

TXOne Networks

2022

/Q2

Supply Chain Security:  
**Are SBOMs Ready?**

TXOne Networks

Supply Chain Security:  
**Are SBOMs Ready?**

# Supply Chain Security: **Are SBOMs Ready?**

## Table of Contents

<b>Introduction</b> .....	4
<b>What is an SBOM?</b> .....	4
Software Component Lists.....	5
<b>Open Standards</b> .....	7
<b>Minimum Viable SBOM</b> .....	8
SPDX Example.....	9
SBOM Proofs-of-Concept.....	10
<b>Challenges to the SBOM</b> .....	11
<b>The OT Health Check</b> .....	11

## Introduction



As work sites globally fall victim to cyber attacks spreading through the supply chain, state governments, world industry leaders and cybersecurity experts have given supply chain security renewed and urgent focus. One security tool that is receiving a lot of attention recently as a potential bulwark against supply chain attacks is the SBOM (Software Bill of Materials), which was even mentioned in U.S. President Biden's Executive Order 14208 *Improving the Nation's Cybersecurity*. We expect it will fully revolutionize supply chain security when the technology is ready for full use and standardization.<sup>1</sup> In the meantime, organizations in verticals including pharmaceutical, energy, aerospace, and semiconductors have been using TXOne's *Trend Micro Portable Security 3* to make sure that all inbound and outbound assets are scanned, creating an OT health check to serve as proof of digital hygiene. Through asset shielding, vulnerabilities that would normally be challenging to track can be quickly and easily addressed without interrupting operations.

## What is an SBOM?



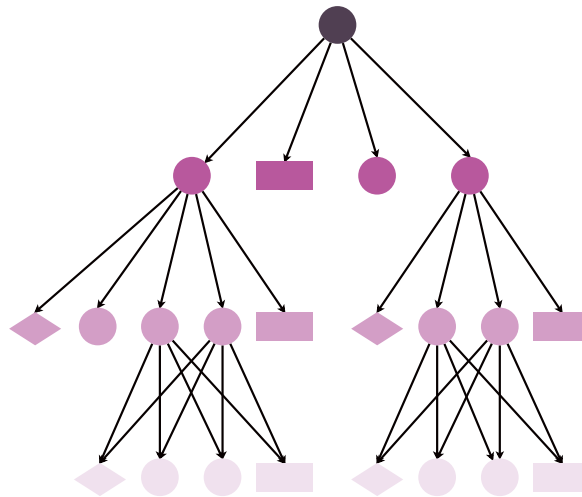
A **Software Bill of Materials (SBOM)** describes the components in software. In the wake of 2021's attack on a well-known pipeline company, Executive Order 14208 released in May of 2021 the United States government put this idea forward as a way that software developers can verify that the open-source and third-party software being included in their product is up-to-date and free of vulnerabilities. Software buyers could use an SBOM to perform vulnerability or license analysis that is, among other benefits, beneficial to risk evaluation, while software operators could quickly and easily determine whether a newly discovered vulnerability puts them at risk.

In order to realize all these gains, a widely-used, machine-readable SBOM format is needed along with a repository where an SBOM can be easily queried by other applications and systems. As part of this ecosystem, an SBOM could help security teams know when they are affected by unpredictable but high-risk threats like zero-day malware. In the future, an automated system could search through your inventory and let you know when a new threat arises – it would compare components used in your software to components with vulnerabilities, and recommend what action you should take. In the meantime, the needs of global infrastructure and enterprise security call for a different method of addressing similar concerns until SBOMs are completely ready for widespread adoption.

While The Linux foundation and others have created and continue to develop the necessary open standards, there may be a lot of ground to cover before SBOMs become fully practical. Lists of software components are available in popular programming languages, and there are domestic and international standards for SBOMs.

## Software Component Lists

Operational technology (OT) software automates the control of digital devices. A simple software program can flip a switch on or off. To flip more switches, run motors, control servos, and so forth, software modules are chained together to build more and more powerful programs. Because downstream modules are dependent on upstream modules, this chain forms a dependency tree.



**A dependency tree**

Programming languages require a developer to include a list of all the dependencies needed to build an app in a requirements file. In theory, this list of dependencies is the SBOM. In practice, requirements files are sometimes difficult to decipher due to missing details, references to other lists of requirements, and inconsistent formatting.

