opentext™ | Cybersecurity

Position Paper

The Need for a Software Bill of Materials

Whether you produce, purchase, or operate software, insights into the supply chain will provide you with a range of benefits.

Table of Contents

The Software Bill of Materials 1
What Is a Software Bill of Materials? 1
Drivers, Motivators, and Challenges 5
The Software Bill of Materials: The SBOM File 10
The Software Bill of Materials: SBOM with Debricked 16
Conclusion 20

The Software Bill of Materials

SBOM helps provide numerous insights to an organization. In this position paper, we will discuss several aspects of the SBOM, including benefits and drivers for adoption, and dig a bit deeper into the actual SBOM files and formats. But let us start with defining what an SBOM is.

Simply put, the Software Bill of Materials (SBOM) is a listing of all software dependencies that are included in a software application. It includes not only the direct dependencies used but also the dependencies used by those dependencies, also known as indirect or transitive dependencies. As such, it describes the supply chain relationships used when building the software.

What Is a Software Bill of Materials?

Simply put, the Software Bill of Materials (SBOM) is a listing of all software dependencies that are included in a software application. It includes not only the direct dependencies used but also the dependencies used by those dependencies, also known as indirect or transitive dependencies. As such, it describes the supply chain relationships used when building the software.

A List of Ingredients

Just like food in the grocery store has a list of ingredients written on the package, we can think of the SBOM as a list of ingredients for a software application. For people with allergies, the list of ingredients can be used to verify that it does not contain anything unwanted.

Often, people may want to stay away from unethical or unhealthy content or things with too many unnatural chemicals used only for preservation, color, or profit. The list is mandatory since we want to allow people to make informed decisions about the food they buy. The transparency also puts pressure on the manufacturer to not include unnecessary bad ingredients since the food and the manufacturer can now be judged by the ingredients.

The SBOM serves a very similar purpose. By listing all packages included in a software application, users will be able to make informed decisions about which applications to use based on the included packages, and the developers will be incentivized to use up-to-date, secure, and well-maintained software.

Not Just Ingredients

The analogy to ingredients is often used. Yes, it will show you the components that the software product consists of. But it does not stop there. Looking at the most common SBOM formats used today, there is also support for adding valuable metadata about the components.

This metadata can consist of details on known vulnerabilities for the component. It can also be detailed license information, i.e., the requirements and the restrictions for including the component in another piece of software. The metadata can also include how the different components fit together, i.e., which component depends on other components. If these relationships are complete, the SBOM can provide the full dependency graph for all components in the software.

Thus, while the ingredients analogy is easy to grasp, there can be quite a lot more to it if the SBOM capabilities are fully used.

Benefits and Use Cases

The SBOM can be used to provide insights into your software. It is an invaluable enabler for several business-critical operations related to software development, software management and software consumption across the value chain.

Not a Silver Bullet

Before discussing the benefits, we note that the SBOM does not really solve any problems on its own. It needs to be accompanied by organizational processes to take advantage of the data it holds. With technical tools and automations, you will be able to collect, present, and add business value to the data in the SBOM.

This will make the data actionable and improve software and product security. It will also allow organizations to be compliant with both licenses and security requirements. Assuming such tools and processes are in place, let's look at some of the benefits the SBOM will give you.

Security

The main claim for success is risk management and risk reduction, with security being the most well-known use case. It is easy to argue for the security case. We all want to avoid a costly data breach. In 2022, the average cost of a data breach was estimated to be $4.24 million. At the same time, together with phishing, using known vulnerabilities are the two main attack vectors seen today. Now, add to this that the number of discovered vulnerabilities is constantly increasing.

The main claim for success is risk management and risk reduction, with security being the most well-known use case. It is easy to argue for the security case. We all want to avoid a costly data breach.

With the SBOM listing all software dependencies, it is possible and feasible to assess if any of these dependencies have known security vulnerabilities. And if they do, we know to patch them. Without the SBOM, or at least without the detailed insights into the supply chain that the SBOM provides, there would be no way of really knowing if the software is vulnerable or not.

This is a game changer for those purchasing and using the software. If there is a new vulnerability, they can immediately assess if they are exposed.

License Compliance

Another benefit is license compliance. Every time we include code written by someone else, e.g., Open-Source Software (OSS), we are using copyrighted code. We cannot use that code without a license. The license will tell us what we are allowed to do with the code and under what circumstances.

In some cases, the restrictions and our obligations are rather heavy if we want to include the code in distributed software. With the SBOM, we get insights into third-party dependencies. Then we can also know what licenses apply to the different dependencies. These licenses can also be written directly in the SBOM.

Dependency Health

Security and license compliance are the two benefits that are most often discussed in the SBOM context. At the same time, we see that the use of OSS is increasing, and today's codebases have around 80–90% OSS. This increased dependency on OSS presents new challenges, some of which the SBOM can help meet.

One thing that many organizations are struggling with is how to choose the best OSS component for a specific task. There can be lots of OSS projects supporting similar functionality, so which one should we choose? This question is more important than it may seem at first. You want a project that has ongoing community support, not one that was or will be abandoned soon. You also want a project that will patch vulnerabilities, otherwise, there is no safe version to upgrade to, and you must patch the source yourself.

You may also want to choose a project that engages experienced developers, a project with reasonable documentation, and perhaps a project with an active core team. Though there are no current security vulnerabilities or license compliance risks, all these properties will contribute to a forward risk.

