The risks of open-source software for corporate use

This article analyzes the origin of the open-source software (OSS) movement, how it relates to the ongoing trends in the enterprise and open source worlds, as well as the corresponding risks. Based on these inherent specificities, this article subsequently lays the foundations to control risks related to the use and contribution to open source without reducing its business potential.

Aristide Bouix MSc
is Senior Consultant and Cloud and Open Source Evangelist at the Cybersecurity department of KPMG.
bouix.aristide@kpmg.nl

Relying on open source software often means being at the edge of innovation

INTRODUCTION

The acquisition of Red Hat by IBM last year, which follows the acquisition of GitHub by Microsoft in 2018, demonstrates that open-source professional-grade software is no longer a utopia. Moreover, the rise of DevOps in corporate IT and the constant need to shorten the time-to-market for new digital products increased the temptation from product owners and development teams to use freely available code before analysis or validation to improve the delivery of Key Performance Indicators. The question is, is this practice safe for your security and compliance program? Alternatively, if it's not safe, what controls could be applied to your product team to mitigate the risks?

In this article, we will first present the origin, rise, and ideology that drives the OSS community. Second, we will explain some pitfalls of corporate open-sourcing both as a code user and at code producer level, followed by some controls and best practices aiming at keeping a healthy open-source ecosystem.

FROM THE FOSS TO OSS COMMUNITY, A BRIEF HISTORY

The emergence of free software started in the 1970s when Richard Stallman, a staff programmer at the MIT Artificial Intelligence Labs, modified the source code of the department printer to send error notifications when bugs occurred. After the replacement of the printer by a new model, Richard Stallman found the source code wasn't accessible, so he requested it from the printer company, who refused to give it. Getting quite upset with this situation, he decided to quit his job and to start working on an open software ecosystem called GNU in 1983. The name GNU is a recursive acronym for "GNU's Not Unix", mostly chosen as it was a real word and fun to say ([FSF17]). This date marks the creation of the Free Software Movement that later evolved into the Free Software Foundation (FSF). The approach to open source was highly oriented towards an interpretation of software liberty back then, with the famous quote from Stallman: "'Free software' is a matter of liberty, not price. You should think of 'free' as in 'free speech,' not as in 'free beer.'" ([FSF19]). This origin of the Free & Open-Source Software (FOSS) community still explains today why GNU open source licenses are more restrictive for corporations as they always underlie the release of any source code developed from a GNU licensed code.

In the 1980s, almost all software was proprietary, and one of the main goals of the FSF was to create the first real free operating system, and by the early 1990s, the GNU project had most of the major components of a free operating system, such as a compiler, editor, text formatter, mail software, graphic interface, libraries, etc. Nonetheless, the last missing piece of this ecosystem was a full operating system called Kernel. It's only in 1991 that Linus Torvald released this missing

Table 1. Top 5 GitHub enterprise domain name contributors according to the number of active users.

Enterprise domain	GitHub Users	Contributed repositories	Sum of GitHub stars from contributed repositories
microsoft.com	1,303	832	263,525
google.com	911	1116	538,587
redhat.com	442	338	87,192
ibm.com	300	226	56,091
pivotal.io	297	258	57,732

piece with the open-source project Linux, the complete Linux operating system incorporates many elements from Stallman's GNU project. While Linux is probably the world's largest and most successful open-source project in history, it's perhaps thanks to Torvald's second-biggest open-source project Git – which aimed to help developer collaboration on the Linux Kernel source code in 2005 – that the open-source world could start to take over the proprietary world. The main reason behind Git's success in open-source projects is that it didn't need to be continuously synchronized with a code repository. This specificity enabled a large number of developers to collaborate in a decentralized and asynchronous manner. However, it's GitHub in 2008 that imposed Git as the standard for open-source collaboration by giving it a web interface and a social dimension overpassing the legacy client-server version control systems such as CVS and SVN.

ONGOING TRENDS

Since then, IT continued to evolve and become more complex. The ease to reuse code and to collaborate brought by sharing public repository platforms such as GitHub, GitLab, or even DockerHub, gave a boost to Information Technology development.

Table 1 is derived from ([Hoff18]) research; it shows the main companies which contributed to open-source projects on GitHub in 2018. We clearly see that large Silicon Valley Tech companies continue to be the driving force. Still, many other companies outside the IT industry started to contribute heavily to the open-source world, such as Walmart, Nike, or Disney. Contributing to open source may first seems as counterintuitive to enable company business growth. Indeed, it goes against the ITIL concept of developing internal software and services to sell them externally. Yet, having your company using and maintaining open-source software may have several advantages, in particular, the following four aspects.