*Example 1*

SBOM	
SW library 1	1.0.1
SW library 2	1.1.1
SW library 3	4.2.1
SW library 4	2.0.0
SW library n	4.1.1

SBOM lists software components

*Example 2*

```
##### Requirements without Version Specifiers #####
Pytest software
Pytest-cov software
```

Missing details about software components

*Example 3*

```
##### Requirements with Version Specifiers #####
# see https://www.python.org/dev/peps/pep-0440/#version-specifiers
docopt == 0.6.1 # Version Matching. Must be version 0.6.1
keyring >= 4.1.1 # Minimum version 4.1.1
coverage != 3.5 # Version Exclusion. Anything except version 3.5
Mopidy-Dirble ~= 1.1 # Compatible release. Same as >= 1.1, == 1.*

##### Refer to other requirements files #####
~r other-requirements.txt

##### A particular file #####
./downloads/numpy-1.9.2-cp34-none-win32.whl
http://wxpython.org/Phoenix/snapshot-builds/wxPython_Phoenix-
3.0.3.dev1820+49a8884-cp34-none-win_amd64.whl
```

Components that refer to other components

*Example 4*

```

configurations {
    // A configuration meant for consumers that need the API of this
    component
    exposedApi {
        //This configuration is an "outgoing configuration, it's not
        meant to be resolved
        canBeConsumed = true
    }
    // A configuration meant for consumers that need the implementation
    of this component
    exposedRuntime {
        canBeResolved = false
        canBeConsumed = true
    }
}

```

**Different format for listing components**

There are licensing concerns, and identifying software modules is complicated by the lack of universal naming conventions.<sup>2</sup> Software developers and vendors often create products by assembling existing open source and commercial software components, which creates a situation where the different components can be challenging to track and verify as secure.

## Open Standards



SPDX and CycloneDX are two open standards projects underway aiming to create the standards necessary to automate the SBOM. SPDX is hosted by the Linux Foundation. The CycloneDX working group came from the OWASP community. Both projects address these minimum elements while maintaining flexibility and uniformity:

- **hash** – mathematical method for verifying the authenticity of a file
- **life cycle phase** – the maturity level of the software
- **relationships with other software dependencies**
- **software license**
- **SBOM integrity and authenticity**
- **vulnerabilities associated with the SBOM**
- **severity of associated vulnerabilities**
- **information about legacy software**
- **Cloud-based or SaaS-based concerns**

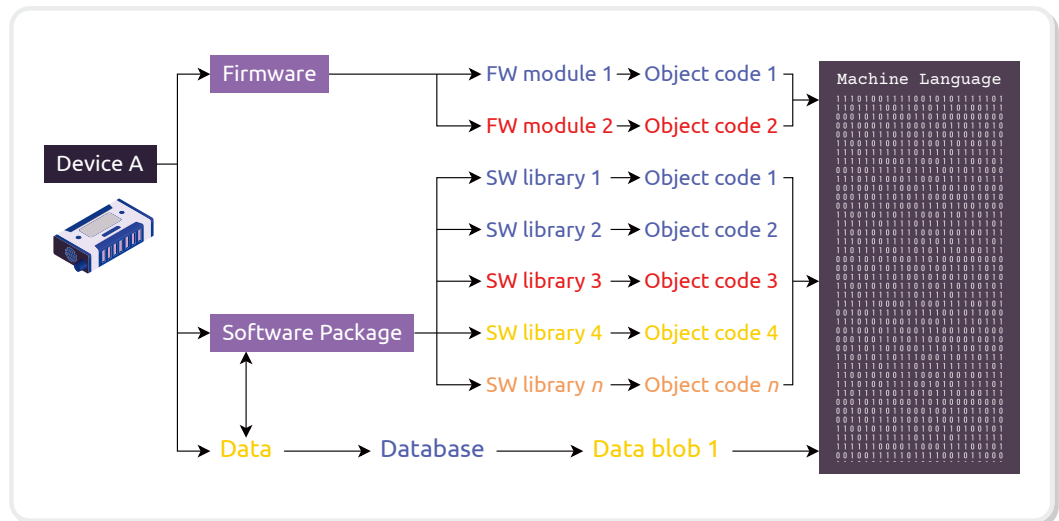
<sup>2</sup> Dr. Allan Friedman, Cybersecurity and Infrastructure Security Agency (CISA), "Benefits of an SBOM Across the Software Supply Chain", Mar 8 2022.

# Minimum Viable SBOM



To show how what a minimum viable SBOM would require, let's take "Device A" as an example. Device A is a smart inverter controlled by firmware (firmware is a special type of code that controls the hardware directly). For this example, assume this inverter communicates through the cloud.

