2.0

Healthcare SBOM Proof of Concept

UPDATE 2020-10-22

Topics

Overview / Status	Jim Jacobson
SBOM Generation For Iteration 2 Medical Device Manufacturer (MDM) - Iteration 2 Objectives - Resources	Ed Heierman
SBOM Consumption From Iteration 1 Healthcare Delivery Organization (HDO) - Quick Start Guide - Challenges from Iteration 1	Michael Dittamo
Conclusion – Looking Forward	Jennings Aske

Overview / Status

2.0

Goals

Prove viability of Framing document's definition

Expansion beyond initial PoC
- Expanded use cases
- Expanded participant list: HDOs, MDMs, vendors/suppliers
- Tooling and automation

"How-to" / playbooks for HDOs and MDMs

Approach

Collaborate with other working groups on definition

SBOMs produced for a predefined set of devices

Execute proposed use cases including procurement

Iterate to increasing complexity and speculative topics with published deliverables each iteration

Participants

HDOs
- Cedars-Sinai
- Christiana Care
- Cleveland Clinic
- Mayo Clinic
- New York Presbyterian
- Sutter Health

MDMs
- Abbott
- Medtronic
- Philips
- Siemens Healthineers
- Thermo Fisher Scientific

Vendors
- Medigate
- Censinet
- Nuvolo

SBOM Generation For Iteration 2

ED HEIERMAN, ABBOTT

Iteration 1 Objectives

2.0

Execute Naming-Focused Use Cases

- Use Case 1: A Supplier Creates an SBOM for a Primary Component
- Use Case 2: An SBOM Stakeholder Creates an SBOM

Confirm SPDX format supports content

- One format for this iteration
- Additional formats in next iteration

Confirm Baseline Elements

- Author Name
- Supplier Name
- Component Name
- Version String
- Unique Identifier
- Relationship
- Primary Component
- Included Components

Iteration 2 Objectives – Expand on Iteration 1 Content

2.0

Software Document Version

SBOM Component Completeness

Unique Identifier using **purl**

Software Identity

- List of common components
- Conventions to establish software identity
- Alignment across participant SBOMs for common components