Having a software inventory through the SBOM will help in analyzing the software dependencies for such forward risks. An automated tool, such as **OpenText™ Debricked**, will automatically scan the SBOM and present you with a range of metrics that will help you understand the health of your software dependencies.

Having a software inventory through the SBOM will help in analyzing the software dependencies for such forward risks. An automated tool, such as **Debricked**, will automatically scan the SBOM and present you with a range of metrics that will help you understand the health of your software dependencies.

Increased Transparency

The benefits do not stop here. Using the data to assess security, license compliance, and health can be seen as very direct benefits. But we also need to consider the effect of having to supply an SBOM when software is distributed or sold to customers. With the SBOM, the software is no longer a black box. There is transparency in what you deliver.

The software provider can no longer hide bad practices when it comes to patching and vetting the included software, and license compliance need to be top-of-mind to avoid facing legal problems.

When customers have insight into the components of an application, they can also check for security vulnerabilities, license compliance, and scrutinize the software for out-of-date and unsupported components. And by doing this, they can judge their suppliers by their practices in choosing and maintaining dependencies.

This clearly incentivizes better practices on the supplier's side. Security vulnerabilities will affect the customer if they are exploited, so the customer can put pressure on the supplier to have patched software in the applications. This will lead to better, more secure, and compliant software.

Fixing security problems is more costly the later they are done. Updating to a secure version of a dependency can be easily done at development time. If you do it later, there will be added complexity. Updating software that is in production or that has already been distributed can be very costly.

Stronger Supplier-Customer Relationships

The supplier can also use the SBOM as a chance to get stronger relationships with their customers. Consider an organization that chooses between two suppliers, one of them is able to provide a detailed and up to date SBOM, while the other is not willing or able to do so. As a customer, which one would you choose?

In one case, you will be in control of vetting the software yourself if you wish, and the supplier is also incentivized to have good software practices for their third-party components.

In the other case, you are buying a black box without any possibility of scrutinizing the application's components. And why are they not providing an SBOM? Is it because they just don't have the tools or knowledge to produce one, or is it because the software has known vulnerabilities? Or do they not know if there are vulnerabilities or not? Are they using tons of outdated software? Do they even know if they do? None of the reasons are very flattering, and all other things equal, the supplier would surely go for the supplier that provides an SBOM.

The SBOM will also facilitate an ongoing discussion between the supplier and the customer. Why did you choose this software? Are we vulnerable to this new CVE related to an included component? Yes, there will likely be more questions from customers, some good and some less relevant, but it is a chance for the supplier to show good practices throughout the software lifecycle. This will increase confidence in the supplier and improve the relationship between the customer and the supplier.

Reduce Remediation Costs and Time-to-Market

Fixing security problems is more costly the later they are done. Updating to a secure version of a dependency can be easily done at development time. If you do it later, there will be added complexity. Updating software that is in production or that has already been distributed can be very costly.

Using SBOMs and an accompanying process for keeping track of vulnerabilities, licenses, and health information will allow developers to find problems quickly. This will also reduce the remediation cost. In fact, having an SCA tool for keeping track of all these things related to dependencies will probably quickly pay off when vulnerabilities, licenses, and health are continuously monitored.

With carefully considered choices for third-party dependencies, there will hopefully be fewer problems with this software in the future. This includes fewer vulnerabilities, faster patch processes, no license issues, and better-maintained software. Less added complexity will allow developers to focus more on performance, stability, user experience, and added features. In the end, this will reduce the time-to-market and allow the supplier to be more competitive.

Concluding
The SBOM presents several benefits to all stakeholders. Though the pure benefits should be enough to immediately adopt SBOMs, this is often not enough to push organizations over the edge. Adoption sometimes requires a push from governments and authorities. In the next section, we will discuss these drivers as well as the emerging threat landscape and the challenges presented when faced with SBOM adoption.

Adoption sometimes requires a push from governments and authorities. In the next section, we will discuss these drivers as well as the emerging threat landscape and the challenges presented when faced with SBOM adoption.

Drivers, Motivators, and Challenges

SBOMs are not new but have received an increased interest recently. For many organizations, it has gone from being a nice-to-have thing to a must-have. This shift is driven partly by new compliance requirements and, in part, by the cybersecurity threat landscape.

The many benefits discussed earlier, both for suppliers and customers, have been significant drivers for the popularity of SBOMs. Still, working with an SBOM presents a set of challenges to be aware of and to overcome. In this section, we take a more detailed look at the drivers, motivators, and challenges for the usage.

Compliance and Regulatory Requirements
New regulations and requirements have appeared from a range of different organizations, governments, and similar. These requirements are in response to the many supply chain attacks that we have witnessed over the last few years.

Cybersecurity Executive Order
Perhaps the one that is most cited is the Biden cybersecurity executive order from May 2021. It is noted that the private sector needs to step up the game if they are to provide systems to the U.S. Federal Government. To enhance software supply chain security, the order lists a set of requirements that need to be fulfilled for these suppliers.

One part of the order discusses SBOMs and specifically requires that the purchaser is provided an SBOM together with the purchased software. At the same time, the National Telecommunications and Information Administration (NTIA) was tasked to create a list of the minimum required elements of such an SBOM.

Proposed DHS Law

Related is the H.R.4611—DHS Software Supply Chain Risk Management Act of 2021, which is a proposed law that will require contractors to the Department of Homeland Security (DHS) to submit an SBOM together with a certification that there are no security vulnerabilities in the software. Alternatively, if there are known vulnerabilities, they must provide a list of these.

