SBOM Lifecycle

Adrian Diglio
Principal PM Manager of Secure Software Supply Chain (S3C)

5/31/2023
Agenda

• SBOM Pyramid
• Generating Different Types of SBOMs
• Sharing SBOMs based on software type
• Preparing for SBOM Consumption
• Security practices to help with SBOM hygiene
The SBOM Pyramid

- Generate
- Share
- Consume
<table>
<thead>
<tr>
<th>SBOM Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>SBOM of intended, planned software project or product with included components (some of which may not yet exist) for a new software artifact.</td>
</tr>
<tr>
<td>Source</td>
<td>SBOM created directly from the development environment, source files, and included dependencies used to build a product artifact.</td>
</tr>
<tr>
<td>Build</td>
<td>SBOM generated as part of the process of building the software to create a releasable artifact (e.g., executable or package) from data such as source files, dependencies, built components, build process ephemeral data, and other SBOMs.</td>
</tr>
<tr>
<td>Analyzed</td>
<td>SBOM generated through analysis of artifacts (e.g., executables, packages, containers, and virtual machine images) after its build. Such analysis generally requires a variety of heuristics. In some contexts, this may also be referred to as a “3rd party” SBOM.</td>
</tr>
<tr>
<td>Deployed</td>
<td>SBOM provides an inventory of software that is present on a system. This may be an assembly of other SBOMs that combines analysis of configuration options, and examination of execution behavior in a (potentially simulated) deployment environment.</td>
</tr>
<tr>
<td>Runtime</td>
<td>SBOM generated through instrumenting the system running the software, to capture only components present in the system, as well as external call-outs or dynamically loaded components. In some contexts, this may also be referred to as an “Instrumented” or “Dynamic” SBOM.</td>
</tr>
</tbody>
</table>
Microsoft’s open source SBOM-Tool

- Microsoft open sourced its SBOM generation tool
  [github.com/microsoft/sbom-tool](https://github.com/microsoft/sbom-tool)

**Features:**

- Cross-platform support (Windows, Linux, and Mac)
- General purpose dependency detection (NPM, NuGet, Pypi, CocoaPods, Maven, Golang, Rust Crates, RubyGems, containers and their Linux packages via [Syft](https://github.com/syft), Gradle, Ivy, and vcpkg)
- Able to detect transitive dependencies
- Produces NTIA-compliant SPDX SBOMs
- Ideally used for Build SBOMs, but can produce Source SBOMs as well
- Can detect and reference other SBOMs
GitHub’s Self-Service SBOMs

Introducing self-service SBOMs | The GitHub Blog

- Ability to produce SBOMs On-Demand or programmatically generate them via the SBOM gh CLI extension
- REST API for generating an SBOM is coming soon
- Able to upload SBOM to dependency graph to receive Dependabot alerts
Defender for IoT Firmware Analysis

Automated identification of potential firmware security vulnerabilities

- SBOM
- Known vulnerabilities (CVE)
- Binary hardening
- Crypto Material
- Built-in accounts / weak passwords
- Continuous monitoring of new threats

Owner/operator

- Surfaced in Defender for IoT
- Brownfield device visibility
- Ongoing monitoring
- Enforce security policy before accepting delivery

OEM

- Part of Secure Development Lifecycle (SDL)
- Future GitHub integration
- Enforce security policy before ship
- Supply chain validation
- Regulatory (vuln mgmt / SBOM, etc.)
## Share: SBOMs by Type

<table>
<thead>
<tr>
<th>Software Type</th>
<th>Delivery Method</th>
<th>Example Links</th>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System, Installable Binary</td>
<td>Included in compiled code</td>
<td><a href="https://docs.microsoft.com/en-us/powershell/scripting/security/PowerShell-security-features">PowerShell security features - PowerShell</a></td>
<td>Components worried about reproducibility might keep SBOMs detached (stored online)</td>
</tr>
<tr>
<td>Container image</td>
<td>Detached, stored side-by-side in the registry</td>
<td><a href="https://docs.microsoft.com/en-us/azure/container-registry/">Attach, push, and pull supply chain artifacts - Azure Container Registry</a></td>
<td></td>
</tr>
<tr>
<td>Virtual Hard Drive (VHD)</td>
<td>Detached (stored online)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Embedded, IoT, Firmware</td>
<td>Detached (stored online)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open Source Components</td>
<td>Multiple (depends on ecosystem). Stored in repo, detached, or included in package</td>
<td><a href="https://anaconda.org">SBOMs at Anaconda</a></td>
<td>If included in the compiled code, it couldn’t be extracted</td>
</tr>
<tr>
<td>Cloud Service</td>
<td>Detached (stored online)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microservices</td>
<td>Detached (stored online)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SBOM Sharing

