What are third-party dependencies and SBOMs?

What are third-party components?

There is this well-known XKCD comic showing the entire pile of all modern digital infrastructure being stabilized by one single library maintained by a random person from Nebraska. This simple satire perfectly summarizes both the incredible usefulness of third-party dependencies as well as all the dangers that they bring with them. While every company uses third-party components in their own products, handling them in a simple and secure manner is often still a closed book.

Simply put, anything that is not a "homemade" part of a software product, but pre-built by external developers is considered a third-party dependency. This includes pieces of code, libraries, frameworks, and services. With custom changes and forks this border has become a little bit fuzzier, but more on that later. In most cases - although not always - these components are open-source, i.e., units and services for which the source code is openly available on the web. Third-party dependencies are often organized within repositories which are integrated into the development processes used package managers, such as Maven with its Central Repository.

What are the benefits of using third-party dependencies?

Too many cooks may spoil the broth, but they write great software. When using well proven and well tested components, the danger of unknowingly building vulnerable solutions can be minimized (at least if they are kept up to date). Some crucial features, such as cryptography, should never be developed single-handedly, but rely on the efforts of an entire community of outspoken experts. Outsourcing the design and development of certain functionalities like logging, testing or the authentication process saves immersive resources. This gives the development team more time to focus their capabilities on building entirely new features and adding direct value to the customers.

How can third-party components endanger businesses?

Of course, using external components not only has its benefits, but may also introduce new issues. One of them is straightforward: No piece of software is a hundred percent secure. The widespread usage of some libraries virtually attracts malicious players and exploitable vulnerabilities in essential components can put whole industries in danger (e.g., log4j). Usually, the first countermeasure is an update to a non-vulnerable version of the affected software; sometimes, a mitigation of the attack surface is possible by adapting the integration between the third-party component and the own solution. In the worst case, the unsafe dependency needs to be exchanged with either an in-house element or another open-source alternative.
Another considerable risk for organizations using third-party components can arise from a lack of due diligence concerning their licensing. Full compliance to the terms and conditions of some licenses can have serious implications for companies, potentially putting their business model on the verge of a collapse. The catchphrase here is Copyleft, which requires the manufacturer of software incorporating certain third-party libraries to maintain their same open licensing terms. This is especially an issue regarding the free availability of source code.

How can I secure my business model?

You cannot fight what you cannot see. This holds especially true in large organizations managing thousands of software assets. Therefore, the basis for all measures in securing a company's usage of third-party components is building a comprehensive overview of them; a type of document better known as "Software Bill of Material" (SBOM). A deeper dive into the world of SBOMs and how they became one of the most important tools in modern software development will be featured in future articles of this series.

What problems can SBOMs solve?

A special place exists for organizations that do not track the third-party components used in their products it is called: Dependency Hell. It originally described a state of frustration arising when dealing with incompatibilities between software versions; but today, its meaning can easily be enhanced to include security and compliance issues emerging from dependency mismanagement.

Keep your friends close and your enemies closer.

Third-party components can be both at the same time: friends, as they can help with saving resources and often provide an easy way to implement industry best-practices, and enemies, as they might introduce (sometimes well-known) vulnerabilities. The proverb above does even describe the solution to this problem: 'Keeping them close' should be interpreted as 'Know, where they are' – and not only where they are, but also who and why. Having a comprehensive overview of its third-party dependencies is crucial for any organization to be able to detect and mitigate risks before they turn into exploitable vulnerabilities.

One document to rule them all.

Documentation is key in any effort to make the usage of third-party components more secure and compliant. The tool of choice for this is the Software Bill of Material, SBOM in short. It identifies all dependencies of a software product unambiguously, usually by label their providers, names, and version numbers. Additional fields to describe a component can include a hash of the source code or the respective licensing terms. Furthermore, the relationships between the elements should be contained to fully unleash the potential of the document. This adds visibility to the so-called transitive dependencies; components not directly implemented by a product itself but required by other third-party components. The end-result are often entire dependency trees, containing hundreds or thousands of units, some of them multiple times. Being able to conveniently pinpoint a single dependency (along with parent or child components), analysing its use case and estimating the implications of an update or substitution is an advantage that cannot be underestimated.

It might make sense to create one document for each application or module to sharpen focus, but SBOMs can easily be defined to describe entire product families or organizations to create a comprehensive dependency overview, detect weaknesses and help streamlining efforts.

How to create an SBOM?

No need to reinvent the wheel here – there are great tools available for nearly every software ecosystem, although their proper implementation can be quite challenging at times. Additionally, the number of restrictions and requirements imposed by various legislators on the creation of SBOMs grows with each day. Soon, the distribution of a respective SBOM alongside each product will become a necessity, demanding for full due diligence – but more on this in our article on the Cyber Resilience Act.

It's decisive to implement third-party analysis into the Secure Software Development Lifecycle (SSDL) as early and as automated as possible – neglecting responsibility in this field has tremendous consequences. Therefore, calling SBOMs one of the most important tools in modern software development is surely no exaggeration.