The EU Cyber Resilience Act

In the EU, there is a proposal for a regulation for cybersecurity requirements, the Cyber Resilience Act. Regulations are mandatory to follow for all member states. Among other things, the Cyber Resilience Act requires manufacturers to draw up an SBOM. Different from the U.S. regulations, this EU regulation will apply to all manufacturers of products with digital elements that connect to a device or a network. On the other hand, only top-level dependencies need to be included in the SBOM.

FDA Requirement

For specific markets, the FDA is currently pushing for an SBOM to be a mandatory requirement for healthcare products. This is in response to an increased number of cybersecurity incidents in healthcare, as, e.g., reported by Forbes. Moreover, patient data protected by healthcare products are typically very sensitive, and service disruption by these products can jeopardize the life of people.

Other Guidelines

In addition, guidelines from the National Highway Traffic Safety Administration mention SBOM as a means to track vulnerabilities in the vehicle development process. These guidelines are non-binding and voluntary but underline the importance perceived throughout several verticals.

The Cybersecurity Threat Landscape

Requirements and legislation will drive the adoption, but these requirements emerge from the actual need in industry and society. The cybersecurity threat landscape is present with or without regulations, and many businesses adopt SBOM practices regardless of external requirements. Let us take a brief look at the cybersecurity threat landscape and how it is developing.

New Vulnerabilities

First, the number of vulnerabilities registered as CVEs in the National Vulnerability Database is increasing. In 2017, the number of new vulnerabilities jumped to more than 14,000 after previously never exceeding 8,000 in a year. Since then, the number has steadily increased, and in 2022 it surpassed 25,000.

There are more vulnerabilities if we include the GitHub Advisory Database and those that are language specific, e.g., FriendsOfPHP and the Python Packaging Advisory Database, but there are significant overlaps.

Requirements and legislation will drive the adoption, but these requirements emerge from the actual need in industry and society. The cybersecurity threat landscape is present with or without regulations, and many businesses adopt SBOM practices regardless of external requirements.

Exploiting Vulnerabilities Is a Common Attack Vector
A known vulnerability can be used as an attack vector in a breach. With many vulnerabilities across a range of applications, there are more opportunities to mount attacks. Surely enough, looking at the top attack vectors as observed by IBM Security X-Force in the 2022 report, 34% was due to exploiting vulnerabilities, second only to phishing. Thus, fixing security vulnerabilities must be top-of-mind for organizations relying on software applications in their business.

Cost of Breaches
So, clearly, there are not only breaches due to security vulnerabilities, but they are prevalent. Add to this; a breach is very costly. The global average cost of a data breach caused by a vulnerability in third-party components is estimated to be $4.55 million. If you do not take application security seriously, it is just a matter of time before it happens.

In all, the cybersecurity threat landscape calls for investing in application security. The alternative is just too costly. With assessing and remediating security vulnerabilities being a top SBOM use case, it is natural to adopt it.

Reliance on Software
Software is shaping our society, and every day we have become increasingly reliant on software. In the smart city, we try to optimize for sustainability and efficiency through sensors, actuators, databases, communications, and processing.

The data that is collected, processed, and stored will often be sensitive, so we need confidentiality. Also, integrity protection is needed to ensure that the data is not modified in transit or at rest. All parts and their functionality are controlled by software.

Since software influences how we live and work, the need to have better insights into its inner workings becomes more important. The SBOM can be used to provide at least parts of this insight.

Challenges
From the previous discussion, it should be clear that SBOMs are here to stay. But, when generating and working with SBOMs, there are several challenges to consider. It's not just to generate an SBOM and call it a day. Having an SBOM is not worth much if you cannot, or do not, use it for its intended purposes.

Completeness
Completeness refers to the SBOM including all data that is expected. Looking at the various SBOM formats, there is support for many different entries. A complete SBOM does not have to include all this data. Instead, it does have to cover all software components that it sets out to include. Moreover, if there is information for a component that can be expected to be included, this must be included.

When generating and working with SBOMs, there are several challenges to consider. It's not just to generate an SBOM and call it a day. Having an SBOM is not worth much if you cannot, or do not, use it for its intended purposes.

Missing Components

If information is missing, e.g., there is an open-source software component that is used but not included in the SBOM, then this poses a risk to the receiving organization. It could mean critical vulnerabilities that cannot be listed and assessed. It can also mean that the application uses a component with a non-permissive license in a way that violates the license. In addition to the security and license compliance risks, incomplete SBOMs will reduce the trust in the provider and can delay the time-to-market for an application.

Missing Information

The same is true for open-source components that are included, but information about the component is incomplete. In many cases, vulnerability information is written directly in the SBOM. Then, if vulnerability information is only taken from NVD, there will likely be vulnerabilities that are present but not included.

Known Unknowns

It can be argued that an incomplete SBOM can be worse than no SBOM at all. If we think the SBOM is complete, we will have a false sense of security, perhaps letting the guard down and not being fully prepared to handle an exploited security vulnerability. With knowledge of a vulnerability, even if it is not patched, other measures can be taken to avoid exploitation and breaches.

To help with “known unknowns,” the common SBOM formats have support for indicating if a dependency relationship is (possibly) incomplete or if all relations have been accounted for.

Up to Date

An SBOM is not a one-off thing. It is a moving target that needs to be kept up to date. Having an outdated SBOM comes with the same risks as having an incomplete one, erroneous data.