The firmware, software, and data are color-coded depending on their known risk. Blue is safe, yellow is low risk, orange is medium risk, and red is high risk. Both the firmware and software were compiled from human-readable scripts (source code) into object code that only the machine can read. This is important because source code and object code can get out of sync during the development process if the configuration is not managed during the development life cycle.



The Software Bill of Materials for Device A is shown in json code. Json is a popular encoding technique used for automation. You can see that the json code has the risk encoded beside each component of the SBOM.

```
{ SBOM:Device A,[
  Software: [SW library 1: safe,
            SW library 2: safe,
            SW library 3: High Risk,
            SW library 4: low risk,
            SW library n: medium risk],

  Data:    [Data:safe],
  Firmware: [FW library 1: safe,
            FW library 2: High Risk]
]}
```

The goal of creating this fully-functioning JSON code is to demonstrate the purpose of the SBOM – codifying a security rating for each software component. Our simple example is missing key information and making the assumption that assessing the risk of vulnerability fits neatly into low, medium, and high risk categories. For practical use, the minimum viable SBOM must factor in consideration of architecture, use cases, and data formats so as to create metadata that includes details such as the frequency of SBOM generation, the depth of the software dependency tree, known gaps, known mistakes, and how to access SBOM data.

## SPDX Example

SPDX (Software Package Data Exchange) is one open standard SBOM. This example file taken from their GitHub shows one C source file with a simple “hello world” program, compiled via a Makefile into a single binary object with no dependencies.<sup>3</sup> Dependencies such as operating system kernel, C standard library, and so on are not addressed. The Makefile, source file, and binary object were included together as a single package for distribution. The dependency tree is simple, but this SBOM contains the minimum elements.

Dependency Tree	SBOM
<pre> content ├── build │   └── hello └── src     ├── Makefile     └── hello.c </pre>	<pre> SPDXVersion: SPDX-2.2 DataLicense: CC0-1.0 SPDXID: SPDXRef-DOCUMENT DocumentName: hello DocumentNamespace: https://swinslow.net/spdx-examples/example1/hello-v3 Creator: Person: Steve Winslow (steve@swinslow.net) Creator: Tool: github.com/spdx/tools-golang/builder Creator: Tool: github.com/spdx/tools-golang/idsearcher Created: 2021-08-26T01:46:00Z  ##### Package: hello  PackageName: hello SPDXID: SPDXRef-Package-hello PackageDownloadLocation: git+https://github.com/swinslow/spdx-examples.git#example1/content FilesAnalyzed: true PackageVerificationCode: 9d20237bb72087e87069f96afb41c6ca2fa2a342 PackageLicenseConcluded: GPL-3.0-or-later PackageLicenseInfoFromFiles: GPL-3.0-or-later PackageLicenseDeclared: GPL-3.0-or-later PackageCopyrightText: NOASSERTION  Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-hello  FileName: /build/hello SPDXID: SPDXRef-hello-binary FileType: BINARY FileChecksum: SHA1: 20291a81ef065ff891b537b64d4fdccaf6f5ac02 FileChecksum: SHA256: 83a33ff09648bb5fc5272baca88cf2b59fd81ac4cc6817b86998136af368708e FileChecksum: MD5: 08a12c966d776864cc1eb41fd03c3c3d LicenseConcluded: GPL-3.0-or-later LicenseInfoInFile: NOASSERTION FileCopyrightText: NOASSERTION </pre>

<sup>3</sup> The SPDX Team, “[spdx-examples / example1 / spdx / example1.spdx](https://github.com/spdx-examples/example1/spdx/example1.spdx)”, SPDX, Aug 26 2021, Accessed May 9 2022.

Dependency Tree	SBOM
	<pre> FileName: /src/hello.c SPDXID: SPDXRef-hello-src FileType: SOURCE FileChecksum: SHA1: 20862a6d08391d07d09344029533ec644fac6b21 FileChecksum: SHA256: b4e5ca56d1f9110ca94ed0bf4e6d9ac11c2186eb7cd95159c6fdb50e8db5a823 FileChecksum: MD5: 935054fe899ca782e11003bbae5e166c LicenseConcluded: GPL-3.0-or-later LicenseInfoInFile: GPL-3.0-or-later FileCopyrightText: Copyright Contributors to the spdx-examples project.  Relationship: SPDXRef-hello-binary GENERATED_FROM SPDXRef-hello-src Relationship: SPDXRef-hello-binary GENERATED_FROM SPDXRef-Makefile Relationship: SPDXRef-Makefile BUILD_TOOL_OF SPDXRef-Package-hello </pre>

## SBOM Proofs-of-Concept

The USA National Telecommunications and information Administration (NTIA) and the Idaho National Laboratory have undertaken SBOM proofs-of-concept for healthcare, automotive, and energy industry SBOMs using open standards.<sup>4</sup> The following chart compares the SBOM fields for the energy industry between the SPDX, SWID and CycloneDX formats.

