Managing Risk in Open Source

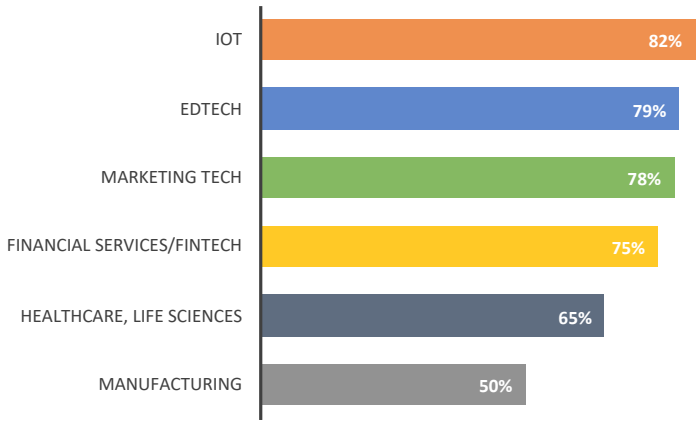
September 2020

Contents

Introduction

Open Source Software (OSS) is freely distributed code and/or binaries that can be used, modified, and redistributed in accordance with its licensing terms. The use of open source in the development process is pervasive and will continue to grow over time. Results from the Synopsys 2020 Open Source Security & Risk Analysis Report demonstrates the dependency on open source.

Figure 1. % of Open Source in Codebase by Industry[1]



The use of open source can provide benefits by reducing development cycles, thereby reducing the cost and time required to deliver features to the customer. At the same time, OSS can increase business risk if not properly governed. The lack of proper governance can result in legal action due to the violation of licensing terms, unstable products brought on by the inclusion of unreliable projects, and vulnerable solutions due to the incorporation of open source components containing weaknesses.

The objective of this document is to provide practical guidance to manage open source risk (see risk considerations below). Legal, product, software development and cybersecurity leadership must promote these practices to ensure product managers, developers and other appropriate personnel understand open source risk implications and take the necessary steps to mitigate them.

Open Source Risk Management Considerations:

- Understand open source license implications
- Select trustworthy projects
- Manage vulnerabilities in open source
- Govern code contribution to external projects
- Perform open source inventories and audits

[1] Source: 2020 Synopsys Open Source Security and Risk Analysis Report

Understand Open Source License Implications

Open source licenses generally fall under one of two categories: copyleft or permissive. Copyleft licenses such as GNU General Public License (GPL) govern downstream redistribution and require use of the same licensing terms. For example, copyleft licenses allow developers to use software without incurring fees and prohibits the developer from charging fees for the final product. Copyleft licenses may not be monetized.

Permissive licenses such as Berkeley, Apache, and Massachusetts Institute of Technology (MIT) don't place these restrictions on the use or distribution of open source. The business has more flexibility in this model.

The Open Source Initiative provides a complete reference of open source licenses and associated terms. A general guideline for open source licensing is described in Table 1. Open Source Licensing Guidelines. Your business model will determine the appropriate licensing guidelines. For example, if the final solution will be sold for profit, copyleft licensing may be problematic. Consult legal personnel within your organization and rationalize what's acceptable, questionable, and prohibited for your business.

Table 1. Open Source Licensing Guidelines

Guideline	Open Source License
Acceptable – Very limited risk to the organization. License can be used without consulting leadership.	▪ Public Domain ▪ BSD ▪ MIT ▪ Apache 1.1 or 2.0 ▪ Artistic License 1.0 or 2.0 ▪ PHP License ▪ Python Software ▪ WTFPL ▪ CC0 ▪ Creative Commons Only ▪ Foundation License ▪ Boost Software ▪ OpenSSL/SSLeay
Questionable – May pose a risk to the organization. Approval is required prior to distributing code using these licenses. Open source licensed under these licenses can be used for internal purposes.	▪ GPL 2.0 ▪ LGPL 2.1 ▪ Mozilla Public License 1.1 or 2.0 (MPL) ▪ CDDL ▪ CPL or IBM ▪ Eclipse Public License ▪ GPL 2.0 + plus exception ▪ Apache 1.0

Prohibited – Significant risk can result from using open source under these licenses. Consult legal personnel for guidance if any of these licenses are required.	▪ GPL 3.0 ▪ LGPL 3.0 ▪ Affero GPL v1 or v3 ▪ Sleepycat ▪ GNU Documentation ▪ License ▪ Creative Commons ▪ ShareAlike ("CC BY-SA") ▪ Open Software License ▪ Academic Free License

Select Trustworthy Projects

Deriving the full benefit of open source software requires thoughtful project selection. The key is identifying trustworthy projects as they typically result in lower risk. Developers should take the following steps to perform reasonable due diligence when assessing the trustworthiness of projects.