The SBOM can become outdated for different reasons. An application continuously developed and updated will soon have an outdated SBOM. New dependencies will be used, some will be updated to newer versions, while others might be removed.

Any assessments based on outdated SBOMs risk having errors. Vulnerabilities can be missed, while some might already be fixed. The first is a security problem, and the latter gives overhead for developers and security analysts since there will be false positives in the assessment.

Outdated External Data

The SBOM can also be outdated in terms of the external data it can provide. Security vulnerabilities are constantly discovered. If the SBOM includes a list of known vulnerabilities, e.g., CVE identifiers, such a list will be outdated as soon as there is a new vulnerability affecting any of the included components.

Any assessments based on outdated SBOMs risk having errors. Vulnerabilities can be missed, while some might already be fixed. The first is a security problem, and the latter gives overhead for developers and security analysts since there will be false positives in the assessment.

This should come as no surprise and looking at the guidelines for how to use the SPDX specification, it is even explicitly stated that “SPDX consumers should always assume vulnerabilities enumerated by an SPDX creator to be out-of-date.” The need for having up to date SBOMs makes it important also to include a timestamp.

Automation and SCA

To help generate the SBOM, automation is almost always necessary. There are just too many dependencies in software today, and there is too much information that needs to be collected and to keep up to date to do it manually. An automated tool is less error-prone and can generate a full SBOM in a fraction of the time compared to manual processes.

Instead of having to constantly update the SBOM due to external changes, an SCA tool can be used to keep track of vulnerabilities, alert you when they arise, and even help you to fix them. This will always provide an up-to-date view of the risks. For developers, by integrating the code repositories with the SCA tool, the view will also update when there are new or updated components.

The current threat landscape with an increasing number of vulnerabilities and attacks should be enough drivers for adopting SBOMs on a wider scale. If that is not enough, the push from regulations and authorities will surely help organizations in the right direction.

Actionable

The SBOM is useless if the information in it is not used. It cannot do anything on its own, which is why it is crucial that it is actionable. This means that both the content of the SBOM needs to be in a format that can be easily consumed and that its content can be used for the use case it is generated for. It also means that there need to be organizational processes in place to use the SBOM when it is received.

Targeting the Use Case

An SBOM with only license information could be sufficient if only license compliance is considered, but not if you need to certify that there are no vulnerabilities. If you want to use the SBOM to create an attribution report for your use of open-source software, the license text also needs to be included. It is not enough with the license name.

Concluding

The current threat landscape with an increasing number of vulnerabilities and attacks should be enough drivers for adopting SBOMs on a wider scale. If that is not enough, the push from regulations and authorities will surely help organizations in the right direction.

However, as we have seen, it is not just to turn a switch and have everything working in two shakes of a lamb’s tail. Some challenges need to be considered for a purposeful deployment.

To help push forward, to have automation, and to have interoperability between organizations, there are a few well-defined formats for encoding the SBOM information.

The leading formats, SPDX and CycloneDX, will be described in the next section.

The Software Bill of Materials: The SBOM File

There are a few different formats for storing and encoding SBOM information. The most well-known targeting supply chain transparency is the SPDX and the CycloneDX formats.

In this post, we take a deeper dive into these formats and provide a comparison between them. We also briefly discuss the SWID tags, which can also be used for SBOM information, but has a somewhat different target use case.

The vast number of components, and their relations, require tools support for both reading and generating the SBOM.

NTIA Minimum Elements

The Cybersecurity Executive Order instructed (among others) the National Telecommunications and Information Administration (NTIA) to publish a set of minimum elements for an SBOM. These elements are divided into three categories.

- Data fields
- Automation support
- Practices and processes

Let us discuss these categories in a little more detail.

Data Fields

The data fields define what data an SBOM should include. This is the minimum amount of information required for each component, as well as metadata for the SBOM file itself. Seven data fields are defined. These are the *supplier* of a component, the *component name*, its *version*, *other unique identifiers*, the *relationship between the dependencies*, i.e., which upstream components are used by a component, the *author of the SBOM*, and a *timestamp*.

Having other unique identifiers will allow the component information to be mapped to known vulnerabilities and licenses. Such mappings assume that the component is not confused with other components of a similar name. The main unique identifiers are CPE, PURL, and SWID.

Automation Support

The vast number of components, and their relations, require tools support for both reading and generating the SBOM. Automation and tools support will also ensure interoperability between organizations. Since SBOMs will often be provided from a supplier to a purchaser/consumer, such interoperability is crucial for its usage.

While automation requires a machine-readable format, the SBOM should also be human-readable. This will help with manual troubleshooting and a quick overview of certain specific data in the SBOM. To support these requirements, NTIA mandates using one of the SPDX, CycloneDX, or SWID data formats for an SBOM. This list might be expanded in the future, but proprietary formats should be explicitly avoided.

Practices and Processes

NTIA defines several minimum requirements for the processes surrounding the creation and management of SBOMs. Related to the frequency of generating an SBOM, it must be generated every time there is a new software release.

The dependencies used in software can be seen as a tree hierarchy, with the direct dependencies at the top and the upstream transitive dependencies below. At a bare minimum, the SBOM must include all top-level direct dependencies. These should be provided with enough detail so that it is possible to find the transitive dependencies. Additionally, it must be clear if there are no further transitive dependencies or if the presence of such dependencies is unknown.

NTIA also highlights the importance of starting with generating and providing SBOMs as soon as possible. This includes accepting that an SBOM can have some initial errors and omissions, but instead of waiting for perfection, SBOM practices should start today.