SBOM Content for Included Components

- Include in the SBOM Document
- Reference an External SBOM Document

SBOM for Medical Device System-of-Systems

- Single Endpoint
- Multiple Endpoints

SBOM Registry

- List of POC SBOMs

Resources

2.0

Microsoft Word Document

SBOM Generation How-to Document

- Guidance
- Actions for Iteration 2
- Format Examples

Adobe Acrobat Document

SBOM Medical Device System-of-Systems Examples

Microsoft Excel Worksheet

Component Master List

- SBOM Examples
- SBOM Generation Tools
- SBOM Conformance Tools

Microsoft Excel Worksheet

- SBOM Registry

SBOM Consumption From Iteration 1

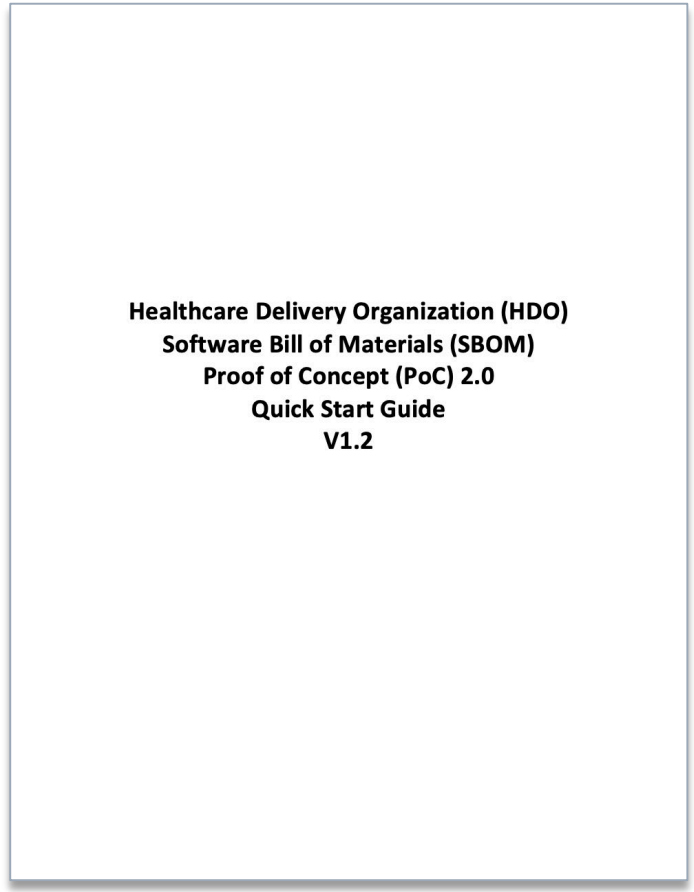
MICHAEL DITTAMO, NEW YORK PRESBYTERIAN

Quick Start Guide

2.0

- Background Information on the PoC 1.0
- Data Sharing / NDA
- How to Get Involved
- Technologies Utilized
- Formats and Standards
- Use Case Exercises
  - Ingestion, Parsing, and Correlation
  - Procurement
  - Asset Management
  - Risk Management
  - Vulnerability Identification and Management

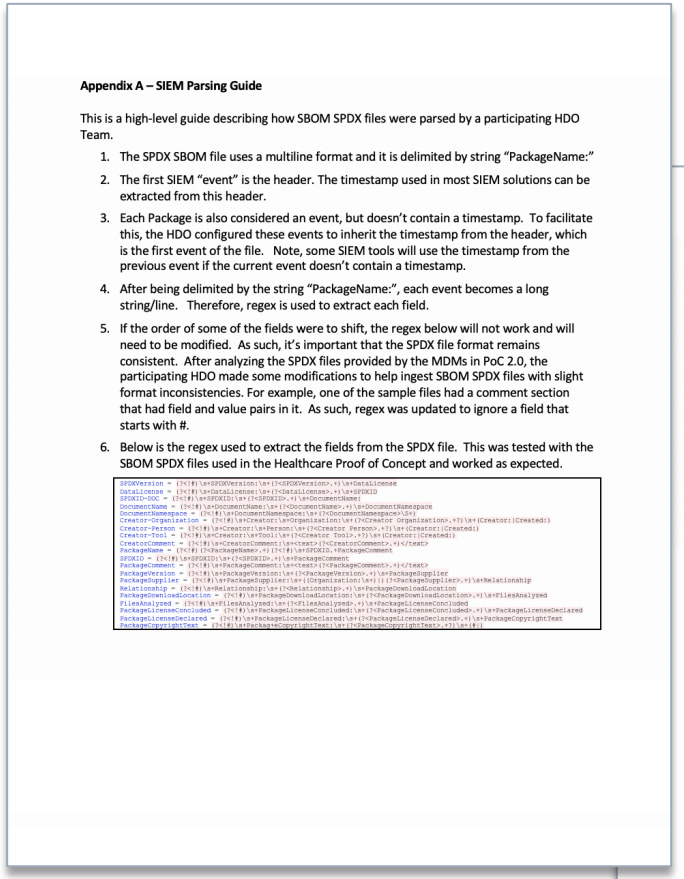
Microsoft Word Document



Quick Start Guide

2.0

- Technical Information
  - SIEM Parsing Guide (Regex)
  - Example SWID SBOM Syntax



Appendix A – SIEM Parsing Guide

This is a high-level guide describing how SBOM SPDX files were parsed by a participating HDO Team.