1. *Innovation:* By being open-sourced, a lot of software is easier and faster to test and customize to company needs without the constraint of a presale agreement. For this reason, relying on open source software often means being at the edge of innovation.
2. *Community:* Some open-source software such as Kubernetes or Prometheus benefits from the support of an active community of developers spanning across borders and organizations.
3. *Freedom:* By adopting an open-source standard, there is a reduced risk of a vendor lock-in as the software will be less subject to an EOL ("End Of Life") support situation as in a traditional software business model. Moreover, community members are often keen to look for an optimal alternative when a maintainer drops its support.
4. *Brand:* The combination of the three previous bullets can improve the image of a company to make it more attractive for newcomers. This is further discussed below.

As stated in the introduction, the above specificities of open-source software allow the development team to test and debug their prototypes faster without losing time and money in the presale agreement. It is, therefore, often a no-brainer in most companies to favor the reliance on open-source components for cost-oriented reasons. What is the additional gain of releasing internal software to the public, however? Some could consider that it may be related to Linus' law or wisdom of the crowd that would make open software more secure because it is easier to control. It could also mean that security vulnerabilities are easier to spot by malicious parties whose best interest isn't to fix the code but to keep it vulnerable as long as possible. A notable example of such a malicious third party may be the American National Security Agency itself, which was maintaining a significant database of exploits until they were leaked in 2017 by a hacking group called Shadow Brokers.

While the 21st century is the century of information technologies, the most significant advantage for companies to become open-source maintainers is to cultivate a vibrant technical brand. As already stated earlier, a missed digital transformation may lead to technical debt and a slow market death for the organization. It is, therefore, crucial to be able to attract talents capable of developing new products and leading the innovation path. Nonetheless, there is a lack of such profiles on the market nowadays, resulting in a highly competitive environment for businesses. Given this context, creating a robust technical label by maintaining successful open-source projects and being active in community events (blogs, talks, meetups, conferences, ...) is a determining factor. A more recent concept to justify open-sourcing activities that started to appear in the technology sector is promoting open source as a way to give back to society. Open sourcing can, indeed, be seen as a moral obligation to publish code built on the work of others.

RISKS RELATED TO OPEN SOURCE

As discussed in the previous section, open source can bring many benefits to your enterprise. However, in the following section, we will focus on the possible downsides and risks of open source for your company's business. Risks associated with open source tend to fall

into four different categories: *technical*, *governance*, *legal*, and *contributing*.

On a *technical layer*, three main risks factors can be identified:

1. *Code vulnerability:* While more significant open-source projects may have full-time sponsored developers to track reported issues and bring in new functionalities, some community-driven repositories may be side projects, maintained in best effort mode by a single developer during his or her free time. As a result, *security issues and vulnerabilities may remain unaddressed* for a more extended period, after being the object of a CVE (Common Vulnerability Exposure), than in a traditional commercial solution where the software vendor took engagements on security Service Level Agreement.
2. *Technology support:* It should also be noted that open-source software also mostly comes *without enterprise-grade support and sometimes a lack of coding and testing standards*. Accordingly, it becomes the responsibility of your organization to provide an additional engineer FTE (Full-Time-Equivalent) for supervision and troubleshooting. For instance, if the code doesn't completely fit your use case, it will be the responsibility of your engineers to bring the right modifications that make it fit into your product stack, and to find solutions to any bug in the meantime. This reason brought many non-tech focus institutions to rely on third-party consulting companies, which is one of the primary incomes for open-source-based businesses.
3. *Distribution:* Another common security hole of open-source software is the *distribution channel*. In case the code is published as a binary, most IT teams won't undertake further verification than checking if the provided hashes match the binary. Yet the binary and hashes often come from the same source and can both be compromised by an attacker. This happened in recent years when several Linux distribution repositories were hacked (Mint, Gentoo, etc.).

The *governance layer* of an open-source project is often a high-risk factor. In the open source world, there is a high level of trust. Community members rely on individual achievement rather than identity. A member having a sufficient record on a project may sometimes become the primary maintainer while having a Gmail

Table 2. The six most common open source licenses.