There are two main formats for SBOMs that are widely used and accepted. SPDX, which is maintained and supported by the Linux Foundation, and CycloneDX, maintained and supported by OWASP.

Two Main Formats: SPDX and CycloneDX

There are two main formats for SBOMs that are widely used and accepted. SPDX, which is maintained and supported by the Linux Foundation, and CycloneDX, maintained and supported by OWASP.

Let us briefly look at the SPDX and CycloneDX files to get a feeling for the information they can contain. Both formats have support for much more data than given here, and we refer to the respective specifications for details. The information provided here is based on SPDX v2.3 and CycloneDX v1.4.

Inside the SPDX SBOM File

An SPDX SBOM consists of a set of sections. The first part, which is mandatory, is the meta-information about the SPDX file. This is called the Document Creation Information. This includes, e.g., when the SBOM was created, which tool was used to create it, which SPDX version it is based on, and other SPDX documents that are referred to in this document.

PACKAGE INFORMATION

Then there are sections for each of the packages. Each package includes basic information on its name, version, and download location. There is also a unique identifier to be used within the SPDX document to reference other information.

The package section also includes license information, and if different files within the package have different licenses, then the complete list of all found licenses within the package can be listed. The package section in SPDX also has support for free text comments on licenses, copyright text, and other types of free text comments on the package in general.

SECURITY INFORMATION IN EXTERNAL REFERENCES

An important field is the one for *external references*. This field can be used to refer to an external source for more information about the package.

One defined category for external information is security, which can be used to link to advisories, fixes, or URLs with security-related information. The advisory can include links to CVEs, the vendor's vulnerability disclosure document, or even security information formatted in a CycloneDX SBOM file.

FILES AND SNIPPETS

Following information about a package, it is also possible to add information about specific files inside a package. Such information is given in a separate section after the corresponding package section. Further details can be given in yet another section referring to specific snippets inside a file. These snippets can be referenced by byte ranges or line numbers and can have licenses that are different from the rest of the file or from the package.

DESCRIBING THE DEPENDENCY GRAPH

In the package, file, and snippet sections, the data given in each element is independent of the others. The relationship between a package and its files is implicit in that the files section follows the corresponding package section. But there can also be relationships between files and, maybe more importantly, relationships between packages. One package typically depends on another package, and there are transitive dependencies such that one package will depend on a package that, in turn, depends on a third package, etc.

These relations between components are described in their own section. The relationship can be one of many but "depends on" and "dependency of" are useful for describing the dependency graph for the software.

The relation can also be marked to indicate that a part of the graph might be incomplete or that the creator assures that it is complete.

Inside the CycloneDX SBOM File

Similar to SPDX, CycloneDX starts with identification information and metadata. This specifies that it is a CycloneDX SBOM, which specification version it conforms to, and the SBOM version for that particular software. Then there is, e.g., a timestamp and an identifier for the tool used to generate the SBOM (or the author if it was manually generated).

COMPONENTS

Following the metadata, the components are described. The component type is defined as, e.g., file, container, library, or application. Some notable component information includes the component's type, name, and version.

Similar to SPDX, CycloneDX starts with identification information and metadata. This specifies that it is a CycloneDX SBOM, which specification version it conforms to, and the SBOM version for that particular software.

To make it uniquely identifiable, it can also include one or several of the CPE, PURL or SWID identifiers. This will allow the SBOM file to be used to identify and monitor new vulnerabilities in the software. The component information will also include license information. It will hold the license ID but can also include the license text itself or a URL pointing to the license file. Each component can also include a bom-ref identifier which can be used to reference the component in other parts of the SBOM.

SERVICES

Separate from components, it is also possible to list services, e.g., microservices. The SBOM can then be used to define if using a service crosses a trust boundary if it requires authentication and specific API endpoints for a service.

EXTERNAL COMPONENTS

CycloneDX has also support for adding external references. These can be either declared as part of a specific component or be defined outside the components part of the SBOM. External references are added in the form of URLs to the information.

DESCRIBING THE DEPENDENCY GRAPH

The relationship between dependencies is documented in a separate part. It is here possible to refer to a component using the bom-ref attribute and to declare which other components it directly depends on. Doing this for all components will provide a dependency graph of the software that represents both direct and transitive relationships between dependencies.

COMPOSITIONS AND ASSEMBLIES

CycloneDX has also support for describing compositions, which is a collection of components, services, and dependency relationships. A composition can describe an assembly which can be seen as a well-defined part of the software or application that, in turn, can include other parts in a nested fashion. The composition can also be described with dependencies, which are parts of the software that requires other independent parts.

VULNERABILITIES

Vulnerabilities are described explicitly in a separate part of the CycloneDX SBOM. A vulnerability description refers to the bom-ref of the affected component and can include several pieces of information. This includes the vulnerability ID, the publisher, references, the CWE identifier, CVSS information, a description of the vulnerability, advisory information, timestamps, etc.

It is also possible to include analysis details for the vulnerability, e.g., describing it as resolved, exploitable, in triage, or not affecting the component or service, including a justification for this assessment.

Vulnerabilities are described explicitly in a separate part of the CycloneDX SBOM. A vulnerability description refers to the bom-ref of the affected component and can include several pieces of information.

SIGNING DATA

Finally, the complete SBOM can also be signed using a JSON-formatted digital signature, including the public verification key and a certificate path. In addition to signing the SBOM, individual parts, such as components, services, and compositions, can also be individually signed.

