields

<table>
<thead>
<tr>
<th>Elements</th>
<th>SPDX</th>
<th>CycloneDX</th>
<th>SWID</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Supplier</strong></td>
<td>(3.5) PackageSupplier:</td>
<td>publisher</td>
<td>&lt;Entity&gt; @role</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(softwareCreator/publisher), @name</td>
</tr>
<tr>
<td><strong>Component</strong></td>
<td>(3.1) PackageName:</td>
<td>name</td>
<td>&lt;softwareIdentity&gt; @name</td>
</tr>
<tr>
<td><strong>Unique Identifier</strong></td>
<td>(3.2) SPDXID:</td>
<td>bom/serialNumber and component/bom-ref</td>
<td>&lt;softwareIdentity&gt; @tagID</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td>(3.3) PackageVersion:</td>
<td>version</td>
<td>&lt;softwareIdentity&gt; @version</td>
</tr>
<tr>
<td><strong>Component Hash</strong></td>
<td>(3.10) PackageChecksum:</td>
<td>hash</td>
<td>&lt;Payload&gt;/../&lt;File&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[@[hash-algorithm]:hash]</td>
</tr>
<tr>
<td><strong>Relationship</strong></td>
<td>(7.1) Relationship: CONTAINS</td>
<td>(Nested assembly/subassembly and/or dependency graphs)</td>
<td>&lt;Link&gt;@rel, @href</td>
</tr>
<tr>
<td><strong>SBOM Author</strong></td>
<td>(2.8) Creator:</td>
<td>bom-descriptor: metadata/manufacturer/contact</td>
<td>&lt;Entity&gt; @role</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(tagCreator), @name</td>
</tr>
</tbody>
</table>
Start to Build a Secure pipeline for your Development and Deployment
Then use secure pipeline for your app!

• Create and maintain software dependency for each build

• Track ALL 3rd party, including open-source libraries, used in
  1. code development,
  2. build, and
  3. runtime
Operations – How Does It All Fit

SBOM Policy (Design/Source)

1. Establish Policy to mandate, software delivered to the enterprise from a supplier to include a contractual software bill of materials in common format.
2. Ask requestor to develop first set of SBOM files.

Policy to acquire/enforce (Vendor SBOM)

• Perform impact analysis of supply chain software libraries.
• Perform impact analysis on vendor SBOM using OWASP dependency track.
• Use analysis output as an additional input to overall risk assessment quantification.

SRA (OWASP Dependency Track)

CI/CD Policy Compliance (Build, Analyze, Deploy)

1. Continuous Integration (GitLab)
   - Build pipelines in GitLab submit finished artifacts to Artifactory for deployment phase activities.
   - Developers will have visibility to component composition and vulnerabilities through GitLab.

2. Policy Compliance (Artifactory)
   - VG to create and maintain policies that identify security violations based on matching conditions of CVE severity. <New>
   - Artifactory scans finished component artifacts and generates SBOM containing CVE details.
   - VG to measure policy violations across the portfolio, and against individual projects and components for SecureUP reporting. <New>
   - Components that pass security and compliance policies check make to deployment. <New>
   - Artifactory enforces security violation policy rules be blocking components getting into continuous deployment stage.

3. Continuous Deployment (Artifactory)
   - Components failed from security policy scans are reported back to GitLab for remediation.
Integrate and automate SBOM

• Develop and implement artifact catalog
• Document automation scripts for dependencies
• Integrate SBOM files /scripts into build files
• Enable automated artifact pull/push one each build
• Develop automated scripts including artifacts and environment configuration
• Develop and update scripts to release any dependencies along with new version of the app
• Enable base containers/internal repository
• Continuously monitor and analyze dependencies
  • Security vetting process for each approved libraries
  • Monitor for each newly CVEs
Understand How to Create, Test, Validate, and Recognize SBOM across DevSecOps
SBOM across DevSecOps

SBOM should be integrated into SDLC across DevSecOps practices and process.
Realities of SBOM

SBOM in action
What is the project

- Rust code base that contains a chain vulnerability
- Determine issues from supply chain dependencies
- Verify the process for building the project
- Execute checks of the code being written
- Execute checks of the external libraries
Pipeline

- Static code analysis
- Build the code
- Package up support library
- Generate an SBOM (Software Bill Of Materials)
- Upload build artifacts
- Scan SBOM
- Verify the build commands
Demo Time

Let us see exemplary project to for SBOM lifecycle
Realities of SBOM

Pipeline Stages
Lint, Build, and Package Stages

- Cargo Clippy
  - Static code analysis
  - Lint checks
- Cargo Build
  - Builds the crate
  - Builds the binary
  - Fails only if code errors
- Cargo Publish
  - Push the crate to Artifactory
SBOM and Verify Stages

