

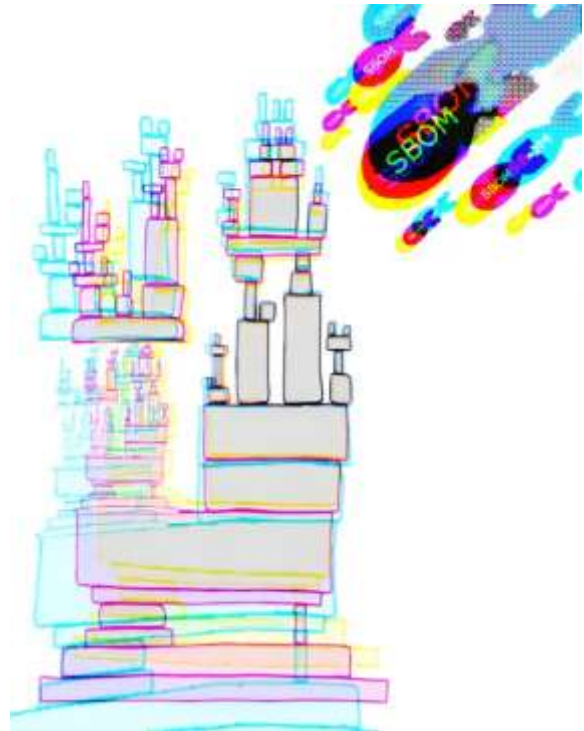
# Come Friendly SBOMs

## Leveraging software bill of materials for your benefit and protection

**Alice Sowerby**

I was going to give this piece the title *How I learned to stop worrying and love the SBOM* but I discovered that this American witticism has already been used extensively for the topic. Instead, I decided to pivot to an altogether more British alternative based on the much-loved poem, ***Slough***, by Sir John Betjeman which has the first line "Come friendly bombs and fall on Slough!". It was published in 1937 to lament the industrialisation of a British town near London after World War I.

The poem invites bombs to destroy the town as it had, in Betjeman's view, become ruined and beyond rescue. I hope to draw a more hopeful parallel by inviting friendly SBOMs (Software Bill of Materials) to drop on code. Not to destroy it – there is much to love and save about the great legacy of code we



work with (and starting again may be impossible) – but to support it, to improve it, and to make it safer for the world to use.

We'll start by looking at the context that SBOMs fit into – both regulatory and cultural. I'll discuss how we got here, why security matters more and more, how SBOMs help, and then I'll share a deep dive into what FreeBSD is doing about it.

The entry point to our story is a software security regulation that has been brought in by the European Union. It's called the **Cyber Resilience Act (CRA)**. Although it may not sound very exciting, its implications are significant for software vendors and will reshape dynamics across the software industry, including open source.

The CRA specifies that anyone placing "products with digital elements" on the EU market is responsible for the security of those products. In practice, this means manufacturers must design products securely, maintain evidence that they have done so, and provide that evidence to European market surveillance authorities on request. They also have to provide the product to the consumer in the most secure state possible and, if there is any actively exploited vulnerability or security incident relating to the product, they have to meet stringent reporting and remediation timelines.

Compliance with the reporting requirements is mandatory from 11 September 2026, and compliance with the "secure by design" and "secure by default" requirements is mandatory from 11 December 2027.

The fines for non-compliance are pretty eye-watering. The highest tier of fines, which applies to non-compliance with core requirements, is €15 million or 2.5% of total worldwide annual turnover, whichever is higher.

Let's look at why this is important now, and why this is of any interest to the humble open source developer.

The good news is that open source projects finally have regulatory backing that requires downstream manufacturers to build and maintain software responsibly. The regulation mandates the use of SBOMs and requires

manufacturers to perform due diligence on the security of every component in their codebase, whether developed in-house or sourced from a third party.

The **European Union Agency for Cybersecurity (ENISA)** is also developing guidance on what constitutes "secure by design" and "secure by default", while market surveillance authorities are responsible for assessing manufacturers' compliance.