1. The SPDX SBOM file uses a multiline format and it is delimited by string "PackageName:"
2. The first SIEM "event" is the header. The timestamp used in most SIEM solutions can be extracted from this header.
3. Each Package is also considered an event, but doesn't contain a timestamp. To facilitate this, the HDO configured these events to inherit the timestamp from the header, which is the first event of the file. Note, some SIEM tools will use the timestamp from the previous event if the current event doesn't contain a timestamp.
4. After being delimited by the string "PackageName:", each event becomes a long string/line. Therefore, regex is used to extract each field.
5. If the order of some of the fields were to shift, the regex below will not work and will need to be modified. As such, it's important that the SPDX file format remains consistent. After analyzing the SPDX files provided by the MDMs in PoC 2.0, the participating HDO made some modifications to help ingest SBOM SPDX files with slight format inconsistencies. For example, one of the sample files had a comment section that had field and value pairs in it. As such, regex was updated to ignore a field that starts with #.
6. Below is the regex used to extract the fields from the SPDX file. This was tested with the SBOM SPDX files used in the Healthcare Proof of Concept and worked as expected.

```
SPDXVersion = (?<!#)\s*SPDXVersion:\s*(?<SPDXVersion>.*)\s*DataLicense
DataLicense = (?<!#)\s*DataLicense:\s*(?<DataLicense>.*)\s*SPDXID
SPDXID-DOC = (?<!#)\s*SPDXID:\s*(?<SPDXID>.*)\s*DocumentName:
DocumentName = (?<!#)\s*DocumentName:\s*(?<DocumentName>.*)\s*DocumentNamespace
DocumentNamespace = (?<!#)\s*DocumentNamespace:\s*(?<DocumentNamespace>\S*)
Creator-Organization = (?<!#)\s*Creator:\s*Organization:\s*(?<Creator Organization>.*?)\s*(Creator:|Created:)
Creator-Person = (?<!#)\s*Creator:\s*Person:\s*(?<Creator Person>.*?)\s*(Creator:|Created:)
Creator-Tool = (?<!#)\s*Creator:\s*Tool:\s*(?<Creator Tool>.*?)\s*(Creator:|Created:)
CreatorComment = (?<!#)\s*CreatorComment:\s*<text>(?<CreatorComment>.*)</text>
PackageName = (?<!#)(?<PackageName>.*)(?<!#)\s*SPDXID, *PackageComment
SPDXID = (?<!#)\s*SPDXID:\s*(?<SPDXID>.*)\s*PackageComment
PackageComment = (?<!#)\s*PackageComment:\s*<text>(?<PackageComment>.*)</text>
PackageVersion = (?<!#)\s*PackageVersion:\s*(?<PackageVersion>.*)\s*PackageSupplier
PackageSupplier = (?<!#)\s*PackageSupplier:\s*(?:(Organization):\s*)|(?<PackageSupplier>.*)\s*Relationship
Relationship = (?<!#)\s*Relationship:\s*(?<Relationship>.*)\s*PackageDownloadLocation
PackageDownloadLocation = (?<!#)\s*PackageDownloadLocation:\s*(?<PackageDownloadLocation>.*)\s*FilesAnalyzed
FilesAnalyzed = (?<!#)\s*FilesAnalyzed:\s*(?<FilesAnalyzed>.*)\s*PackageLicenseConcluded
PackageLicenseConcluded = (?<!#)\s*PackageLicenseConcluded:\s*(?<PackageLicenseConcluded>.*)\s*PackageLicenseDeclared
PackageLicenseDeclared = (?<!#)\s*PackageLicenseDeclared:\s*(?<PackageLicenseDeclared>.*)\s*PackageCopyrightText
PackageCopyrightText = (?<!#)\s*PackageCopyrightText:\s*(?<PackageCopyrightText>.*?)\s*(#|)
```

Appendix B – Example SWID SBOM (ACME Corporation Road Runner Detector)

```
<SoftwareIdentity
xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd" name="ACME
Roadrunner Detector 2013 Coyote Edition SP1"tagId="com.acme.rrd2013-ce-sp1-v4-1-
5-0" version="4.1.5">
        <Entity name="The ACME Corporation" regid="acme.com" role="tagCreator
        softwareCreator"/>
        <Link rel="license" href="www.gnu.org/licenses/gpl.txt"/>
        <Meta product="Roadrunner Detector" colloquialVersion="2013"
        edition="coyote" revision="sp1"/>
        <Payload>
                <File name="rrdetector.exe" size="532712"
                SHA256:hash="a314fc2dc663ae7a6b6bc6787594057396e6b3f569
                cd50fd5ddb4d1bbafd2b6a"/>
</Payload>
</SoftwareIdentity>
```

Challenges

2.0

Cedars-Sinai

- The lack of a common way to uniquely identify software packages across SPDX files and vulnerability databases makes consistent mapping to vulnerabilities challenging. Fuzzy matching produces both false positive and false negative matches. False negatives will result in vulnerabilities going unidentified.
- Fields are not scalable for uniquely identify software packages. Cause challenges in cross file mapping and cause errors in mapping to known vulnerabilities

New York Presbyterian

- Agree with both #1 and #2 (above) and that cover nearly all of NYP challenges.
- Other challenge faced is more on the NVD side. It can be summed up as a challenge in true entity resolution due to a lack of detailed version information for vulnerability “sub-components” in the JSON feed.
- Another one, which is minor, are slight differences in the SPDX header schema across MDMs.