- Software Bill Of Materials
  - CycloneDX
  - SPDX
- Artifactory
  - Push binary
  - Add build metadata
- verify
  - In-toto-verify the build steps
- bomber_scan
  - Kungfu-Bomber
    - Vulnerability Scan SBOM
    - Check dependent crates
lint_check

$ in-toto-run -s -n lint_check -k ./secure_files/gitlab -v -- cargo clippy

... Checking supplychain-sandbox-wrap v0.1.9 (/var/lib/runner/builds/sei/supplychain-sandbox/supplychain-sandbox-wrap) warning: variable does not need to be mutable
-- supplychain-sandbox-wrap/src/supplychain_sandbox_wrap.rs:17:9

| 17 | let mut v = vec![1, 2, 3];
| ----^ help: remove this `mut`
| |
| = note: `#[warn(unused_mut)]` on by default
arning: `supplychain-sandbox-wrap` (lib) generated 1 warning
Checking supplychain-sandbox v0.1.1 (/var/lib/runner/builds/sei/supplychain-sandbox) Finished dev [unoptimized + debuginfo] target(s) in 23.17s
<bom ...
...
<component type="library" bom-ref="pkg:cargo/regex@1.5.4">
  <name>regex</name>
  <version>1.5.4</version>
  <description>...</description>
  <scope>required</scope>
...
<purl>pkg:cargo/regex@1.5.4</purl>
<externalReferences>
  ...</externalReferences>
</component>
...
</bom>
Bomber scanning

$ bomber scan bom.xml

DKFM - DevOps Kung Fu Mafia
https://github.com/devops-kung-fu/bomber

Version: 0.4.4

- Ecosystems detected: cargo
- Scanning 5 packages for vulnerabilities...
- Vulnerability Provider: OSV Vulnerability Database (https://osv.dev)
- Files Scanned
bom.xml (sha256:dadcdb030f572f30441136ea1088242a33d7bf213ff74b9265e83c587d05485f)

- Licenses Found: Unlicense OR MIT, MIT OR Apache-2.0

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NAME</th>
<th>VERSION</th>
<th>SEVERITY</th>
<th>VULNERABILITY</th>
<th>EPSS %</th>
</tr>
</thead>
<tbody>
<tr>
<td>cargo</td>
<td>regex</td>
<td>1.5.4</td>
<td>UNSPECIFIED</td>
<td>CVE-2022-24713</td>
<td>52%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HIGH</td>
<td>CVE-2022-24713</td>
<td>52%</td>
</tr>
</tbody>
</table>
What is next?
How do we solve it?

Software Bill of Materials (SBOM) needs to be considered mandatory for any effective DevSecOps or AppSec effort.

Document the “ingredients” within each software package

- Expediates detection of vulnerable components downstream.
- Avoids heavy costs incurred due to security breaches, a bad reputation, and regulatory penalties.

- Map all Apps to codebases
  - Decide: project team be established to review and update data quality management

- Enforce CI/CD policy compliance
  - Decide: by default, formally require teams follow end-to-end for build and deploy using CI/CD pipeline that supports integration of SBOM generation, or they file an exception.
Takeaway!

- Develop and share Open source software usage policy
- Bring the SBOM into development early:
  - As early as in planning stage like part of security stories.
  - Evaluate a product’s threat resistance
  - Create a centralized private repositories of vetted 3rd party components for all developers
- Apply policy and management through DevOps pipeline
- Continuous training and monitoring developers' activities
- Automate and monitor dependencies management
- Track build and deploy dependencies list
- Apply discovered(new) vulnerabilities and deployment process
- Establish good product distribution practices
- Automate your SBOM tools as much as possible.
- Integrate. Integrate. INTEGRATE!
For More Information

DevSecOps: https://www.sei.cmu.edu/go/devops
DevOps Blog: https://insights.sei.cmu.edu/devops
Webinar Series: https://www.sei.cmu.edu/publications/webinars/
Podcast Series: https://www.sei.cmu.edu/publications/podcasts/
Some day we won’t even need coders any more. We’ll be able to just write the specification and the program will write itself.

Oh wow, you’re right! We’ll be able to write a comprehensive and precise spec and bam, we won’t need programmers any more!

Exactly!

And do you know the industry term for a project specification that is comprehensive and precise enough to generate a program?

Code

Uh... no...

It’s called code

CommitStrip.com
Question and answer time

What does this mean to you?

How can we put these ideas into action?
Contact Information

Hasan Yasar
Technical Director, Adjunct Faculty Member
Continuous Deployment of Capability,
Software Engineering Institute | Carnegie Mellon University
hyasar@cmu.edu