If you are getting a sinking feeling about how this will shake down onto us, the little guys, there is reason for hope. Thanks to some fantastic teamwork in the open source community over the last few years, the **European Commission (EC)** has become much more aware and educated about the nature of open source software development and its role in the software industry. Various individuals and organisations from the open source community have been engaging with the EC as the CRA has been developed. The CRA now includes specific exemptions and protections for open source software. There are even some opportunities to develop a source of income for open source projects. Specifically, there is a proposal for a "voluntary security attestations" framework. If implemented as currently envisaged (the details are still being developed), it could create a framework that enables open source projects to charge manufacturers for security attestations.

Open source projects and their developers are not subject to the CRA, and though open source foundations may be subject to some reporting obligations, they are not liable for fines for non-compliance.

The regulatory burden and associated penalties – the big sticks – are aimed primarily at commercial manufacturers.

Perhaps the most important outcome of all this work has been that the EC now has institutional knowledge of the value of open source and has become invested in safeguarding its role in European technological innovation and sovereignty. As the EC continues to refine its guidance on the CRA, and develop its initiatives, such as the voluntary security attestations, we can be sure that they are considering open source as a first-class stakeholder.

Note: You may recall that the USA had released a similar directive: [US Executive Order 14028 of May 12, 2021, Improving the Nation's Cybersecurity \(2021\)](#). This was later partially withdrawn on January 29, 2026, making specifications optional.

In some ways, it's surprising that it's taken so long for the security of software to become regulated in this way. To understand why, let's take a brief trip down memory lane (may I suggest with a hot drink at hand?).

In the mid twentieth century, around the time when my grandfather was making computers with English Electric Computers, it was a bit of a niche field. Computers were expensive, scarce, and usually operated by trusted institutions such as governments, universities, and large corporations. Security was achieved by physical isolation and restricted access by known users. Reliability, performance, and making these systems work at all were much more pressing concerns than defending against malicious actors.

As personal computing became more common in the late twentieth century, when my father started as a software engineer, the push was to build networks and develop open and interoperable protocols rather than secure ones, often under the assumption that participants could be trusted. Connectivity and adoption were seen as the primary challenges. At the same time, commercial software development also incentivised delivering new features, which customers could easily evaluate and would pay for, but the cost of poor security was difficult to measure until something went wrong.

In the internet era, when I started working at software security firm, Sophos, ordinary people were starting to go online. This led to a much greater awareness of "viruses" in the general public as the first famous incidents of malware attacks hit the headlines. Security was often treated as a separate discipline, addressed through layers of protection such as anti-virus software, firewalls, security appliances, and penetration testing, rather than being designed into software from the outset.

The 2010s brought cloud and mobile computing, and the continuous delivery approach enabled by DevOps. The existing bolt-on approach to security was poorly suited to an era of continuous deployment, where software could be

updated many times a day and deployed instantly into connected environments. Securing through periodic pen testing was no longer suitable and DevSecOps was the response to this challenge, it was the logical next step for the shift-left principle that had created DevOps.

Today, software security extends beyond code defects to encompass supply chains, open-source dependencies, cloud infrastructure, identity management, and AI-enabled systems. High-profile incidents have demonstrated that vulnerabilities can arise anywhere in a complex ecosystem.

We can see that, over time, software security practices have evolved in response to changes in the way software is built and used. However, it has often been a trailing concern because the incentives driving software development – functionality, usability, connectivity, and speed of delivery – are often prioritised above addressing the associated security risks. And thanks to the creativity and opposing incentives of malicious actors, many lessons have had to be learned the hard way.

In recent years, regulating software security has been a growing focus for governments across the world. The security of software that is being used by governments, or as critical industry infrastructure, or in consumer products has become key to national security as nation state actors and other malicious groups have noticed and sought to exploit the opportunities of software being everywhere. The safety of the state and the individuals represented by the state both rely on the security of the software being used in every layer of society.

I mentioned before that the CRA requires manufacturers to use "secure by design" practices. Further to this, they are specifically directed to adopt the use of something called a Software Bill of Materials (SBOM).

