Prevalence and Evolution of License Violations in npm and RubyGems Dependency Networks

Ilyas Saïd Makari, Ahmed Zerouali, and Coen De Roover

Software Languages Lab, Vrije Universiteit Brussel, Belgium
{ilyas.said.makari | ahmed.zerouali | coen.de.roover}@vub.be

Abstract. It can be challenging to manage an open source package from a licensing perspective. License violations can be introduced by both direct and indirect package dependencies, which evolve independently. In this paper, we propose a license compatibility matrix as the foundation for a tool that can help maintainers assess the compliance of their package with the licenses of its dependencies. Using this tool, we empirically study the evolution, popularity, and compliance with dependency licenses in the npm and RubyGems software package ecosystems. The size of the corresponding dependency networks renders verifying license compliance for indirect dependencies computationally expensive. We found that 7.3% of npm packages and 13.9% of RubyGems have direct or indirect dependencies with incompatible licenses. We also found that GPL dependencies are the major cause for incompatibilities. Our results provide a good understanding of the state of license incompatibilities in software package ecosystems, and suggest that individual ecosystems can differ significantly in this regard.

Keywords: software license, license compatibility, package dependency, npm, RubyGems

1 Introduction

Open Source Software (OSS) has become the standard in the software industry, encouraging collaboration and reuse. Much of this Open Source Software is reused by developers incorporating it into their own software as independent building blocks, e.g., packages. As a matter of fact, online package repositories, such as npm for JavaScript and RubyGems for Ruby, provide an enormous amount of free open source software packages. These packages are available for any developer to use in their own software, but in many cases under particular restrictions imposed by the package owners. These restrictions constitute a software license.

Open source packages can be distributed under licenses with varying degrees of freedom, decided on by their owner. The variety among license types complicates using multiple package dependencies in one software system. License restrictions of one package may conflict with the restrictions of another, which encumbers ensuring legal compliance of one's own software with the licenses of its dependencies. Moreover, depending on a package that is in violation of a license may lead to one's own software violating the same license.

To make sure no conflicts will occur, developers can review each package's license prior to incorporating it into their own software. However, this quickly becomes cumbersome when these used packages depend on other packages, which may in turn depend on other packages, and so on, forming a huge network of directly and indirectly incorporated dependencies. Worse, developers have no control over the dependencies that get included transitively into their software. Online package repositories such as npm and RubyGems which form massive dependency networks only aggravate the problem.

In this paper, we conduct an empirical study into the prevalence and evolution of open source package licensing. Using a license compatibility matrix that includes more licenses than prior work, we study the severity of license incompatibilities in the dependency networks of npm and RubyGems, while distinguishing between incompatibilities with direct or indirect dependencies. More concretely, we study the following research questions:
RQ_1: **What are the most prevalent licenses in package repositories?**
RQ_2: **To which extent do packages rely on direct dependencies with incompatible licenses?**
RQ_3: **How does license incompatibility spread across package dependency networks?**
The tool used to answer these questions can also be used as support by package maintainers in detecting dependency license incompatibilities for their software.

2 Background and Related Work

This section provides the necessary background information and an overview of the related work concerning the use of licenses and compliance therewith.

2.1 Software Licenses

Under the Berne Convention, the international agreement concerning copyright law, an author automatically obtains the exclusive copyright to their work. This applies to software as well, implying that software creators have the ultimate say when it comes to granting rights and placing restrictions. The latter are typically expressed through a software license, i.e., a contract between the copyright holder (licensor) and the user of the software (licensee). For open source software, it is generally advised to choose from an existing license that has already been reviewed by organizations such as the Open Source Initiative (OSI) and the Free

Software Foundation (FSF). The SPDX[1] standard provides a unique agreed-upon identifier for each such license. They can be categorized as follows from least to most restrictive:

- **Public domain**: Software that is released into the public domain is free to use, copy, modify, distribute and sell without any restrictions or attribution required. For example, the "Unlicense" is a template for dedicating software to the public domain. Note that software with no specific license mentioned cannot be automatically interpreted as public domain [2].
- **Permissive licenses**: Software released under permissive licenses (i.e., MIT and Apache), have little restrictions imposed. Permissive licenses allow software reuse for any purpose, but redistributions and derivatives of the work must include the copyright and license notice from the original author. In general, there are little to no restrictions that will cause conflicts when a permissive license is combined with stricter ones.
- **Copyleft licenses**: These form the most restrictive category of OSS licenses, due to their prohibition of proprietization. They require any derivative work to be distributed under a license that preserves all the rights established by the original license. The most popular example is the GPL license, which, among other restrictions, requires derivatives to be licensed under the same GPL license. Copyleft licenses that impose these rules on any kind of derivative work are called strong copyleft licenses, whereas weak copyleft licenses make exceptions for some types of derivative work. For example, weak copyleft licenses like LGPL make exceptions when the original work is used as an independent building block (e.g., a library). In addition, there are network copyleft licenses that expand strong copyleft licenses to software that can be ran via internet or other networks, e.g., AGPL and OSL 3.0.