Given that so many software types would favor storing SBOMs detached from the software it represents, **there is a need to strongly link a software artifact or service to its SBOM stored online**

For the U.S. Presidential Executive Order 14028, in accordance with **OMB M-22-18**, Microsoft is first going to provide SBOMs to Federal Agencies when requested

As we mature in our processes, individual product teams may choose to include SBOMs in their shipping bits – just as **PowerShell** did starting with version 7.2

---

**Software Bill of Materials (SBOM)**

Beginning with PowerShell 7.2, all install packages contain a Software Bill of Materials (SBOM). The SBOM is found at `$PSHOME/manifest/spdx-2.2/manifest.spdx.json`. The creation and publishing of the SBOM is the first step to modernize Federal Government cybersecurity and enhance software supply chain security.

The PowerShell team is also producing SBOMs for modules that they own but ship separately from PowerShell. SBOMs will be added in the next release of the module. For modules, the SBOM is installed in the module’s folder under `manifest/spdx-2.2/manifest.spdx.json`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>manifest.spdx</td>
<td>6/24/2022 10:13 PM</td>
<td>JSON Source File</td>
</tr>
<tr>
<td>manifest.spdx.sha256</td>
<td>6/24/2022 10:13 PM</td>
<td>SHA256 File</td>
</tr>
</tbody>
</table>
Supporting the SBOM Everywhere Initiative

OpenSSF Mobilization Plan identified 10 workstreams, including SBOMs Everywhere.

NuGet’s 2023 Plan includes supporting SBOM Generation natively within Clients and SDKs and more.
SBOM Validation at Release

- In a 28-day period, Microsoft generated over 1.59 million SBOMs at build time
  - Not all builds go through release
- Validating SBOMs at release ensures that what we built is what we released (integrity verification)
- Storing and sharing SBOMs from a release is the area of our focus today
SBOM Sharing Lifecycle Phases

As more Governments, contracts, and other enforcement mechanisms start requiring SBOMs, there will be a tipping point where SBOMs will soon become commonplace across the industry.

The demand to discover and access SBOMs will likely increase as organizations, enterprises, and small and medium businesses adopt SBOM Consumption/Management tooling into their Risk Management practices.

Source: CISA SBOM Sharing Lifecycle Report
Organizational Preparation for SBOM Consumption

- Readiness for SBOM Consumption
  - SDKs or Tooling for Partners/Vendors
    - SBOM Consumption Tooling
    - Risk Management Team's acceptance of Increased Scope
    - Contractual requirements updated and shared
    - New Incident Response & other Processes Captured
    - Budget
Example SBOM Consumption Tooling

Software consumers will load SBOMs into SBOM Management tools that will reveal:
  1) OSS licenses
  2) OSS vulnerabilities

An SBOM’s core purpose is for incident response. When the next Log4J happens, our customers can answer the question “Am I affected?” and “What is affected?”

Many Organizations will have SBOMs available before they have VEX statements available

*This is a 3P tool that represents what software consumers will see
Hygiene for Producing Better SBOMs
OpenSSF Secure Supply Chain Consumption Framework (S2C2F)

- The S2C2F is a set of practices and requirements, organized into a maturity model, on how to securely consume OSS into the developer’s DevOps workflow
  - This is referring to NuGet, NPM, Maven, Pypi, Rust Crates, Rubygems, Golang, vcpkg, etc.
- The S2C2F pairs with producer-focused frameworks such as NIST, NSA ESF, SLSA, CIS, and more
# Real-world OSS supply chain threats