Licenses name	Permissions	Conditions	Limitations
GNU Alfero General Public License v3.0 (GNU AGPLv3)	Commercial use Distribution Modification Patent use Private use	Disclose source License and copyright notice Network use is distribution Same license State changes	Liability Warranty
GNU General Public License v3.0 (GNU GPLv3)	Commercial use Distribution Modification Patent use Private use	Disclose source License and copyright notice Same license State changes	Liability Warranty
GNU Lesser General Public License v3.0 (GNU LGPLv3)	Commercial use Distribution Modification Patent use Private use	Disclose source License and copyright notice Same license (library) State changes	Liability Warranty
Mozilla Public License 2.0	Commercial use Distribution Modification Patent use Private use	Disclose source License and copyright notice Same license (file)	Liability Trademark use Warranty
Apache License 2.0	Commercial use Distribution Modification Patent use Private use	License and copyright notice State changes	Liability Trademark use Warranty
MIT License	Commercial use Distribution Modification Private use	License and copyright notice	Liability Warranty

address as the only identification. This situation has been the cause of last years' greatest hacks, such as the compromise of NPM to steal cryptocurrency credentials ([Clab18]). In this case, only a small library was compromised, but the incident took a broader dimension as many other libraries depended on it, resulting in two million downloads a week. As maintainers sometimes have an unclear identity, the probability of having them impersonated by a malicious party who compromised their account is, consequently, real.

On the *legal side*, another risk that applies for both code consumers and code publisher companies, which is connected to the risk related to contributing, is the choice of an open-source license. As the FSF was built on a sharp interpretation of the meaning of software liberty, which has been highlighted in the first section, GNU licenses tend to be the most restrictive on what is allowed to do with software.

However, most code based on open-source software often ends up being closed source, making it difficult for code owners to prove it and force a commercial entity to reveal the code. A recent comic example of a licensing issue may be the lawsuit between Ubiquiti and Cambium ([Ging19]), where the two companies are suing each other regarding license violation while they both have been found violating a GNU General Public License for the underlying technology. To summarize, while the legal risks of open source are often more limited than with traditional software, a non-respect of some license specificities may, in some cases, lead to copyright infringement procedures (see box).

As a reminder, Table 2 enumerates the properties of the six most common open-source licenses. As discussed earlier, GNU licenses are the most restrictive as they impose to release the associated source code as well as the list of changes, but you're allowed to use the GNU trademark in the name of your project. The LGPLv3 license is a bit less restrictive as you can keep your code closed source if you only use the open-source code as a library for your project. The Mozilla license is similar to the LGPL, except you don't have to state the changes made to the codebase and cannot use the Mozilla trademark. As a result, if you do not intend to commit to the open source community, the best strategy is to target projects licensed under the Apache or MIT license as you don't have an obligation to disclose the source. Note that the MIT license is a bit more permissive than the Apache License 2.0. It is usually considered as a best practice to go for a less restricted license when unsure, as, the risk of license violation is then decreased. Some mitigations to this risk are proposed in the last section of this publication.

In the open source world, there is a high level of trust

Regarding the risks associated to *contributing* to open source, we may observe that the open source business may be divided into two distinguished models.

- On the one hand, traditional *open core vendors*, such as Red Hat, develop a free Core software, such as CentOS, on which they build proprietary extensions bundled with support and services, such as Red Hat Enterprise Linux (RHEL).
- On the other hand, there are *cloud-hosted services* built with open source software. They include additional features such as cross-region replication, automated scalability, and monitoring dashboards. A cloud-hosted service may be sold as Software as a Service (SaaS) or Platform as a Service (PaaS) by the Open Core vendor itself, such as MongoDB Atlas or as an offer in a Public Cloud Provider portfolio such as AWS DocumentDB.

Nowadays, both of those business models seem to be challenged by the significant growth of Public Cloud Service Providers this last decade. Public Cloud has played a centralizing role in the way networks, and IT, operate by placing all of the needed IT services with a single provider. It simplifies workflows, billing and decreases the need for service maintenance and support. As a result, most IT teams would choose a less performant managed service provided by one of the major Public Cloud Players (AWS, Azure, GCP, ...) rather than going through the trouble of setting up a

new contract with a smaller SaaS player. It ended up in a significant financial loss for some open-source companies, such as Elastic, which initiated a trademark lawsuit against AWS, after having intertwined more and more private components in their new releases.