Field	SPDX	SWID	CycloneDX
Supplier	(3.5) PackageSupplier:	<Entity> @role (softwareCreator/ publisher), @name	publisher
Component	(3.1) PackageName:	<softwareIdentity> @name	name
Unique Identifier	(3.2) SPDXID:	<softwareIdentity> @tagID	bom/serialNumber and component/ bom-ref
Version	(3.3) PackageVersion:	<softwareIdentity> @version	version
Component Hash	(3.10) PackageChecksum:	<Payload>/../<File> @[hash-algorithm]:hash	hash
Relationship	(7.1) Relationship: CONTAINS	<Link>@rel, @href	(Nested assembly/ subassembly and/or dependency graphs)
SBOM Author	(2.8) Creator:	<Entity> @role (tagCreator), @name	bom-descriptor: metadata/ manufacture/contact

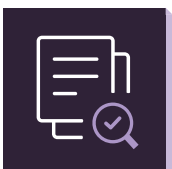
<sup>4</sup> National Telecommunications and Information Administration, "SOFTWARE BILL OF MATERIALS", United States Department of Commerce, Accessed on May 9 2022; Idaho National Laboratory, "Software Bill of Materials", Accessed on May 9 2022.

## Challenges to the SBOM



1. **Software Naming Standards** - One of the most difficult tasks in software development is naming the software. With no harmonized naming standards, how can we compare dependencies? How do dependencies map to vulnerability databases such as CVE and NVD? The problem is that there are too many software standards, and consensus is needed to create one that is unified.
2. **Software Dependencies** – There are many potential challenges to the discovery of software dependencies. Open-source software is published in public repositories, and there are different repos for different programming languages. Proprietary software is often not available to the public, some code is dynamically generated. SaaS and Cloud-based software rely on APIs to ingest and share data.
3. **Anti-SBOM Threats** - There are threat models that an SBOM would not prevent. For example, attackers corrupting the source code repository (as was seen in the SolarWinds attack of 2020) would have generated a corrupt SBOM. Even with an SBOM, OT-native defenses that can prevent unapproved updates and scan devices that come on site is a ground floor requirement for a secure facility.

## The OT Health Check



From looking at standards such as the semiconductor industry's new E187, we can see that practices such as OT verification are already in use successfully in many operational environments. For industry leaders in verticals including semiconductors, railway transportation, and aviation, the OT health check is already the standard for guaranteeing supply chain security.

Security inspection is a fact of life for any productive OT facility, and especially for those important to critical infrastructure. Regular security inspection of assets creates an OT health check that can be stored, referenced, and shared as necessary to create a trustworthy record of asset security.

1. **Scan new assets before onboarding** – The first scan begins the asset's scan record with its first OT health check and adds it to the asset inventory
2. **Scan outbound assets** – Verify that all outbound assets are free of malware before shipping and maintain a record for future use as necessary
3. **Store and review records centrally** - All scan records are centrally stored and managed from a single pane of glass to be easily searched and referenced as necessary

**Detailed asset information collected by Portable Security includes:**

- *Host ID number*
- *Host name*
- *Domain name*
- *Logged in account and domain*
- *MAC address*
- *IP*
- *OS description and type, including version, build, service pack, product ID, language, installation date and time, and previously installed updates*
- *Windows and system directories*
- *Internet Explorer version, build, service pack, and update version*
- *Vendor*
- *HW model and serial number*
- *BIOS version, date, and type*
- *Secure boot status*
- *CPU, CPU architecture*
- *Processors and cores*
- *Physical memory and availability of physical memory*
- *System drive size, available space, and boot drive*
- *Time zone, system date and time*

Secure your supply chain, keep records of digital hygiene, and maintain detailed asset knowledge with asset inventory and scan reports from TXOne's ***Trend Micro Portable Security 3***. While the SBOM and systems like it will be a tremendous boon to OT cyber defenses, knowing what software you are using is only half the battle. TXOne Networks' OT zero trust approach integrates software inventories with OT health checks or SBOMs (when available) to secure high-risk vulnerabilities in assets. Until SBOM technology is totally ready for widespread use, keeping your own detailed and easily-referenced OT health check records allows you to maintain client trust, monitor cyber risk levels, and much more easily track asset information.