An SBOM is an inventory of every third-party component contained within a code base. The inventory contains meta data including the name of the package, the licensing, and the version number. This is provided in a machine-readable file which supports tooling that can cross check for

reported exploitable vulnerabilities, unpatched packages and license conflicts.

SBOMS have been around for a little while and are now starting to become well-standardized and well-adopted, and tooling around them is maturing.

SBOMs are becoming a backbone for many meta activities relating to software code bases and software supply chain management. Knowing key meta data about all of the components within your software allows you to do things like compare the version numbers of the packages you have to the most recent packages known, to check for unpatched dependencies in your code base. You can also check your SBOMs against a list of known vulnerabilities so that you can quickly determine whether your product will be affected by them.

SBOMs were also originally intended to support license management, that is, understanding whether you are permitted to use open source components in the way you intend. They are still useful for this even if you primarily want your SBOMs for security reasons.

As you might imagine, SBOMs can include elements of recursion, allowing you to see multiple layers deep into your supply chain. This is crucial for understanding and managing the risk level for your product.

Becoming educated and skilled in the use of SBOMs is not only a marketable asset for any developer. It is also a great way to increase security awareness and develop more secure development practices.

For open source projects, providing an SBOM is not essential, as it is possible for downstream manufacturers to use third-party tools to scan the code and provide an SBOM that way.

However, this method is somewhat suboptimal. There are many ways that an SBOM can be inaccurate when created this way, not least because it has no insight into the build environment.

Therefore, although it is not essential for an open source project to provide an SBOM, or SBOM tooling, as part of its project, it is actually very beneficial for all parties if one is provided. For downstream users it means that an accurate SBOM can be created on demand and, for projects themselves, committing to an SBOM mindset and practice will improve the overall security of the project. When open source projects create their own SBOM they are then able to proactively undertake housekeeping such as patching dependencies and fixing exploitable vulnerabilities.

## An SBOM of One's Own

It turns out that we can use literary references all through this article. The allusion this time is to [\*\*\*A Room of One's Own\*\*\*](#) – an extended essay by British author, Virginia Woolf. In it, she declares that "a woman must have money and a room of her own if she is to write fiction", referencing the role of financial independence in creating the conditions in which art can be produced.

We might note that an open source project too needs resources to create an SBOM of its own – this is the kind of work that benefits everyone but that it is hard to find commercial sponsors for, or volunteers to tackle.

Thankfully, the FreeBSD Foundation has been able to access several sources of investment to support some SBOM-related projects for FreeBSD over the last few years.

## Getting to Know FreeBSD's Dependencies

In 2025-26 the Foundation ran a project named [\*\*\*Beach Cleaning\*\*\*](#), sponsored by the Alpha Omega project (part of OpenSSF). It had the goal of improving our understanding of FreeBSD's open source dependencies and their associated risk. The components in FreeBSD's base system were manually analysed for risk, and an action plan was created to work with upstream projects to mitigate these.

From this data, and other component metadata, we built a database to organize the information about their maintainers and dependencies. It also includes annotations about security reviews and plans to reduce risks where found. The data needed to create an SBOM matched what was included in the database.

We have created tooling that can produce reports on demand about the components in the base system. Each report type focuses on a specific area of information across the population of components, allowing insight into the whole picture of FreeBSD's dependencies, their risk analyses and the current state of the plan to mitigate any issues. It can also generate a CODEOWNERS file to the standard required by GitHub. This tooling is now available for use on an ongoing basis to allow maintainers to continue to manage risk in FreeBSD dependencies more easily.

## **SBOM Fundamentals**

Another project managed by the Foundation was an investment from the German government's [\*\*Sovereign Tech Agency \(STA\)\*\*](#) which commissioned the development of SBOM fundamentals for FreeBSD. The STA is concerned with "improving and maintaining foundational digital technologies that enable the creation of software" as a route to national digital sovereignty, ensuring the viability of open source alternatives to commercial products from international providers.