2.2 License Compliance

Despite the ubiquity of OSS packages, one must be careful when incorporating them as a software dependency. For example, the licenses of two dependencies may contain contradictory statements, rendering their combination legally impossible. This may also be the case for the license of the software into which the package is included. For instance, software released under the permissive MIT license cannot include packages released under the more restrictive GPL license as GPL requires all derivatives to be released under the same license.

In order to identify license incompatibilities, there must be a reliable source of truth that dictates which pairs of licenses are compatible with each other. The FSF provides two lists[3] stating which licenses are (in)compatible with the GPL.

[1] https://spdx.org/licenses/
[2] https://choosealicense.com/no-permission/
[3] https://www.gnu.org/licenses/license-list.en.html

However, these lists only provide an answer for a limited number of license combinations. They provide no information about, for instance, the combination of MIT and BSD. Unfortunately, the required source of truth cannot be generated automatically, since licenses are written in natural language with complex legal terms [1]. The following approaches to determining license compatibility have been proposed instead:

– **Set-based approach**: This approach is used by libraries.io's license compatibility tool[4]. Licenses are grouped into disjunct sets ordered from least to most restrictive: public domain, permissive, weak copyleft, strong copyleft, etc. To say that license A is compatible with license B: (1) A must be either public domain, permissive or weak copyleft; or (2) If A is strong copyleft, then B must be either weak copyleft, strong copyleft or network copyleft; or (3) If A is network copyleft, then B must also be network copyleft. However, generalizing license compatibility in this manner will not account for the many exceptions that exist.
– **Formal approach**: This approach was described by Gangadharan et al. [2]. All terms and conditions of licenses are formally specified thus enabling the algorithmic detection of incompatibilities. The algorithm uses the following rule: license A is compatible with license B if all license clauses from A are compatible with those of B. However, it is far from trivial to convert all license text into a formal specification.
– **Graph-based approach**: Relations between different licenses can be modelled using a directed graph [3] in which each node denotes a license and each directed edge between two nodes is used to denote a one-way compatibility, i.e., an edge from license A to license B implies that A is compatible with B, but not necessarily the inverse. Similarly, all licenses that can be reached from license A are also compatible with A. The graph was constructed using the rules that apply when the derivative work is based on the original work. This is important, since weak-copyleft licenses (e.g., LGPL) can make the distinction between work that is "based on a library" and work that "uses a library". Later on, this graph was expanded by Kapitsaki et al. [4] to include additional licenses (e.g., MPL). The resulting graph has subsequently been simplified by removing the incompatibility edges [1].

2.3 Studies on License Compliance in Dependency Networks

Dependency networks are created when one software package depends on other packages which in turn depend on their own packages. The resulting network may aggravate problems detected in a single package, as the issue may rapidly spread to all direct and indirect dependents. This has promoted researchers to conduct studies on license compliance in dependency networks. Kechagia et al. [5] have

[4] https://github.com/librariesio/license-compatibility

studied the FreeBSD ports collection from this perspective. They found that GPL-licensed applications that run under FreeBSD use LGPL-licensed components. Qiu et al. [6] have studied the prevalence of dependency-related license violations among 419,708 npm packages. They found that very few packages (0.644%) have dependency-related license violations. In this paper, we found that this proportion increases to 7.3% when we study the entire ecosystem of npm packages while inspecting more license combinations that were not considered in previous studies. In addition, developer surveys in [6] revealed two main causes for these violations: 1) developers overlook and misunderstand them; and 2) developers find it difficult to manage them and ask for tool support.

Novelty of our contribution. Our work extends and updates the insights from previous studies by considering a larger license compatibility matrix that includes additional license combinations. We also carry out an empirical analysis of license violations in both npm and RubyGems dependency networks, and compare them. To the best of our knowledge, we are the first to study licensing in the RubyGems ecosystem. Moreover, we provide a visual tool that helps developers find incompatible transitive dependencies easily.

3 Research Method

This section discusses the details of the research method we used to answer the research questions enumerated in Section 1, including how we created an extended license compatibility matrix.