[txone.com](https://txone.com)

Copyright © 2022 TXOne Networks. All rights reserved.





TXOne Networks

Supply Chain Security:  
**Are SBOMs Ready?**



# Supply Chain Security: **Are SBOMs Ready?**

## Table of Contents

<b>Introduction</b> .....	4
<b>What is an SBOM?</b> .....	4
Software Component Lists.....	5
<b>Open Standards</b> .....	7
<b>Minimum Viable SBOM</b> .....	8
SPDX Example.....	9
SBOM Proofs-of-Concept .....	10
<b>Challenges to the SBOM</b> .....	11
<b>The OT Health Check</b> .....	11

## Introduction



As work sites globally fall victim to cyber attacks spreading through the supply chain, state governments, world industry leaders and cybersecurity experts have given supply chain security renewed and urgent focus. One security tool that is receiving a lot of attention recently as a potential bulwark against supply chain attacks is the SBOM (Software Bill of Materials), which was even mentioned in U.S. President Biden's Executive Order 14208 *Improving the Nation's Cybersecurity*. We expect it will fully revolutionize supply chain security when the technology is ready for full use and standardization.<sup>1</sup> In the meantime, organizations in verticals including pharmaceutical, energy, aerospace, and semiconductors have been using TXOne's *Trend Micro Portable Security 3* to make sure that all inbound and outbound assets are scanned, creating an OT health check to serve as proof of digital hygiene. Through asset shielding, vulnerabilities that would normally be challenging to track can be quickly and easily addressed without interrupting operations.

## What is an SBOM?



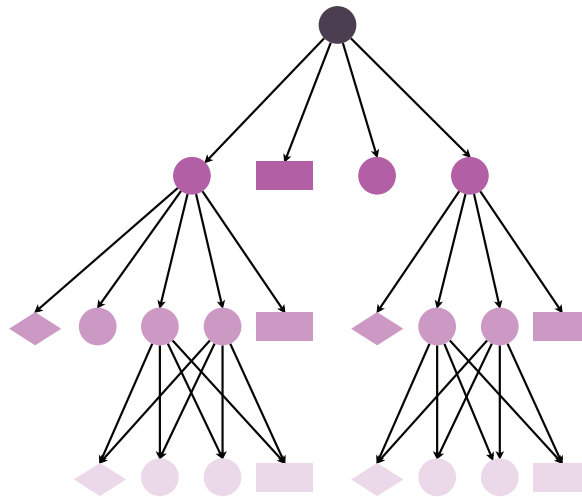
A **Software Bill of Materials (SBOM)** describes the components in software. In the wake of 2021's attack on a well-known pipeline company, Executive Order 14208 released in May of 2021 the United States government put this idea forward as a way that software developers can verify that the open-source and third-party software being included in their product is up-to-date and free of vulnerabilities. Software buyers could use an SBOM to perform vulnerability or license analysis that is, among other benefits, beneficial to risk evaluation, while software operators could quickly and easily determine whether a newly discovered vulnerability puts them at risk.

In order to realize all these gains, a widely-used, machine-readable SBOM format is needed along with a repository where an SBOM can be easily queried by other applications and systems. As part of this ecosystem, an SBOM could help security teams know when they are affected by unpredictable but high-risk threats like zero-day malware. In the future, an automated system could search through your inventory and let you know when a new threat arises – it would compare components used in your software to components with vulnerabilities, and recommend what action you should take. In the meantime, the needs of global infrastructure and enterprise security call for a different method of addressing similar concerns until SBOMs are completely ready for widespread adoption.

While The Linux foundation and others have created and continue to develop the necessary open standards, there may be a lot of ground to cover before SBOMs become fully practical. Lists of software components are available in popular programming languages, and there are domestic and international standards for SBOMs.

## Software Component Lists

Operational technology (OT) software automates the control of digital devices. A simple software program can flip a switch on or off. To flip more switches, run motors, control servos, and so forth, software modules are chained together to build more and more powerful programs. Because downstream modules are dependent on upstream modules, this chain forms a dependency tree.



**A dependency tree**

Programming languages require a developer to include a list of all the dependencies needed to build an app in a requirements file. In theory, this list of dependencies is the SBOM. In practice, requirements files are sometimes difficult to decipher due to missing details, references to other lists of requirements, and inconsistent formatting.