This project, which was specified by key FreeBSD stakeholders, set out to explore the options for how SBOMs could be generated for FreeBSD, propose a solution, and implement as much as possible in the time available.

Note: The project was STA-commissioned to run from April-December 2025. The Foundation has continued to fund an extension through to the end of October 2026.

The requirements for the SBOM solution were:

- The SBOM must be in a format that is widely used and sustainable.
- The SBOM must be based on source code and not compiled binary.

- It must be possible to generate the SBOM on demand for the image being built.
- An SBOM must be provided for official released binary images of FreeBSD.
- The solution must be well-documented for the community to use.

Starting in April 2025, the project first evaluated the available SBOM standards. The two most popular open source options are **CycloneDX** and **SPDX**, and the latter was chosen based on its popularity, status as an ISO standard, and tooling ecosystem. SPDX is also more nuanced in its handling of licence information, something that is beneficial for open source projects like FreeBSD where, despite a project preference for the 2-Clause BSD license, the source tree contains code under many different licenses. To increase utility and compatibility of the SBOM solution, we also looked for ways to include the ability to support the CycloneDX format in future as a secondary output so that end users could have the choice of which to use.

Work then began on establishing a workable solution for creating an SBOM for the FreeBSD base system.

Let's take a look at what goes into an SBOM. An SBOM is a data file which contains key metadata about each package in the code base.

The SBOM contains a list of package summaries that include:

- **Name:** Provides a human-readable name for the package.
- **Version:** The version of the package.
- **Requires, Conflicts:** These specify dependencies and are used to produce a dependency tree in FreeBSD SBOMs. (*Conflicts* is currently not used in the FreeBSD SBOM solution.)
- **Copyright:** The copyright information for the application.
- **License:** The application's license(s), expressed with the corresponding SPDX short identifier (e.g., BSD-4-Clause AND BSD-3-Clause). Licenses are gathered using scancode-toolkit and SPDX-License-Identifier tags if available in source or header files.
- **Source:** The location where the source code was obtained.

- **URL:** The application's homepage. If no alternative exists, man pages are used, especially for non-third-party applications.

Having this data enables some powerful insights and capabilities.

*Name* and *Version* can be used to determine whether a known vulnerability is in your codebase by cross-checking against public **Common Vulnerabilities and Exposures (CVE)** databases. This can be optionally extended into dependencies by using the *Requires*, *Conflicts* information .

The *License* field means you can check that you are using each package within the terms of the licence. If you are trying to avoid components with a more restrictive licence this is critical information. The *Copyright* field supports the requirement that many licences have, which is that copyright notices are preserved.

*URL* and *Source* help with tracing exactly where the package was obtained from. Where available, we refer to the PURL (Package URL) for each package as this is a format which is becoming adopted to provide predictable URL structures for package identifiers.

All this information helps anyone building on FreeBSD to know more about what it contains and to be able to manage risk, compliance, and security more thoroughly and efficiently.

The next step was to choose the right way to build the SBOM for FreeBSD. While there are excellent tools for inspecting binaries and for returning an SBOM, the gold standard is to create an SBOM at build time. This provides the most accurate SBOM possible, so this was our goal. Therefore, we had to look for suitable tools that could help us to achieve that.

As the FreeBSD Project is mostly written in C, there are not as many tools available to create an SBOM toolchain as there are for other languages. However, we learned about new capabilities of pkgconf, which is a tool that can read and interpret pkg-config files (.pc files). It is related to pkg-config but it supports additional tags in the .pc files for the purpose of creating an SBOM. Before we could use pkgconf for this task there were a few hurdles to

overcome. First, pkgconf was not part of the base system. Even though it is available in the FreeBSD ports collection, it had to also be in the base system if we wanted to use it there at build time. Secondly, though pkgconf can output SPDX 2.0 via the bomtool facility, we wanted to be able to use the newer SPDX Lite 3.0 standard. And thirdly, no component in the base system had a suitable .pc file. (A few 3rd party contrib software packages include a .pc file, but they were not integrated into the build system.)