License Compatibility Matrix. We created an extensive and up-to-date compatibility matrix starting from the aforementioned graph proposed by Kapitsaki et al. [1]. The graph alone does not suffice for our study, as it does not include some of the licenses used by popular npm and RubyGems software packages. Moreover, for the context of this work, the meaning of the edges in the graph needs to change from a project being "based on" another project to a project "depending on a library". For example, a project with a permissive license such as MIT is allowed to include an LGPL dependency. This requires including additional edges that account for this possibility. Finally, the absence of a path between two licenses A and B in the graph does not suffice to deem A and B as incompatible. It merely means that the compatibility cannot be determined from the graph alone.

Our matrix therefore aims to provide a definitive answer for as many actual license combinations as possible. To this end, we started from the graph and manually included information from the following sources: (1) the FSF has two lists[5] stating which licenses are compatible and incompatible with the various GPL licenses; (2) the European Commission maintains a matrix that has an an-

[5] https://www.gnu.org/licenses/license-list.en.html

swer for a couple more license combinations [6]; and (3) the FSF provides a matrix[7] to determine which (L)GPL licenses are compatible with each other.

Currently, our license compatibility matrix includes 1,681 pairs of licenses. However, it does mark the compatibility of 205 (12.2%) license pairs as "Unknown". These are the license pairs for which we could not determine the compatibility either due to a lack of publicly available information or due to varying opinions among lawyers. Many licenses and combinations thereof remain to be tested in the courts.

Package License Extraction: Package managers have a structured method to keep track of metadata, such as licenses and dependencies. Packages from npm come with an automatically created "package.json" manifest that includes these metadata, while RubyGems has "Gemfile". To have the information about licenses used by npm and RubyGems packages, we relied on the latest version of libraries.io Open Data [8] that was released on 12 January 2020. libraries.io dataset contains metadata of packages hosted on 32 different package repositories, including npm and RubyGems. It identifies the licenses of each package using the standardized SPDX identifiers. To identify the versions of the dependencies of each package, we relied on the dependency constraint resolver proposed in [7], which supports the npm and RubyGems package distributions. We determined the appropriate versions of packages to be included for each dependent according to the constraints for its run-time dependencies. As some constraints may resolve to different versions at different points in time, we used the libraries.io snapshot date as the resolution date. This implies that we study licenses in packages as if they were installed or deployed on January 12th, 2020. Having determined the versions of all direct dependencies, we then identify the indirect ones.

More specifically, we inspected $749,823$ and $94,953$ npm and RubyGems packages (i.e., their latest version) which depend on $66,404,594$ and $1,190,422$ run-time dependencies, from which $3,527,000$ and $211,336$ are direct, respectively. We only focused on run-time dependencies because development ones are only needed during development (e.g., unit tests libraries, transpilers, etc). To determine the license compatibility between a dependency and its dependent, we relied on our license compatibility matrix. A dependency is either compatible, incompatible or unknown. We found that 7.3% (54,778) and 13.9% (13,271) of npm and RubyGems packages have direct or indirect dependencies with incompatible licenses, respectively. We also found the amount of unknown incompatibilities for all (transitive) dependencies to be very small, i.e., 2.9% for npm and 5.6% for RubyGems. After having a deeper look at which license combinations were unknown, it became clear that it mostly consists of multi-licenses, i.e., a package can be used with license A or with license B, e.g., *keypair* [9]. This is due to the

[6] https://ec.europa.eu/jrc/sites/default/files/20150930-second-best-practices-tto-circle-gentile_en.pdf
[7] https://www.gnu.org/licenses/gpl-faq.en.html
[8] https://libraries.io/data
[9] https://www.npmjs.com/package/keypair

fact that our compatibility matrix can only provide answers for pairs of singular licenses. Moreover, there are some packages that can be used with different licenses depending on the third-party that is going to use them. For example, developers of *webgazer* [10] state that their package is *"licensed under GPLv3"* while *"companies have the option to license WebGazer.js under LGPLv3 while their valuation is under* $1,000,000*"*. Extracting such information requires us to process natural language text written in different files like README and then check conditions with dependent packages, which is a complex task. Thus, it is important to note that any incompatibilities with multi-licensed packages will not be taken into account in this study.

4 Results

We now evaluate the state of license compliance among the dependency networks formed by the popular package repositories npm and RubyGems.

RQ_1: What are the most prevalent licenses in package repositories?

We start by determining the most common open source licenses found in each package repository. The ecosystems around some repositories might have a more permissive climate than those around others. According to a blog post [8] from WhiteSource, in 2012 open source software was predominantly using copyleft licenses (59%). Three years later, GitHub [9] reported a shift towards more permissive licenses. With this research question, we want to investigate whether such a shift has also happened among the npm and RubyGems packages, by investigating the evolution of the most popular licenses over time.