COMPARING SPDX AND CYCLONEDX

SPDX and CycloneDX share the support for the main use cases in that both licensing information and vulnerability information is supported. However, they differ in the extent of the support. Looking at the specifications, it is clear that SPDX leans more heavily towards the licensing use case, while CycloneDX has more support for vulnerability information.

LICENSE INFORMATION SUPPORT

As an example for license information, SPDX adds a specific field for "concluded license," which can be used if the license can not be determined or if there has been no attempt to find it. It also has a field for collecting all licenses in the files of a package and adding comments to the licenses.

The snippet information section also has its own fields for license information. Such a level of granularity, down to specifying snippets of files, is not supported by the CycloneDX specification. As part of the SPDX specification, there is also the SPDX license list. This list provides a standardized short identifier for all commonly found licenses. This identifier is becoming an industry standard for identifying licenses and is also used by CycloneDX SBOMs.

SECURITY AND VULNERABILITY INFORMATION SUPPORT

Looking at security, CycloneDX defines a large number of fields related to vulnerabilities, their metadata, assessment, and the actions taken for them. This data is not explicitly supported by SPDX, though it is possible to use external references to include some security data.

Another security-related difference is the support for digital signatures in the CycloneDX SBOM. Both the SBOM and parts of the data inside it can be digitally signed to provide data authentication and non-repudiation for the data. It is, of course, also possible to digitally sign an SPDX document. Still, it has no support for enveloped signatures, as is the case for CycloneDX, i.e., the signature is part of the signed document.

Looking at security, CycloneDX defines a large number of fields related to vulnerabilities, their metadata, assessment, and the actions taken for them. This data is not explicitly supported by SPDX, though it is possible to use external references to include some security data.

Encoding of Data

Both SPDX and CycloneDX support JSON formatted data, while SPDX additionally supports YAML, RDF, a tag: value text file, and XLS spreadsheets. CycloneDX has XML support, while SPDX is looking to add this support in the next release.

Software Identification (SWID) Tags

As noted above, NTIA also includes the possibility of using Software Identification (SWID) Tags as an SBOM format. A SWID tag can include the information needed for transparency in the open source software supply chain, but its main use case is somewhat different. A SWID tag is designed for tracking installed software throughout the lifecycle. Here, throughout the lifecycle is supported by defining different types of tags for pre-installed and installed software, as well as patch tags, to define patches to software and supplemental tags for additional information.

The XML-formatted SWID tag will include information about the software, its license, and the files needed to install the software. It can also include information on what other packages are needed as a prerequisite for installation. This will allow for the automated installation of software and for monitoring what software is installed in a system, which version it has, and which patches have been installed.

SWID tags are designed to be removed once the installed software is uninstalled and removed from the system. This shows the close relationship that the SWID tags have with the installed software.

Four Variants of SWID Tags

The *corpus tag* is used pre-installation and is used by the software installer. They can authenticate the issuer and be used to verify the integrity of the software. License information can be used to make sure that no license is violated before the software is installed.

The *primary tag* is used to describe software that has been installed. It has a globally unique tag ID to make it possible to track that particular installation. It can also link to other SWID tags. Such a link can be defined as a component if other software is a component of the software. It can also be defined with a required attribute if it depends on another software component. A simple example is a productivity suite that has a word processor and a spreadsheet processor as components. Both these will, in turn, have some common libraries and functionalities as required.

The *patch tag* describes a patch rather than the software product itself. It includes information about which product the patch is for, if other patches need to be applied before this patch, or if it replaces another patch.

The *supplemental tag* can be used by the local system to provide additional information. This could be, e.g., the time of installation.

Tags Are Tied to Installed Software

SWID tags are designed to be removed once the installed software is uninstalled and removed from the system. This shows the close relationship that the SWID tags have with the installed software. Comparing this to SPDX and CycloneDX, these two SBOM formats are more descriptive of the software and its composition and not tied to the particular installation of the software.

For more details, NIST provides an excellent guideline for SWID tags.

Concluding

Having well-defined formats for storing, communicating, and encoding SBOM information is vital for its adoption. Both CycloneDX and SPDX have been widely adopted, and it seems that the current trend is that CycloneDX is getting the most attention. This can be attributed to the fact that the recent drivers, e.g., the Biden executive order and the EU cyber resilience act, are heavily focused on the security benefits for SBOMs.

In the final section, we will show how **Debricked** supports both exporting and importing of SBOMs to help you stay on top of security and license compliance.

At Debricked, we favor and currently support the CycloneDX format for SBOMs. This is not to say that there are no use cases that are a better fit for the SPDX format. Still, we believe that the license support in CycloneDX is sufficient, and the additional vulnerability fields it provides are very useful.

The Software Bill of Materials: SBOM with Debricked

With Debricked, it is easy to both generate and analyze an SBOM, and there are several ways of doing both. In this post, we look at some of the possibilities to create and scan SBOM files with Debricked.

At Debricked, we favor and currently support the CycloneDX format for SBOMs. This is not to say that there are no use cases that are a better fit for the SPDX format. Still, we believe that the license support in CycloneDX is sufficient, and the additional vulnerability fields it provides are very useful.

Generating an SBOM

Generating or exporting an SBOM is available for our enterprise-tier customers. If you have integrated your repositories with Debricked, an SBOM can be generated as a report. You can choose to generate the SBOM for a single repository or a chosen set of repositories, or you can generate a global report for all your integrated repositories.