So our task list was:

1. Add pkgconf to the base system,
2. Add functionality, named spdxtool, to pkgconf that will output SPDX Lite 3.0.1 SBOMs in RDF JSON-LD format (and upstream the change),
3. Add any missing .pc files in the base system (and upstream these to any of our third-party components that did not already have them) and,
4. Integrate SBOM generation into the base system build process.

At the time of writing, number one is done, following the standard vendor import process. Number two is complete upstream – the new spdxtool utility was just released in pkgconf 2.9.90. Number three and four are in review – we used scancode-toolkit to review the code base automatically and pull out the licence and copyright information. We then reviewed the extracted data and manually corrected or determined information for packages that were corner cases. The data then needed to be stripped of unnecessary fields and placed in the .pc files, which can be found in share/sbom/pkgconfig. Lua tooling was created to achieve this.

Current SBOM information is available for software in the lib, bin, sbin, usr/bin, and usr/sbin components.

Once SBOM generation is fully integrated in the base system it will be possible to create an SBOM on demand during a regular build, like with the command `make buildworld`. The produced files are expected to be installed in `/usr/share/sbom/spdx/` and have a `.spdx` extension for version 2.2, and also in `/usr/share/sbom/jsonld` and have a `.jsonld` extension for version 3.0.1 Lite.

# Modernising FreeBSD's Vulnerability Data Format

Another project has been to move FreeBSD's ports and packages vulnerability data from an in-house format to a standardised industry format. This project was also made possible by another commission from the STA and ran in parallel with the SBOM project.

Like many open source projects, FreeBSD has customarily kept its vulnerability data in a machine readable format – in this case, **VuXML**. VuXML was introduced in 2005 as a document format for describing security issues that affect a software system. VuXML never achieved cross-ecosystem adoption outside of FreeBSD and derivatives.

As the security landscape evolves, this situation becomes unsustainable. In 2021 the **Open Source Vulnerability (OSV)** project was created. It was developed specifically to meet the needs of the open source software community and uses the OpenSSF OSV format to provide a lightweight and high-fidelity JSON output that uses git hashes or package manager versions to describe vulnerabilities.

The OSV project was created to solve the challenges that open source consumers faced when trying to map CVE entries to the package versions they were using. As a solution, it consists of a distributed database along with a common schema. Without a common standard, the work could be time-consuming and inaccurate – the opposite of what is needed when trying to find out if you have an exploitable vulnerability in your codebase.

Adopting an industry-recognized format offers significant benefits as it simplifies how external parties can consume and utilize FreeBSD vulnerability data. It also allows us to manage data with a broader range of existing upstream tools, reducing the need for custom development.

Providing vulnerability data in a standard format increases the security of the FreeBSD ecosystem by facilitating seamless consumption by a wider array of security tools and services, which accelerates the detection and mitigation of threats for all users of FreeBSD and its derivatives.

In consultation with FreeBSD's Security Team and the wider FreeBSD community, we considered the benefits of adopting the OSV format. Other standards were reviewed but we did not pursue them due to their limitations in meeting our expected needs

This project focused mainly on setting up the structures to handle data in a new format. The actual migrating of existing data to the new format is something that needs to be done in an ongoing manner through adopting this format. That task is in the hands of the ports security team.

## All the World's a SBOM

The phrase "All the world's a stage" is from Shakespeare's play, [\*As You Like It\*](#). It is found at the start of a monologue that describes the seven ages of man. Perhaps we could imagine that the SBOM is in its second age, "Then the whining schoolboy, with his satchel / And shining morning face, creeping like snail / Unwillingly to school." It is clear that the laborious work of creating an SBOM tooling ecosystem is still very much ongoing, with the hope of mastery lying ahead.

It is a good moment to look around at the SBOM tooling landscape. Let's see what's available to help put together the SBOM-based processes that you are looking for.