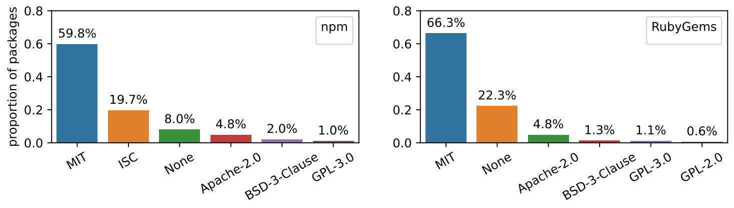


Fig. 1: Top 6 most used licenses

Figure 1 shows the top-6 most popular licenses in npm and RubyGems. We observe that the MIT license is the most popular by far. 59.8% and 66.3% of npm and

[10] https://www.npmjs.com/package/webgazer

RubyGems packages make use of the MIT license. Most licenses in the top 6 list are permissive.

From the figure, we also observe that there is a considerate proportion of packages that do not have any license (i.e., *None*), not even a public domain waiver like "Unlicense". This is worrying as without a specified license, no one can copy, distribute, or modify the package except for the original developers. In fact, it is advised not to make use of such software [11]. We expect these packages to cause many license incompatibility problems in package dependency networks.

Figure 2 depicts the evolution of license usage in the npm ecosystem. The X-axis shows the creation date for each package. We observe that the permissive MIT license has been popular since the beginning. Furthermore, in 2014, there seems to be a sudden increase of the ISC license, taking over the popularity of the BSD-3-Clause license. This can be explained by the v1.4.8 update[12] of npm in May 2014, where the previous default license, BSD-3-Clause, was replaced by the ISC license, which is similar to the BSD-2-Clause and MIT licenses. This caused all new npm packages, from May 2014 onwards, to be licensed under the ISC license by default. The main difference between ISC and its predecessor is that ISC does not include a non-endorsement statement preventing users of a package to claim they are endorsed by the author of that package. The urge to replace BSD-3-Clause with ISC, might be due to the endorsement of the OpenBSD project [10]. However, in 2015, the Internet Systems Consortium (creators of the ISC license) stated that there is no longer a good reason for ISC to have its own license. Even though ISC is equivalent to the very permissive MIT license, lawyers prefer the more popular license that they are already familiar with [11]. There is now an ongoing discussion in npm to change the default license from ISC to MIT. This is mainly to avoid license proliferation, as the permissive ISC license poses no threats when it comes to license compliance.

Figure 3 shows the evolution of license usage in the RubyGems ecosystem. RubyGems is a repository that has been around for a bit longer, compared to npm. From the figures, we observe that MIT gradually evolved into the most popular license as the number of packages without a license decreased to just 6% of the packages created in 2019. At the start of RubyGems, it did not come with a package manager. Later in 2009, Bundler was eventually introduced as its package manager, but did not come with a standard license until 2011[13]. The introduction of MIT as the standard license for RubyGems may be a factor to MIT's rise in popularity as well. Figure 3 also reveals that the Apache-2.0 license is gradually increasing in popularity. Over the last few years, Apache has become the second most popular license choice within the RubyGems ecosystem. Overall, the climate of the ecosystem is predominantly permissive. However, even though Apache-2.0 is a permissive license, it is not fully compatible with the GPL. Apache-2.0 is only compatible with GPLv3. Thus, developers have to

[11] https://choosealicense.com/no-permission/
[12] https://github.com/npm/cli/blob/latest/changelogs/CHANGELOG-1.md
[13] https://github.com/rubygems/bundler/blob/master/CHANGELOG.md

be careful not to use any Apache-licensed dependencies in their GPLv2-licensed software directly or indirectly.

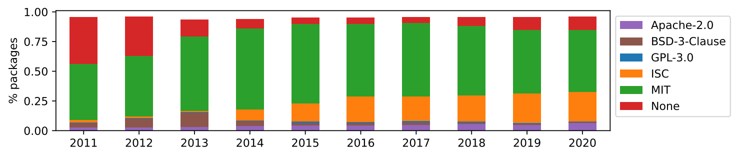


Fig. 2: Proportion of npm packages grouped by license and year of release.

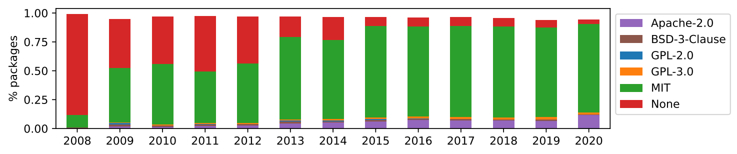


Fig. 3: Proportion of RubyGems packages grouped by license and year of release.

RQ_2: To which extent do packages rely on direct dependencies with incompatible licenses?