*Example 1*

SBOM	
SW library 1	1.0.1
SW library 2	1.1.1
SW library 3	4.2.1
SW library 4	2.0.0
SW library n	4.1.1

SBOM lists software components

*Example 2*

```
##### Requirements without Version Specifiers #####
Pytest software
Pytest-cov software
```

Missing details about software components

*Example 3*

```
##### Requirements with Version Specifiers #####
# see https://www.python.org/dev/peps/pep-0440/#version-specifiers
docopt == 0.6.1 # Version Matching. Must be version 0.6.1
keyring >= 4.1.1 # Minimum version 4.1.1
coverage != 3.5 # Version Exclusion. Anything except version 3.5
Mopidy-Dirble ~= 1.1 # Compatible release. Same as >= 1.1, == 1.*

##### Refer to other requirements files #####
~r other-requirements.txt

##### A particular file #####
./downloads/numpy-1.9.2-cp34-none-win32.whl
http://wxpython.org/Phoenix/snapshot-builds/wxPython_Phoenix-
3.0.3.dev1820+49a8884-cp34-none-win_amd64.whl
```

Components that refer to other components

*Example 4*

```

configurations {
    // A configuration meant for consumers that need the API of this
    component
    exposedApi {
        //This configuration is an "outgoing configuration, it's not
        meant to be resolved
        canBeConsumed = true
    }
    // A configuration meant for consumers that need the implementation
    of this component
    exposedRuntime {
        canBeResolved = false
        canBeConsumed = true
    }
}

```

**Different format for listing components**

There are licensing concerns, and identifying software modules is complicated by the lack of universal naming conventions.<sup>2</sup> Software developers and vendors often create products by assembling existing open source and commercial software components, which creates a situation where the different components can be challenging to track and verify as secure.

## Open Standards



SPDX and CycloneDX are two open standards projects underway aiming to create the standards necessary to automate the SBOM. SPDX is hosted by the Linux Foundation. The CycloneDX working group came from the OWASP community. Both projects address these minimum elements while maintaining flexibility and uniformity:

- **hash** – mathematical method for verifying the authenticity of a file
- **life cycle phase** – the maturity level of the software
- **relationships with other software dependencies**
- **software license**
- **SBOM integrity and authenticity**
- **vulnerabilities associated with the SBOM**
- **severity of associated vulnerabilities**
- **information about legacy software**
- **Cloud-based or SaaS-based concerns**

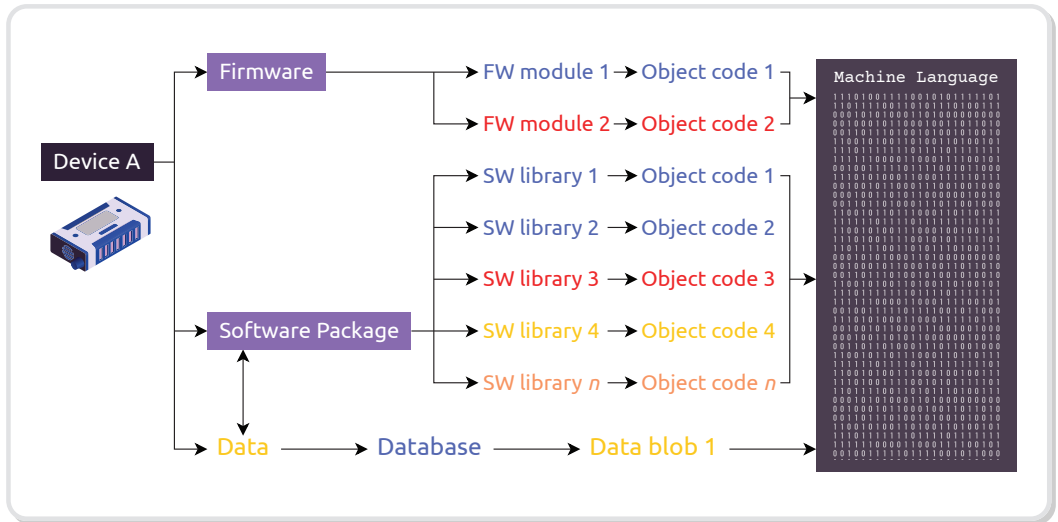
<sup>2</sup> Dr. Allan Friedman, Cybersecurity and Infrastructure Security Agency (CISA), "Benefits of an SBOM Across the Software Supply Chain", Mar 8 2022.

# Minimum Viable SBOM



To show how what a minimum viable SBOM would require, let’s take “Device A” as an example. Device A is a smart inverter controlled by firmware (firmware is a special type of code that controls the hardware directly). For this example, assume this inverter communicates through the cloud.

The firmware, software, and data are color-coded depending on their known risk. Blue is safe, yellow is low risk, orange is medium risk, and red is high risk. Both the firmware and software were compiled from human-readable scripts (source code) into object code that only the machine can read. This is important because source code and object code can get out of sync during the development process if the configuration is not managed during the development life cycle.