<table>
<thead>
<tr>
<th>Threats</th>
<th>Real examples</th>
<th>Mitigation via S2C2F</th>
<th>Framework requirement reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accidental vulnerabilities in OSS code or containers that we inherit</td>
<td>SaltStack</td>
<td>Automated patching, display OSS vulnerabilities as pull requests</td>
<td>UPD-2, UPD-3</td>
</tr>
<tr>
<td>Intentional vulnerabilities/backdoors added to an OSS code base</td>
<td>phpMyAdmin</td>
<td>Perform proactive security review of OSS</td>
<td>SCA-5</td>
</tr>
<tr>
<td>A malicious actor compromises a known good OSS component and adds</td>
<td>ESLint incident</td>
<td>Ability to block ingestion via malware scan, curated feed, all packages are scanned</td>
<td>ING-3, ENF-2, SCA-4</td>
</tr>
<tr>
<td>malicious code into the repo</td>
<td></td>
<td>for malware prior to download</td>
<td></td>
</tr>
<tr>
<td>A malicious actor creates a malicious package that is similar in name</td>
<td>Typosquatting</td>
<td>OSS provenance analysis, curated feed, all packages are scanned for malware prior to</td>
<td>AUD-1, ENF-2, SCA-4</td>
</tr>
<tr>
<td>to a popular OSS component to trick developers into downloading it</td>
<td></td>
<td>download</td>
<td></td>
</tr>
<tr>
<td>A malicious actor compromises the compiler used by the OSS during build</td>
<td>CCleaner</td>
<td>Rebuilding OSS on trusted build infrastructure ensures that packages don’t have</td>
<td>REB-1</td>
</tr>
<tr>
<td>adding backdoors</td>
<td></td>
<td>anything injected at build time</td>
<td></td>
</tr>
<tr>
<td>Dependency confusion, package substitution attacks</td>
<td>Dependency Confusion</td>
<td>Securely configure your package source mapping, curated feed</td>
<td>ENF-1, ENF-2</td>
</tr>
<tr>
<td>An OSS component adds new dependencies that are malicious</td>
<td>Event-Stream incident</td>
<td>All packages are scanned for malware prior to download, curated feed</td>
<td>SCA-4, ENF-2</td>
</tr>
<tr>
<td>The integrity of an OSS package is tampered after build, but before</td>
<td>How to tamper with Electron apps</td>
<td>Digital signature or hash verification, SBOM validation</td>
<td>AUD-3, AUD-4</td>
</tr>
<tr>
<td>consumption</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Upstream source can be removed or taken down which can then break</td>
<td>left-pad</td>
<td>Use package-caching solutions, mirror a copy of OSS source code to an</td>
<td>ING-2, ING-4</td>
</tr>
<tr>
<td>builds that depend on that OSS component or container</td>
<td></td>
<td>internal location for Business Continuity and Disaster Recovery (BCDR) scenarios</td>
<td></td>
</tr>
<tr>
<td>OSS components reach end-of-support/end-of-life and therefore</td>
<td>log4net CVE-2018-1285</td>
<td>Scan OSS to determine if it is at end-of-life</td>
<td>SCA-3</td>
</tr>
<tr>
<td>don’t patch vulnerabilities</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vulnerability not fixed by upstream maintainer in desired timeframe</td>
<td>Prototype Pollution in lodash</td>
<td>Implement a change in the code to address a zero-day vulnerability, rebuild, deploy</td>
<td>FIX-1</td>
</tr>
<tr>
<td>Bad actor compromises a package manager account (e.g. npm), with no</td>
<td>Ua-parser-js</td>
<td>OSS provenance analysis, curated feed, scan OSS for malware</td>
<td>AUD-1, ENF-2, SCA-4</td>
</tr>
<tr>
<td>change to the corresponding open source repo, and uploads a new</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>malicious version of a package</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Secure Supply Chain Consumption Framework Maturity Model

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Minimum OSS Governance Program" /></td>
<td><img src="image" alt="Secure Consumption and Improved MTTR" /></td>
<td><img src="image" alt="Malware Defense and Zero-Day Detection" /></td>
<td><img src="image" alt="Advanced Threat Defense" /></td>
</tr>
</tbody>
</table>

- **Minimum OSS Governance Program**
  - Use package managers
  - Local copy of artifact
  - Scan with known vulns
  - Scan for software licenses
  - Inventory OSS
  - Manual OSS updates

- **Secure Consumption and Improved MTTR**
  - Scan for end life
  - Have an incident response plan
  - Auto OSS updates
  - Alert on vulns at PR time
  - Audit that consumption is through the approved ingestion method
  - Validate integrity of OSS
  - Secure package source file configuration

- **Malware Defense and Zero-Day Detection**
  - Deny list capability
  - Clone OSS source
  - Scan for malware
  - Proactive security reviews
  - Enforce OSS provenance
  - Enforce consumption from curated feed

- **Advanced Threat Defense**
  - Validate the SBOMs of OSS consumed
  - Rebuild OSS on trusted infrastructure
  - Digitally sign rebuilt OSS
  - Generate SBOM for rebuilt OSS
  - Digitally sign protected SBOMs
  - Implement fixes

The framework lists out the requirements and organizes it into a maturity model, where each level has different themes.

S2C2F is referenced in the forthcoming NSA ESF guidebook.

Checkout the S2C2F Guide today: [s2c2f/framework.md at main · ossf/s2c2f · GitHub](https://github.com/ossf/s2c2f/blob/main/s2c2f/framework.md)
Questions?

Adrian Diglio

- LinkedIn: Adrian Diglio | LinkedIn
- Email: Adrian.Diglio@Microsoft.com

Further Reading

- Generating Software Bills of Materials (SBOMs) with SPDX at Microsoft - Engineering@Microsoft
- Microsoft contributes S2C2F to OpenSSF | Microsoft Security Blog
- The Journey to Secure the Software Supply Chain at Microsoft - Engineering@Microsoft