Challenges

2.0

Mayo Clinic

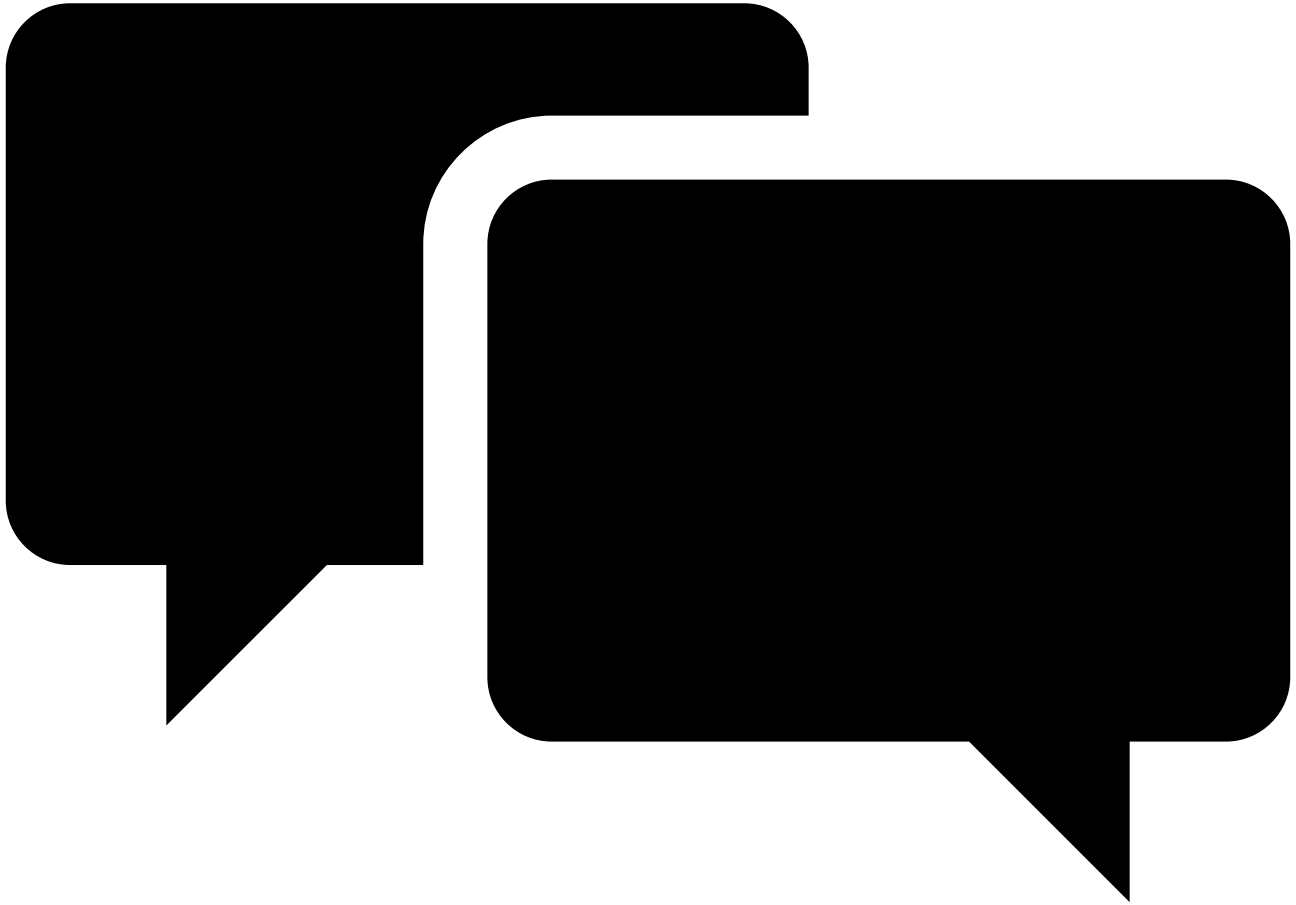
- Agree with above. There are other examples of inconsistent naming of software packages between the SBOM data and NVD that they discovered

Sutter Health

- The issue that their team run into is the software version naming convention of the SBOM. It is different from the format of the NVD which makes it a challenge for their team to yield any type of fidelity with their searches
- Splunk application scraped the NVD database and imported vulnerabilities each time it was ran. However, it does not have a continuous update function or a dedup function which creates a lot of manual labor on their end.
- They utilize Fortisoar (a SOAR platform) and configure a task to run each day: scrape NVD, dedupe vuln data, and push new data into Splunk. They have a dashboard inside Splunk that allows them to search vulnerabilities against the POC SBOMS that they have ingested.
- Running into the common issue that other HDOs have experienced. The naming convention is different between NVD data and SBOMs, which makes the search result unreliable

Conclusion & Looking Forward

JENNINGS ASKE, NEW YORK PRESBYTERIAN



Discussion

Questions? Comments? Suggestions? Volunteers?