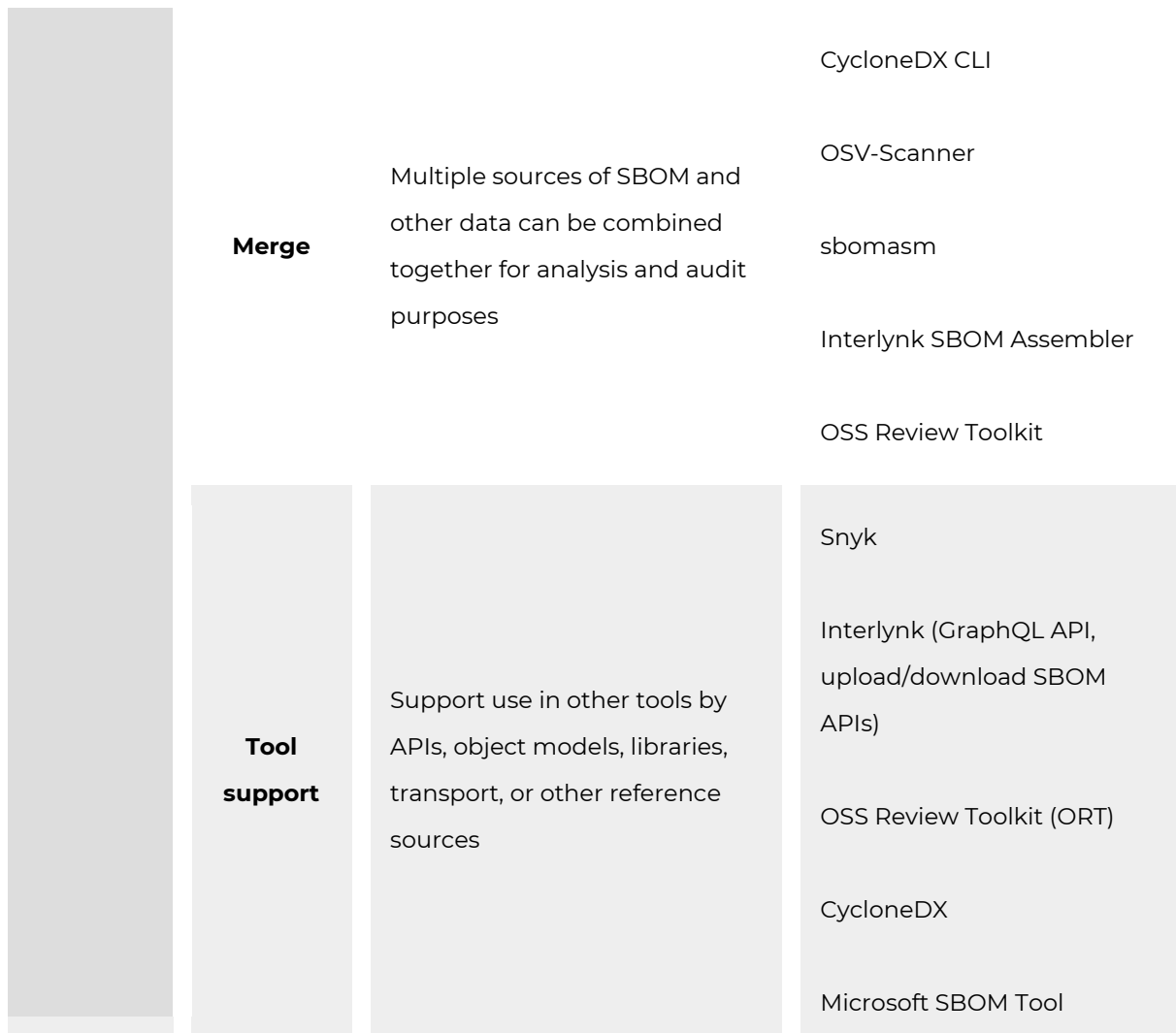
To understand the way that tooling is used to work with SBOMs, it's useful to refer to the [\*SBOM Tool Classification Taxonomy\*](#) developed by the National Telecommunications and Information Administration base in the USA.