1. *Assess the licensing terms* – Identify project licensing and reference Table 1 above to determine the appropriate actions.
2. *Validate the quality of the project* – Source code quality and level of documentation are indicators of project quality. Assessing source code quality does not require a line by line review of each file. Reviewing representative source code for proper structure, organization of functions, and reasonable comments can go a long way.

   Projects should be well-documented to communicate technical and user level details. Technical documentation should highlight installation, configuration, and support requirements. User documentation must provide reference information for software features and enhancements. Poor documentation can be an indicator of project quality.
3. *Ensure active community support* – The project must be actively maintained to ensure the currency of technologies (e.g., cryptographic algorithms) used, enhancement of features, and remediation of vulnerabilities. Reviewing code commits will provide insight into how well the project is maintained. Frequent and current code commits from a broad group of developers provides assurance that the project is kept up-to-date and community support is available.
4. *Verify the project is supported by a reputable community* – The community associated with the project should consist of industry professionals representing a known legal entity or known individual(s) that has established credibility. Communities associated with nation states known to be adversaries of the U.S. should be avoided.
5. *Identify the length of time available* – The use of immature projects should be assessed. Newly released open source should be avoided in high-risk code unless extensive testing is performed. Developers must use reasonable judgment and avoid introducing unnecessary risk.

Manage Vulnerabilities in Open Source

Open source can enter code through several channels such as approved components, third party libraries, third party development, and reuse of code. Identifying vulnerabilities early in the development process will reduce re-work and ultimately reduce risk. There are two methods to reduce vulnerabilities in open source: scanning for vulnerabilities and limiting the attack surface.

The first method, scanning for weaknesses, to reduce vulnerabilities is apparent. Conducting dependency checks and scanning for known vulnerabilities is critical. There are multiple options available to make this happen. Source code repositories such as GitHub include features to identify vulnerabilities in open source. Additionally, there are several commercial tools (e.g., Blackduck, FOSSA, and Whitesource) in the marketplace to accomplish this objective.

Reducing the attack surface is not top of mind when selecting projects. Developers should select projects that provide the features needed rather than large projects that enable expansive functionality. Most of the functionality included in the larger project will not be used and unnecessarily increases the attack surface. For example, when developing an IoT solution that implements Zigbee as the wireless communication protocol, developers should avoid projects that include additional protocols such as NFC (Near Field Communication), BLE (Bluetooth Low Energy), Bluetooth, or Z-Wave.

Addressing vulnerabilities and limiting the attack surface early in the development process is essential. Taking this approach reduces cyber risk, cost and the time required to bring solutions to the marketplace.

Govern Code Contribution to External Projects

Developers must obtain approval from leadership prior to contributing to external projects on behalf of the company. The intent is to prevent adverse impacts to the company's intellectual property assets.

External projects may require a Contributor License Agreement (CLA), Copyright Assignment Agreement (CAA), or no signed agreement. The Contributor License Agreement (CLA) enables the original contributor to retain copyright ownership of their contributions. It also grants the project a broad set of rights such that the project can incorporate and distribute the contributions as needed.

The contributor transfers copyright ownership of the contributions to the project in a Copyright Assignment Agreement (CAA). In this scenario the project can then license it as they see fit. It should be noted that a CAA most often grants very broad non-exclusive rights back to the contributor so that they can use, distribute, sublicense their contribution freely.

In a non-signed agreement scenario the rights granted by submitting the contribution becomes ambiguous and has the potential to elevate business risk. Research is needed to assess the risk & identify the appropriate actions to protect the business.

Perform Periodic Open Source Risk Assessment

Periodic assessments of open source usage can help to manage risk. This activity complements the ongoing review of open source during the development process. Theses assessments should provide an inventory of open source libraries including direct and transitive dependencies.

At a minimum, open source risk assessments should be conducted quarterly. Events that trigger the need for additional assessments include distribution of software and M&A transactions. The results of the assessment must be reviewed with cybersecurity, product, and legal decision makers.

Conclusion

Open source software plays an essential role in many products. Research shows that, on average, 60% - 80% of code is open source. The benefits open source brings are immeasurable, but diligence is needed to contain the risk. Developers, product managers, and technology personnel must take the necessary steps to consider risk early in the development process. Taking the actions described in this document will significantly reduce risk and increase product viability.