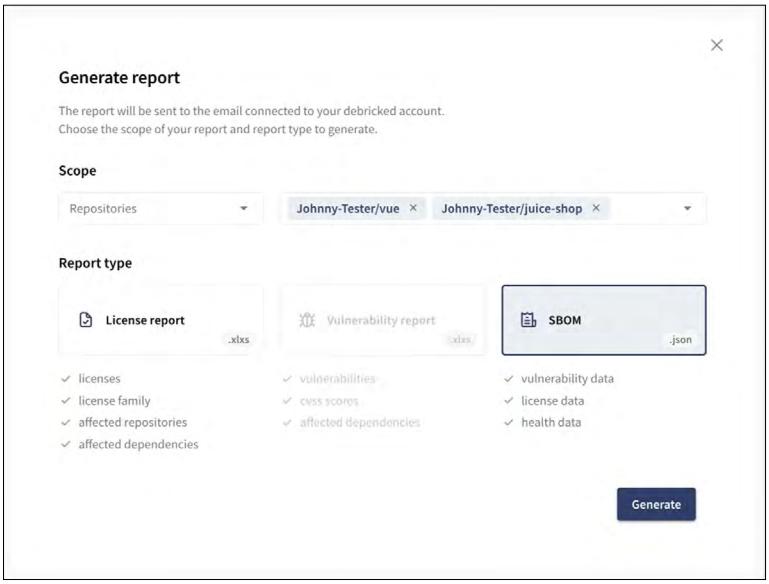

Figure 1. Generating reports in the Debricked interface

The SBOM will be generated as a JSON file and emailed to the email address associated with your account.

Some of the things that will be found in the SBOM generated by Debricked are:

- All dependencies, including transitive dependencies, together with their CPE and/or PURL identifier.
- The identified license for the dependencies. Both the SPDX license short name and the actual license text is provided. As external references, we also point to the URLs of the actual license information. This reference is denoted “Proof of License” and enables anyone to find the license file easily.
- The vulnerability data for each dependency. This data includes the vulnerability identifier (CVE, GHSA, etc.), the source, the CWE, a description of the vulnerability, references to more information, the CVSSv2 and CVSSv3 scores, and dates when it was published and last updated.
- Relations between dependencies. All dependencies are listed for each library, providing the complete dependency graph for all open-source components. If a library has no dependencies, this is indicated with an empty list.

If you prefer to use our API, the SBOM can be generated using the corresponding endpoint. There are a few API endpoints to choose from, and we refer to the API documentation for a complete overview.

Using the API

If you prefer to use our API, the SBOM can be generated using the corresponding endpoint. There are a few API endpoints to choose from, and we refer to the API documentation for a complete overview. One of them is to simply generate an SBOM based on a selected set of repositories, as shown below.

Figure 2. Excerpt from the Debricked API documentation

Here you can choose if you want to include vulnerability and/or license data as well. Using the API will require an access token. A refresh token can be generated in your Debricked account, which can be used to generate a JWT token. Or you can just use your login ID and password to generate a JWT token immediately.

Uploading and Analyzing an SBOM

If you have an SBOM and want it analyzed, Debricked can do it for you. We even monitor the dependencies for new vulnerabilities, and we can alert you if any are found.

If you have an SBOM and want it analyzed, Debricked can do it for you. We even monitor the dependencies for new vulnerabilities, and we can alert you if any are found.

Manual Upload

The easiest way to analyze an existing CycloneDX SBOM is to upload it in the Debricked GUI. Just go to *Repository settings*, and click the *Manual scan* button.

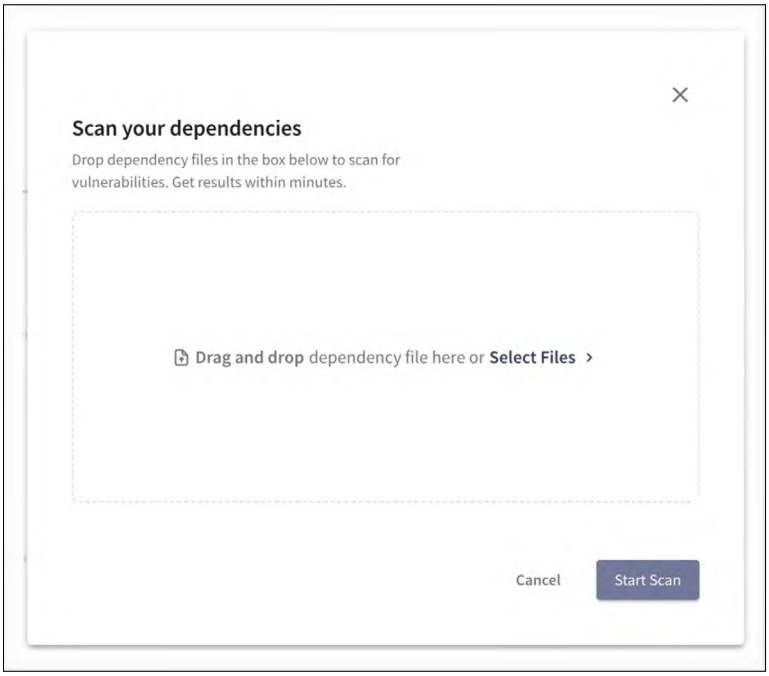

Figure 3. You can manually drag and drop an SBOM to have it scanned

Here you can select the SBOM file or just drag and drop it. The SBOM will show up as a new repository, listing all vulnerabilities, licenses, and dependencies. If there is a new vulnerability, it will also show up in the user interface.