Other open source firms decided to compete by introducing a new type of Public License intending to protect their Intellectual Property from concurrent SaaS offerings. This tactic initiated by MongoDB through the Server Side Public License ([Mong18]) seeks to force the release and open sourcing of a complete SaaS implementation with infrastructure subcomponents as a condition to sell the service, excluding MongoDB as they own the Intellectual Property (IP). Because of this license, the new AWS service DocumentDB is based on the open source code that was released under the previous license two years ago; on a dark note, such a decision may compromise the inter compatibility of MongoDB with any other SaaS service in the future.

To compile this last point:

1. Open sourcing some internal IT products is good for a company's image and may improve internal IT processes, and help to find new talents
2. Open sourcing core business components represent a significant risk of favoring market competitors and hosted offerings no matter the license

Due to the difficulties mentioned above, we consider contributing to open source as a substantial strategic value for modern companies, but with a high risk if relied on for its financial value.

> Prohibiting the use of open source components in the organization is equivalent to ignoring risks related to open source

WHAT SECURITY SAFEGUARDS YOU SHOULD PUT IN PLACE

Because of the aforementioned risks, many security and compliance teams still manifest a pessimistic view of the use of open source. The reality is, similarly to the adoption of cloud computing, the use of open-source software is so straightforward and convenient that it is probably already widely used across the organization as Shadow IT. Also, as it goes for any business enabler, prohibiting the use of open source components in the organization is equivalent to ignoring risks related to open source, which results in increased risk exposure for the business. Even if your organization doesn't have a strong open-source culture, regulating is always a better approach than ignoring. And while you're working on a regulation for the use of open source, it is good practice to already think ahead about potential future extensions covering the release of open source components. Notably, as we've previously discussed, the use of open

source components under some licenses underlie the open sourcing of the resulted software.

As for any new internal capability, the use of open source within a company should follow a clear and dedicated open source policy explaining to IT staff members how to use open source and referring to internal processes and procedures. In the absence of standards and central governance, open-source software tends to appear as a grey controlled area from a security point of view. As a result, we recommend keeping in mind a zero-trust model for writing such a document, in the sense that any free code may include major security vulnerabilities or be insufficiently tested. It is then your company's responsibility to fork and control the repository instead of copy-pasting the original in production, build from source, test the code, and perform minimal due diligence of its maintainers.

As the writing of any policy is always a long and tedious task, I have included a list of five common situations that should be covered:

- When open source code is used as an internal tool.
- When open source code is used unmodified in a business product (library).
- When open source code is modified and used in a business product.
- The process for contributing to an open-source community during working hours.
- The process to open source internal products.

This article could go further by giving you additional guidelines, yet the writing of any policy is dependent on the culture and business of an organization. It is better to get the help of an external IT governance specialized third party to write a policy tailored to your needs. As you can imagine, there is hardly a better policy to be made public by an organization than open source. Consequently, you can find some online reference documents from Google, the Linux Foundation, or Gitlab at the following Github repository ([Todo19]) maintained by the TODO (Talk Openly Develop Openly) group.

As a policy without implementation doesn't go very far, here is a list of probably the most critical security controls to start working on:

- *Centrally monitor* which open source project/libraries are used as well as their *level of patching and licensing model*.
- The permissions of what is possible per licensing model per project should be *identified and listed*.
- *A communication channel*, for instance, a slack webhook, should be put in place to signal updates from each reference project repository. Furthermore, the responsibility of upgrading and patching open source components should be clearly defined. We recommend the project team, but it could also be a central DevOps team for some organizations.
- If an open source project starts to be modified to better fit your internal needs, you should *fork the original* repository and regularly analyze, merge, and troubleshoot commits from the master. As updates from open source repositories are not necessarily adequately tested and assessed, it is your company's duty to raise issues or propose alternative commits in case of doubt.
- As open source repositories may be compromised, always *build from source* following an automated CI/CD process. This automated build process should include both *a static as well as a dynamic vulnerability scanning engine*.
- *Monitor the activity*, for instance, the number of commits per month, of the open source repositories your organization relies on. If you find that an open source project you use is slowly abandoned, start studying alternatives and prospects within the project community to help your research and decision. Also, try to negotiate with the project owner to take ownership of the repository in case no alternative fits your use case. In all circumstances, *never continue using a no longer maintained repository for an extended period!*
- Lastly, before reaching the previous situation, *establish privileged relations with the core developers of the open source software you rely the most on. Set up a small open source give-back budget*, and regularly donate to the external contributors who make your company live.

Figure 1 is an example of how some of those controls can be mapped to address the previously identified risks.