Developers have full control over which direct dependencies to incorporate. Thus, it should be easier for them to verify compliance with direct dependencies, compared to transitive dependencies. However, developers often have to rely on their own knowledge of legal compliance when declaring a license for their own software, allowing room for mistakes. Although tools exist to detect license incompatibilities, they are far from perfect. With RQ_2, we aim to answer to which extent incompatibilities are caused by direct dependencies.

After computing all $(dependent, dependency)$ license pairs, we found that the proportion of direct dependencies with incompatible licenses is low. We only found 32,323 (0.89%) and 11,121 (4.3%) npm and RubyGems dependencies to have licenses that are incompatible with those of their dependents. These incompatibilities are caused by 11,858 and 3,677 unique npm and RubyGems packages used as dependencies and are affecting 23,178 (3.1%) and 9,387 (9.9%) unique npm and RubyGems dependent packages. One could hypothesize that larger ecosystems with lots of dependencies have a higher chance of containing incompatibilities. Interestingly, the smaller ecosystem (RubyGems) seems to have a higher proportion of incompatible dependencies. A possible explanation for this phenomenon is that developers in large ecosystems may have more choices between packages with equivalent functionalities. This allows them to pick a dependency that fits their licensing needs.

Figure 4 shows the most common incompatible direct (*dependent*, *dependency*) license pairs for npm and RubyGems packages (Y-axis in log scale). As expected, most of the incompatibilities (i.e., 85% for npm and 76.7% for RubyGems) happen because of the fact that many packages used as dependencies do not have a license, i.e., (_ , *None*) while "_" refers to any license. For dependencies that have a license, we observe that the most common pair of incompatible licenses in npm is (MIT, GPL-3.0), i.e., when permissive MIT packages depend on strong-copyleft GPL packages. This is the epitome of a severe license violation. Beyond unfamiliarity with software licensing, there is no other conceivable reason for developers to incorporate these strong-copyleft-licensed packages directly into their permissive-licensed packages. Interestingly, after a manual inspection of the MIT packages with dependency license incompatibilities, we found that many of the maintainers (especially of the most popular packages) have decided to get rid of their GPL dependencies in the meantime. The second most common incompatible pairs are strong-copyleft GPLv2-licensed packages relying on permissive Apache-licensed projects. At first sight, this does not seem like it should be classified as an incompatibility since Apache is a permissive license. However, as explained before, Apache does impose some restrictions which render it incompatible with the GPLv2.

For RubyGems, we observe one incompatible pair of licenses (MIT, GPL-2.0) that heavily stands out from the other pairs. Again, unfamiliarity with software licensing is a plausible cause. However, the problem gets amplified because RubyGems has proportionally more GPLv2 packages than npm. Additionally, RubyGems is the smaller ecosystem, implying that developers might not always find a permissive-licensed alternative for their GPL-licensed dependency. With deeper inspection, we also noticed that some packages could also benefit from downgrading their license from GPLv3 to GPLv2. This is the case for GPLv3 packages that use GPLv2 dependencies, which is not allowed, but easily be remedied with a downgrade to either GPL-2.0 or GPL-2.0-or-later.

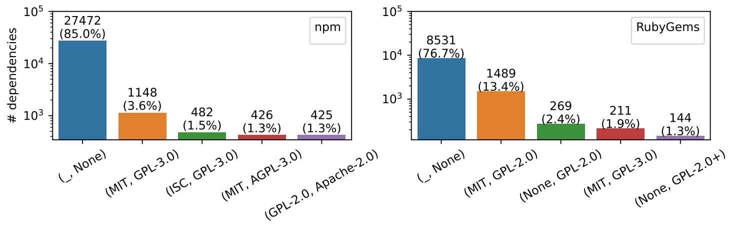


Fig. 4: Top 5 illegal license pairs (dependent, dependency).

RQ_3: How does license incompatibility spread across package dependency networks?

The previous research question focused solely on the direct dependencies, i.e., the dependencies over which the maintainer of a project has complete control. As mentioned before, the maintainer does not have control over which indirect dependencies are used by their project's (in)direct dependencies. Thus, it becomes harder to verify each of the transitive dependencies for potential license incompatibilities. This research question therefore aims to answer to which extent incompatibilities are caused by transitive dependencies. To this end, we will evaluate how prevalent incompatibilities are at each level in the dependency tree of a package. This provides insights into the difficulties maintainers face in keeping track of their transitive dependencies, and into the propagation of incompatibilities through the dependency networks formed by package repositories such as npm and RubyGems.

We start by comparing the proportions of incompatible license pairs between these repositories. We found that npm packages have more indirect dependencies with incompatible licenses than RubyGems. This is due to the high number of dependencies that npm packages include. The latter have license incompatibility with 58,388 indirect dependencies, while RubyGems packages have 8,562. However, RubyGems has proportionally more incompatible indirect dependencies than npm (i.e., 0.82% against 0.09%, respectively).