The taxonomy classifies SBOM-related tooling according to how it is used. I have added some example tools that I found in a curated list available at the well-regarded [\*awesome-sbom\*](#) repo.

| Category | Type    | Description                                                                                                          | Tools                                                                                                  |
|----------|---------|----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Produce  | Build   | SBOM is automatically created as part of building a software artifact and contains information about the build.      | Syft<br><br>Microsoft SBOM Tool<br><br>CycloneDX Maven Plugin<br><br>spdx-sbom-generator<br><br>cdxgen |
|          | Analyse | Analysis of source or binary files will generate the SBOM by inspection of the artifacts and any associated sources. | Syft<br><br>Trivy<br><br>Tern<br><br>cdxgen                                                            |
|          | Edit    | A tool to assist a person manually entering or editing SBOM data.                                                    | SBOM-Manager                                                                                           |
|          |         |                                                                                                                      |                                                                                                        |

|                |               |                                                                                                                                                                                 |                                                                                                                                                          |
|----------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Consume</b> | <b>View</b>   | <p>Be able to understand the contents in human readable form (e.g., picture, figures, tables, text, etc.).</p> <p>Used to support decision making &amp; business processes.</p> | <p>SBOM-Manager</p> <p>SBOM Viewer</p> <p>OSS Review Toolkit</p> <p>Interlynk Platform — supports ingesting, managing, querying and exporting SBOMs.</p> |
|                | <b>Diff</b>   | Be able to compare multiple SBOMs and clearly see the differences (e.g., comparing two versions of a piece of software).                                                        | <p>Alsbom</p> <p>CycloneDX CLI</p> <p>SBOMDiff</p>                                                                                                       |
|                | <b>Import</b> | Be able to discover, retrieve, and import an SBOM into your system for further processing and analysis.                                                                         | SBOM-Manager                                                                                                                                             |

|                  |                  |                                                                                       |                                                                                 |
|------------------|------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <b>Transform</b> | <b>Translate</b> | Change from one file type to another file type while preserving the same information. | <p>CycloneDX CLI</p> <p>SBOM2doc</p> <p>SBOM2dot</p> <p>Interlynk SBOM Move</p> |
|------------------|------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|



That's where I'm going to leave it for this article. I hope you found it interesting and useful to learn a bit about the CRA, SBOMs and how to leverage them for your own benefit and protection.

You can follow along with the FreeBSD Foundation's work on developing the SBOM toolchain for the FreeBSD Project at the [Foundation's GitHub repo](#), where there is a project folder for [CRA Readiness](#). This contains monthly updates on the progress being made.

## Attributions

Image credit: Derivative work of <https://xkcd.com/2347/>  
[\*\*\*SBOM Tool Classification Taxonomy\*\*\*](#) developed by the National  
Telecommunications and Information Administration  
SBOM tool list from [\*\*\*awesome-sbom\*\*\*](#)

## Acknowledgements

Technical reviewer: Tuukka Pasanen, Pierre Pronchery, Ed Maste.  
Original inspiration for article: Peter Hansteen.

---

**Alice Sowerby** has been supporting the FreeBSD Foundation as a contract service provider since mid-2024. She is a multi-skilled program manager and open source leader with over 15 years in technical roles across cloud native, AI/ML, and DevOps. Currently active in the FreeBSD Foundation, CHAOSS, and the TODO Group, she brings expertise in program management and strategic leadership. Alice's past roles include Program Director at Equinix, with other experience spanning product management, UX, and developer relations. She is known for her collaborative approach and commitment to impactful, community-driven initiatives.

---

Copyright © 2026 Held by Owner/Author. Publication Rights Licensed to The FreeBSD Journal. This work is licensed under a [\*\*\*Creative Commons Attribution International 4.0 License\*\*\*](#).

---

3980 Broadway Street  
STE #103-107  
Boulder CO 80304  
1-720-207-5142  
[info@freebsd.foundation.org](mailto:info@freebsd.foundation.org)