The adoption of open source should be driven following a zero-trust model

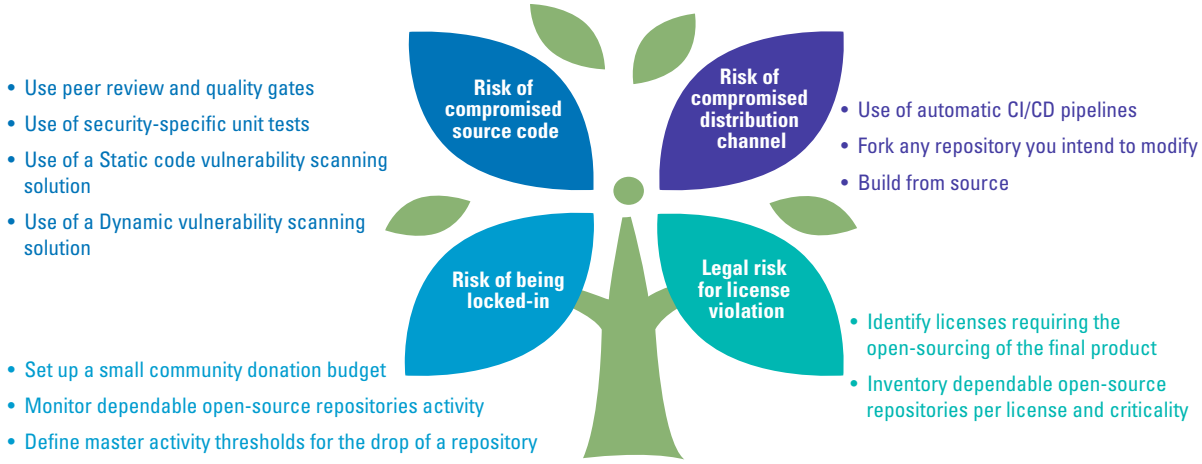

Figure 1. Use of FOSS, Risks-Controls mapping.

CONCLUSION

From a marginal phenomenon in the pre-Internet era, open source moved to the norm for boosting and promoting the world's most innovative software projects. As enterprise IT becomes more code-oriented, processes to create, use, store, and share code internally and externally became an important area of control. In this article, we specifically focused on the reasons that could help justify the use and contribution to open-source projects within your organization, followed by the presentation of some controls to help limit the risks we previously identified related to the use of open-source software.

Due to the fact that open source projects do not generate profits following a traditional license selling business model, they tend to be intrinsically non-profit, which underlie additional risks. Those risks can be technical with the vulnerability of the source code and the lack of vendor support, related to the weak governance of an open-source project, or even legal with possible license violation. Because of those potential intrinsic risks, the adoption of open source in the enterprise should be driven following a zero-trust model where relied external components shouldn't be trusted, but regularly analyzed and assessed at each new update.

At the end of the day, a successfully driven open-source compliance program is key to supporting your company's capability to innovate, develop your strategic market value, and achieve a successful digital transformation.

References

[Clab18] Claburn, T. (2018, November 26). Check your repos... Crypto-coin-stealing code sneaks into fairly popular NPM lib (2m downloads per week). *The Register*. Retrieved from: https://www.theregister.co.uk/2018/11/26/npm_repo_bitcoin_stealer/

[FSF17] FSF Inc. (2017). Overview of the GNU System. Retrieved from: https://www.gnu.org/gnu/gnu-history.en.html

[FSF19] FSF Inc. (2019). What is free software? Retrieved from: https://www.gnu.org/philosophy/free-sw.en.html

[Ging19] Gingerich, D. (2019, October 2). When companies use the GPL against each other, our community loses. Retrieved from: https://sfconservancy.org/blog/2019/oct/02/cambium-ubiquiti-gpl-violations/

[Hoff18] Hoffa, F. (2018). Who contributed the most to open source in 2017 and 2018? Let's analyse GitHub's data and find out. Retrieved from: https://medium.com/@hoffa/the-top-contributors-to-github-2017-be98ab854e87

[Mong18] MongoDB, Inc. (2018). Server Side Public License. Version 1, October 16, 2018. Retrieved from: https://www.mongodb.com/licensing/server-side-public-license

[Todo19] Todogroup (2019). Open Source Policy Examples and Templates. Retrieved from: https://github.com/todogroup/policies

About the author

Aristide Bouix MSc is Senior Consultant at KPMG Cyber. Fond of cloud, open source and new technologies, Aristide leads the DevSecOps and Cloud Configuration Audit service offering within KPMG Netherlands.