Figure 5 depicts the proportion of license incompatibilities at each level in the packages' dependency trees, including the first level (i.e., direct dependencies). We notice that for both package repositories, the proportion of incompatible dependencies decreases from one level to the next until the deepest levels where we see a rise again. The latter is more outspoken for RubyGems, where the proportion of incompatible dependencies is higher than 5% at the 8^{th} and 9^{th} levels. A closer inspection revealed that this is mainly because of the set of *metanorma-x* packages (e.g., *metanorma-gb and metanorma-vsd*) which have a BSD-2-Clause license but indirectly depend on the package *latex-decode* which has a GPL-3.0 license. Note that with these two licenses, it is legal to have a license pair (GPL-3.0, BSD-2-Clause), but not the other way around (BSD-2-Clause, GPL-3.0). Finally, we found that dependencies without a license constitute 86% of license incompatibilities with indirect dependencies for both package repositories.

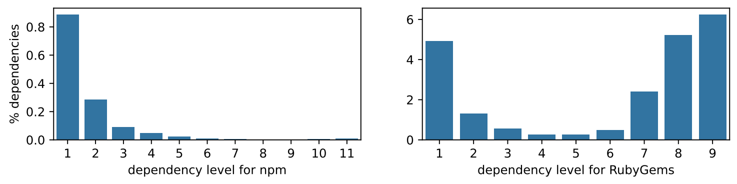


Fig. 5: The proportion of incompatibilities in each level in the dependency tree.

Figure 6 shows the most common illegal pairs of licenses of the format (*dependent, indirect dependency*) that were found at each level of the packages' dependency trees. We observe that the number of dependencies without a license decreases from one level to the next as we go deeper in both package repositories. This means that packages that can be found at deeper levels of the dependency tree usually have a license. One reason might be that deeper packages provide core functionalities and are thus more popular with a more mature management of legal issues such as licensing. We also observe that (MIT, GPL) is less common in indirect dependencies, compared to direct dependencies (See RQ_2). In fact, this is more relevant in the case of npm where we see that (GPL-2.0, Apache-2.0) is the most common incompatible license pair. Interestingly, the number of (MIT, GPL) pairs seems to gradually decrease from a dependency tree level to another while the second-most common illegal pair increases: (GPL-2.0, Apache-2.0). This means that the Apache-2.0 license is more common among the deeper levels. Consequently, direct incompatibilities with this Apache license are more likely in deeper levels.

One hypothesis explaining this phenomenon is that some core packages make use of the Apache-2.0 license. These would be packages that are important but are not regularly needed as direct dependencies, which is why they occur more as indirect dependencies. Such packages are more likely to be older packages which happened to be licensed under Apache 2.0 before the dominance increase of MIT and ISC licenses.

Having a closer look at the concrete packages that reside in the deeper levels, we indeed found that these are packages that provide core functionalities. Examples of these Apache-licensed packages at the deeper levels are *thread_safe* and *addressable* from RubyGems, and *oauth-sign, forever-agent, aws-sign2* and *caseless* from npm. These are popular packages downloaded millions of times and they were all released years ago. Moreover, some of them have not been updated for more than three years. In fact we noticed that most Apache-licensed packages that occur at the deeper levels in npm happen to be coming from the same organization. This is an interesting observation as it shows the influence that one single organization can have in a software ecosystem. By choosing the Apache license for all their packages that provide core functionalities, they unwillingly prevent other developers from releasing their software under the GPLv2. However, several package developers are unaware of this incompatibility or simply do not care, making (GPL, Apache) the most common illegal license pair.

5 Discussion

In RQ_1 we found that package repositories have indeed moved towards a more permissive climate. The MIT license, with its permissive and simple nature, has become the primary license of choice for many open source developers both in npm and RubyGems. An important factor that limits the license choice are the chosen third-party dependencies. If one must comply with the licenses of

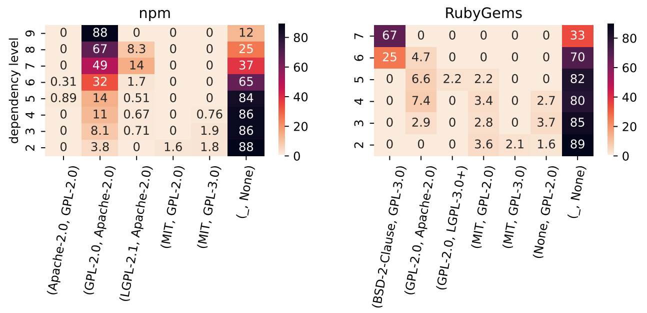


Fig. 6: Top illegal license pairs (dependent, indirect dependency) and their proportion in each level in the dependency tree.

every used dependency, the options become limited, especially when strong-copyleft licensed dependencies are being used. Furthermore, ecosystems that predominantly contain copyleft packages, cause all other packages that depend on them to also be copyleft. *WordPress* plugins [14] form a good example of an ecosystem in which copyleft licenses are popular. In contrast, we noticed that npm has few copyleft projects, possibly due to the high community appeal for npm packages and JavaScript libraries in general. This is also one of the observations already made in the FreeBSD ecosystem [5].