Adding SBOM to a Repository

The manual scan option will show new vulnerabilities in the UI. If you want to be alerted, e.g., with an email, every time there is a new vulnerability, then you can simply add the SBOM to be scanned as part of the CI/CD. When scanning the repository, Debricked will find the SBOM file and scan it for new vulnerabilities.

Upon a scan, you can set up an automation rule to trigger existing or new vulnerabilities. You can tailor the automation rule to, e.g., trigger an alert if the vulnerability is of high or critical severity. Below, we show an example that will send an email to all Debricked account administrators upon a scan if there is a new vulnerability or a vulnerability with at least high severity.

Upon a scan, you can set up an automation rule to trigger existing or new vulnerabilities. You can tailor the automation rule to, e.g., trigger an alert if the vulnerability is of high or critical severity.

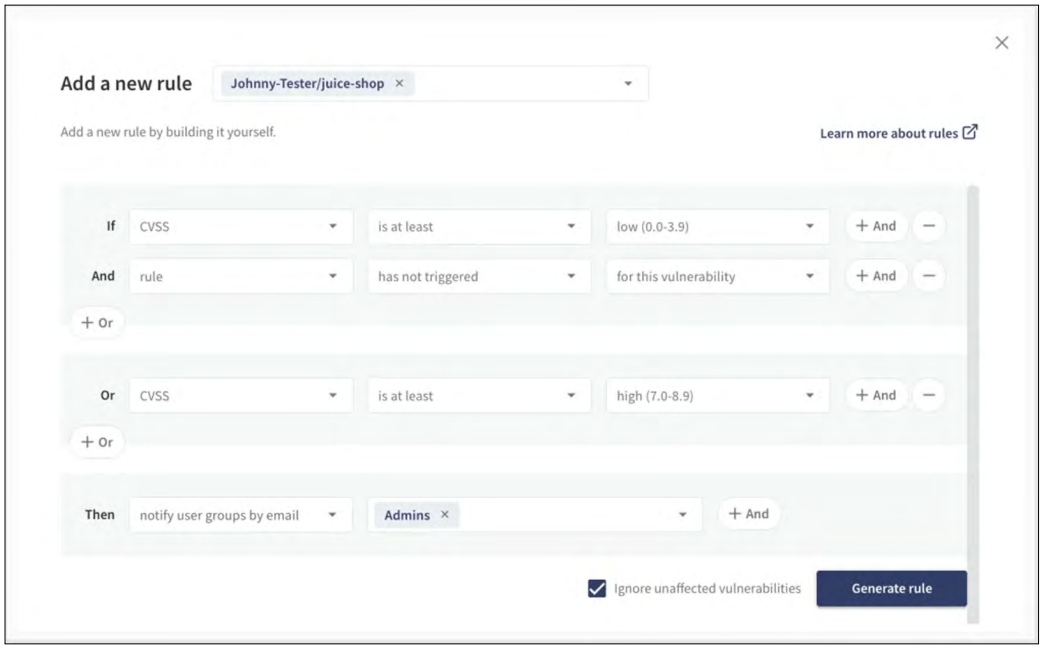

Figure 4. Adding a new automation rule

This will allow the administrators to be reminded of high-severity vulnerabilities on every scan but only to be alerted to lower-severity vulnerabilities once. Also, vulnerabilities that have been triaged not to affect the organization or software will not cause any alerts. This is ensured by checking the box “Ignore unaffected vulnerabilities.”

Let us look at an example of how you can use GitHub for monitoring and alerting on identifying new vulnerabilities. To trigger a scan, you use a scheduled GitHub actions workflow. Workflows are added to the .github/workflows subfolder. For Debricked, the workflow can look like this.

name: **Debricked** scan
on:
schedule:
- cron: “0 9 * * *”
jobs:
vulnerabilities-scan:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@**v2**
- uses: debricked/actions/scan@**v1**
env:
DEBRICKED_TOKEN: **$**{{ **secrets.DEBRICKED_TOKEN** }}

This will run a new scan of the SBOM every day at 9 am and trigger alerts according to the automations rule above.

It is, of course, possible to do similar scheduled scans if you are using other CI/CD tools.

Conclusion

Since Debricked supports scanning and monitoring SBOMs, the SCA tool is not only for software producers and developers. It is also a powerful tool for purchasers and consumers. Debricked will handle the automation and interoperability parts, monitor new vulnerabilities and license changes, and alert you on any significant changes.

Once the requirements to supply an SBOM together with software products are met, all stakeholders throughout the value chain will be able to better understand the products’ security. This will lead to more secure products, better responses to new vulnerabilities, and transparency in the software supply chain.

Register for Debricked for free and take full control of security, compliance and health with a toolkit that will revolutionize the way you use open source.

Register for Debricked for free and take full control of security, compliance and health with a toolkit that will revolutionize the way you use open source.

Connect with Us

www.opentext.com

opentext™ | Cybersecurity

OpenText Cybersecurity provides comprehensive security solutions for companies and partners of all sizes. From prevention, detection and response to recovery, investigation and compliance, our unified end-to-end platform helps customers build cyber resilience via a holistic security portfolio. Powered by actionable insights from our real-time and contextual threat intelligence, OpenText Cybersecurity customers benefit from high efficacy products, a compliant experience and simplified security to help manage business risk.

762-000084-004 | O | 01/24 | © 2024 Open Text