The Software Bill of Materials for Device A is shown in json code. Json is a popular encoding technique used for automation. You can see that the json code has the risk encoded beside each component of the SBOM.

```
{ SBOM:Device A,[
  Software: [SW library 1: safe,
            SW library 2: safe,
            SW library 3: High Risk,
            SW library 4: low risk,
            SW library n: medium risk],

  Data:    [Data:safe],
  Firmware: [FW library 1: safe,
            FW library 2: High Risk]
]}
```

The goal of creating this fully-functioning JSON code is to demonstrate the purpose of the SBOM – codifying a security rating for each software component. Our simple example is missing key information and making the assumption that assessing the risk of vulnerability fits neatly into low, medium, and high risk categories. For practical use, the minimum viable SBOM must factor in consideration of architecture, use cases, and data formats so as to create metadata that includes details such as the frequency of SBOM generation, the depth of the software dependency tree, known gaps, known mistakes, and how to access SBOM data.

## SPDX Example

SPDX (Software Package Data Exchange) is one open standard SBOM. This example file taken from their GitHub shows one C source file with a simple “hello world” program, compiled via a Makefile into a single binary object with no dependencies.<sup>3</sup> Dependencies such as operating system kernel, C standard library, and so on are not addressed. The Makefile, source file, and binary object were included together as a single package for distribution. The dependency tree is simple, but this SBOM contains the minimum elements.

Dependency Tree	SBOM
<pre> content ├── build │   └── hello └── src     ├── Makefile     └── hello.c </pre>	<pre> SPDXVersion: SPDX-2.2 DataLicense: CC0-1.0 SPDXID: SPDXRef-DOCUMENT DocumentName: hello DocumentNamespace: https://swinslow.net/spdx-examples/example1/hello-v3 Creator: Person: Steve Winslow (steve@swinslow.net) Creator: Tool: github.com/spdx/tools-golang/builder Creator: Tool: github.com/spdx/tools-golang/idsearcher Created: 2021-08-26T01:46:00Z  ##### Package: hello  PackageName: hello SPDXID: SPDXRef-Package-hello PackageDownloadLocation: git+https://github.com/swinslow/spdx-examples.git#example1/content FilesAnalyzed: true PackageVerificationCode: 9d20237bb72087e87069f96afb41c6ca2fa2a342 PackageLicenseConcluded: GPL-3.0-or-later PackageLicenseInfoFromFiles: GPL-3.0-or-later PackageLicenseDeclared: GPL-3.0-or-later PackageCopyrightText: NOASSERTION  Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-hello  FileName: /build/hello SPDXID: SPDXRef-hello-binary FileType: BINARY FileChecksum: SHA1: 20291a81ef065ff891b537b64d4fdccaf6f5ac02 FileChecksum: SHA256: 83a33ff09648bb5fc5272baca88cf2b59fd81ac4cc6817b86998136af368708e FileChecksum: MD5: 08a12c966d776864cc1eb41fd03c3c3d LicenseConcluded: GPL-3.0-or-later LicenseInfoInFile: NOASSERTION FileCopyrightText: NOASSERTION </pre>

<sup>3</sup> The SPDX Team, “[spdx-examples / example1 / spdx / example1.spdx](https://github.com/spdx-examples/example1/spdx/example1.spdx)”, SPDX, Aug 26 2021, Accessed May 9 2022.

Dependency Tree	SBOM
	<pre> FileName: /src/hello.c SPDXID: SPDXRef-hello-src FileType: SOURCE FileChecksum: SHA1: 20862a6d08391d07d09344029533ec644fac6b21 FileChecksum: SHA256: b4e5ca56d1f9110ca94ed0bf4e6d9ac11c2186eb7cd95159c6fdb50e8db5a823 FileChecksum: MD5: 935054fe899ca782e11003bbae5e166c LicenseConcluded: GPL-3.0-or-later LicenseInfoInFile: GPL-3.0-or-later FileCopyrightText: Copyright Contributors to the spdx-examples project.  Relationship: SPDXRef-hello-binary GENERATED_FROM SPDXRef-hello-src Relationship: SPDXRef-hello-binary GENERATED_FROM SPDXRef-Makefile Relationship: SPDXRef-Makefile BUILD_TOOL_OF SPDXRef-Package-hello </pre>

## SBOM Proofs-of-Concept

The USA National Telecommunications and information Administration (NTIA) and the Idaho National Laboratory have undertaken SBOM proofs-of-concept for healthcare, automotive, and energy industry SBOMs using open standards.<sup>4</sup> The following chart compares the SBOM fields for the energy industry between the SPDX, SWID and CycloneDX formats.