While having a look at the ecosystems separately in RQ_2, it became clear that both ecosystems are facing the same problems to some extent. The main problem is that some developers directly incorporate dependencies with strong-copyleft licenses or without any license into their permissive-licensed packages. This can be resolved by creating more awareness around this topic within the ecosystem that has formed around the package repositories. We did find, however, that in the meantime some developers had realized their mistakes and got rid of their GPL dependencies. However, the overall prevalence of license dependency incompatibility in package repositories can be easily reduced if packages without a license included one. We would recommend developers of such packages to choose a permissive license such as MIT and ISC, as most of the packages make use of permissive licenses. Another common incompatibility that occurs in both ecosystems is GPLv2 packages using Apache-licensed dependencies. This could be easily solved by upgrading the license from GPL-2.0 to GPL-3.0. One of the main issues that the GPLv3 tried to address was the compatibility with licenses such as Apache.

Given the complexity of assessing license compliance across all dependency tree levels, we hypothesized that most incompatibilities would be caused by licenses of indirect dependencies. After investigating this in RQ_3, we observed the exact opposite happening: the first levels cause many more incompatibilities than

[14] https://choosealicense.com/community/

the deeper ones, and this in both package repositories. Thus, in general the hypothesis cannot be accepted, although there were some cases where one type of incompatibility was more prominent at deeper levels than at the first levels (e.g., (BSD, GPL) in RubyGems). However, these observations seem to be tied to the practices within the ecosystem and its history rather than a mere consequence of their contributors' unfamiliarity with licensing. For example, we have seen that npm has a high concentration of Apache packages at the deeper levels. Similarly, it was mentioned how all ecosystems have a higher concentration of GPL packages at the first levels than at the deeper levels. Due to the viral nature of the GPL license, the concentration of GPL violations is a higher at the first levels than at the deeper ones.

Our findings have only shed a quantitative light on the prevalence and evolution of license incompatibilities across a package's dependency tree. Further research is needed to understand how developers perceive and resolve these issues in practice, as well as the kind of tool support they require.

Tooling: In order to support software developers, we developed a license compatibility tool that enables developers to cope with different license compatibility issues in their own software distributions. For npm packages, our tool uses a visual representation of a project's dependency tree to easily spot legal issues throughout the different levels of the dependency graph. Our tool is based on a GitHub project *anvaka/npmgraph.an*[15] that constructs a dependency graph for any given npm package using npm's API. Currently, our tool is limited to npm packages only but we are planning to extend it in the future to support other package repositories as well.

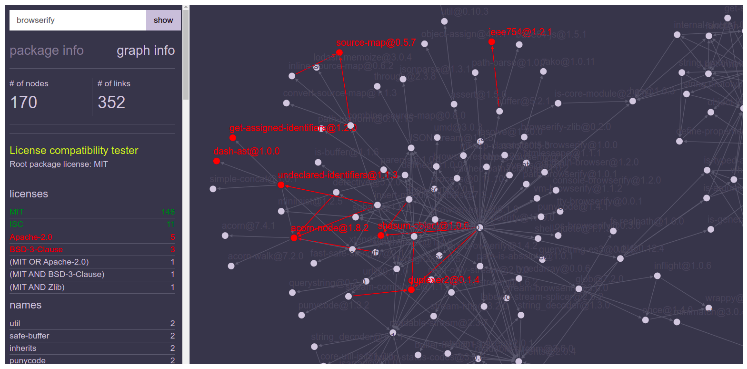
Using our tool [16], maintainers of npm packages can spot possible incompatibilities across all levels of their dependency graph. After activating the license incompatibility tester, the tool will highlight nodes and edges where severe incompatibility is identified with a red color (See Figure 7).

6 Threats to validity

The main threat to *construct validity* comes from imprecision in or incompleteness of the data sources we used to identify licenses. We assumed that the libraries.io dataset represents a sound and complete list of packages and their dependencies and licenses. In constructing our license compatibility matrix, we consulted various sources of information. Mistakes in these sources or our interpretation thereof might lead to false positives and false negatives in the compatibility of a license pair. Nevertheless, to mitigate these issues we have conducted manual inspections of samples of the data to verify their soundness.

As a threat to *conclusion validity*, to identify the license for a package, we only considered the license of the latest release of each package. It could be possible

[15] https://github.com/anvaka/npmgraph.an
[16] https://doi.org/10.5281/zenodo.5913761


browserify show

package info graph info

# of nodes	# of links
170	352

License compatibility tester
Root package license: MIT

licenses

MIT	148
ISC	11
Apache-2.0	5
BSD-3-Clause	3
(MIT OR Apache-2.0)	1
(MIT AND BSD-3-Clause)	1
(MIT AND Zlib)	1

names

util	2
safe-buffer	2
inherits	2
punycode	2

Fig. 7: Screenshot of the license compatibility checking tool.

that some packages have changed their licenses before. However, previous studies have shown that software projects do not usually change their licenses and when they do, they usually migrate to less restrictive licenses [12]. The latter finding means that only considering the license of the latest package releases may lead to an underestimation of the severity of license violations in dependency networks, since older releases of used packages are linked to more restrictive licenses.

Another threat to *conclusion validity* stems from the fact that whenever faced with uncertainty on the compatibility of a license pair, it was marked as "Unknown". Thus, only the most severe incompatibilities were marked as incompatible. Moreover, we only studied packages with a single license. However, we found that the proportion of packages with multiple licenses is small leading to a small proportion of dependencies without a compatibility status "Unknown".

We have also found many packages without a license, which led us to mark any license pair with the format (_ , *None*) as incompatible [17]. It could be possible that some of these packages without a license intentionally neglected to include a license, which means that our results here represent an overestimation with respect to license incompatibility with dependencies without a license.

7 Conclusion

This study evaluated the severity of license violations across transitive dependencies of npm and RubyGems packages. We hypothesized that due to the complexity of manually assessing compliance across all levels of a dependency tree, there would be more incompatibilities caused by deeper-level dependencies. Contrary to the hypothesis, this study has shown that deeper-level dependencies cause fewer incompatibilities than those at the shallow levels. Besides dependencies

[17] In (_ , *None*), "_" refers to dependent with any license and *None* refers to dependency without a license.

without a license, we found that GPL dependencies are the major cause for incompatibilities, and that they are more present in the first level of dependency trees. Furthermore, this study has also shown how a set of packages created by a single organization can influence an ecosystem when it consistently releases useful packages under a particular license. For future work, we plan to investigate the impact of the license preferences of an individual or organisation in OSS package repositories even further. Finally, to assist developers in detecting and resolving these issues, we have created a graphical tool out of the algorithm at the heart of our empirical study.

Acknowledgments

This research was partially funded by the Excellence of Science project 30446992 SECO-Assist financed by F.R.S.-FNRS and FWO-Vlaanderen.

References

1. Georgia M Kapitsaki, Frederik Kramer, and Nikolaos D Tselikas. Automating the license compatibility process in open source software with spdx. *Journal of Systems and Software*, 131:386–401, 2017.
2. GR Gangadharan, Vincenzo DAndrea, Stefano De Paoli, and Michael Weiss. Managing license compliance in free and open source software development. *Information Systems Frontiers*, 14(2):143–154, 2012.
3. David A. Wheeler. The free-libre / open source software (floss) license slide, September 2007.
4. Georgia M Kapitsaki, Nikolaos D Tselikas, and Ioannis E Foukarakis. An insight into license tools for open source software systems. *Journal of Systems and Software*, 102:72–87, 2015.
5. Maria Kechagia, Diomidis Spinellis, and Stephanos Androutsellis-Theotokis. Open source licensing across package dependencies. In *2010 14th Panhellenic Conference on Informatics*, pages 27–32. IEEE, 2010.
6. Shi Qiu, Daniel M German, and Katsuro Inoue. Empirical study on dependency-related license violation in the javascript package ecosystem. *Journal of Information Processing*, 29:296–304, 2021.
7. Alexandre Decan and Tom Mens. What do package dependencies tell us about semantic versioning? *IEEE Transactions on Software Engineering*, 2019.
8. Sivan Michaeli. Top 10 open source software licenses of 2016 and key trends. https://resources.whitesourcesoftware.com/blog-whitesource/top-10-open-source-software-licenses-of-2016-and-key-trends, January 2017.
9. Ben Balter. Open source license usage on github.com. https://github.blog/2015-03-09-open-source-license-usage-on-github-com/, March 2015.
10. Openbsd copyright policy.
11. Brian Reid. Kea to be released under mozilla public license 2.0, December 2015.
12. Christopher Vendome, Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Daniel German, and Denys Poshyvanyk. License usage and changes: a large-scale study of java projects on github. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 218–228. IEEE, 2015.