Field	SPDX	SWID	CycloneDX
Supplier	(3.5) PackageSupplier:	<Entity> @role (softwareCreator/ publisher), @name	publisher
Component	(3.1) PackageName:	<softwareIdentity> @name	name
Unique Identifier	(3.2) SPDXID:	<softwareIdentity> @tagID	bom/serialNumber and component/ bom-ref
Version	(3.3) PackageVersion:	<softwareIdentity> @version	version
Component Hash	(3.10) PackageChecksum:	<Payload>/../<File> @[hash-algorithm]:hash	hash
Relationship	(7.1) Relationship: CONTAINS	<Link>@rel, @href	(Nested assembly/ subassembly and/or dependency graphs)
SBOM Author	(2.8) Creator:	<Entity> @role (tagCreator), @name	bom-descriptor: metadata/ manufacture/contact

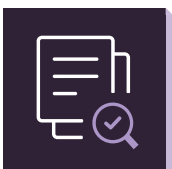
<sup>4</sup> National Telecommunications and Information Administration, "SOFTWARE BILL OF MATERIALS", United States Department of Commerce, Accessed on May 9 2022; Idaho National Laboratory, "Software Bill of Materials", Accessed on May 9 2022.

## Challenges to the SBOM



1. **Software Naming Standards** - One of the most difficult tasks in software development is naming the software. With no harmonized naming standards, how can we compare dependencies? How do dependencies map to vulnerability databases such as CVE and NVD? The problem is that there are too many software standards, and consensus is needed to create one that is unified.
2. **Software Dependencies** – There are many potential challenges to the discovery of software dependencies. Open-source software is published in public repositories, and there are different repos for different programming languages. Proprietary software is often not available to the public, some code is dynamically generated. SaaS and Cloud-based software rely on APIs to ingest and share data.
3. **Anti-SBOM Threats** - There are threat models that an SBOM would not prevent. For example, attackers corrupting the source code repository (as was seen in the SolarWinds attack of 2020) would have generated a corrupt SBOM. Even with an SBOM, OT-native defenses that can prevent unapproved updates and scan devices that come on site is a ground floor requirement for a secure facility.

## The OT Health Check



From looking at standards such as the semiconductor industry's new E187, we can see that practices such as OT verification are already in use successfully in many operational environments. For industry leaders in verticals including semiconductors, railway transportation, and aviation, the OT health check is already the standard for guaranteeing supply chain security.

Security inspection is a fact of life for any productive OT facility, and especially for those important to critical infrastructure. Regular security inspection of assets creates an OT health check that can be stored, referenced, and shared as necessary to create a trustworthy record of asset security.

1. **Scan new assets before onboarding** – The first scan begins the asset's scan record with its first OT health check and adds it to the asset inventory
2. **Scan outbound assets** – Verify that all outbound assets are free of malware before shipping and maintain a record for future use as necessary
3. **Store and review records centrally** - All scan records are centrally stored and managed from a single pane of glass to be easily searched and referenced as necessary

**Detailed asset information collected by Portable Security includes:**

- *Host ID number*
- *Host name*
- *Domain name*
- *Logged in account and domain*
- *MAC address*
- *IP*
- *OS description and type, including version, build, service pack, product ID, language, installation date and time, and previously installed updates*
- *Windows and system directories*
- *Internet Explorer version, build, service pack, and update version*
- *Vendor*
- *HW model and serial number*
- *BIOS version, date, and type*
- *Secure boot status*
- *CPU, CPU architecture*
- *Processors and cores*
- *Physical memory and availability of physical memory*
- *System drive size, available space, and boot drive*
- *Time zone, system date and time*

Secure your supply chain, keep records of digital hygiene, and maintain detailed asset knowledge with asset inventory and scan reports from TXOne's **Trend Micro Portable Security 3**. While the SBOM and systems like it will be a tremendous boon to OT cyber defenses, knowing what software you are using is only half the battle. TXOne Networks' OT zero trust approach integrates software inventories with OT health checks or SBOMs (when available) to secure high-risk vulnerabilities in assets. Until SBOM technology is totally ready for widespread use, keeping your own detailed and easily-referenced OT health check records allows you to maintain client trust, monitor cyber risk levels, and much more easily track asset information.



[txone.com](https://txone.com)

Copyright © 2022 TXOne Networks. All rights reserved.

