MAYER | BROWN

Podcast

Tech Talks

Navigating Open Source Pitfalls in Transactions

December 15, 2022

Announcer
Welcome to Mayer Brown's Tech Talks Podcast. Each podcast is designed to provide insights on legal issues relating to Technology & IP Transactions and keep you up to date on the latest trends in growth and innovation, digital transformation, IP and data monetization and operational improvement by drawing on the perspectives of practitioners who have executed technology and IP transactions around the world. You can subscribe to the show on all major podcasting platforms. We hope you enjoy the program.

Julian Dibbell
Hello and welcome to Tech Talks. Our topic today is "Navigating Open Source Pitfalls in Transactions." I'm your host, Julian Dibbell. I'm a senior associate in Mayer Brown's Technology & IP Transactions practice. I'm joined today by my colleague Paul Chandler. Paul is a partner in Mayer Brown's Technology & IP transaction practice. But more than that, he is, for many of us in this practice group, the go-to person for open source software questions, the person we turn to whenever there is a particularly knotty open source software issue in a deal. So I'm really looking forward to this conversation today, and I'd like to start by just asking you, Paul, how did you become interested in open source software?

Paul Chandler
Thank you, Julian. I started seeing open source in deals in the early 2000s, and at that time it seemed like a bit of a niche. It definitely wasn't mainstream, and in part because open source wasn't widely used at that time and there was even debate about how significant the legal issues were, given that this was free or no-cost software. There was then a lack of open source case law. My interest in open source developed as we started to see it in an increasing number of tech deals and clients asked for help analyzing licenses, developing open source policies and negotiating contract terms for open source. In particular, I was really intrigued by the complexity of open source licenses, which often involve an intricate mix of legal and technology questions. That led me to look for ways to effectively mitigate open source risks for clients.

Julian Dibbell
Okay. You've touched on a lot of interesting elements here, and maybe before we dive in, we should unpack some of those and talk a little bit about what open source software. There's no precise definition. Open source software is generally understood to refer to both a software development model and a software licensing model. As a development model, open source

software refers to software created by crowdsourcing the work to a group of unpaid volunteer programmers organized into a project. The individual programmers contributed code to the overall project, and proponents say that this promotes innovation and collaboration among programmers. This is, in part, because it allows many eyes on the code to find bugs and add improvements.

However, the development model is premised on the licensing model, right? Because for the licensing model there are these two key features. First of all, open source software is software that's made available to the public over the internet in source-code form, which is what allows this ease of collaboration. And then the user is granted a license that permits modification and distribution without payment of any fee to the licenser, obviously removing a big speed bump in the way of any kind of spontaneous collaborative project. You contrast this with most commercial software licenses that only permit use of software in executable, compiled form with no access to the source code and consequently prohibiting users from modifying or distributing the software. This open source model just removes all the obstacles to working with software and sounds very straightforward, and simple. What are the risks here, Paul?

Paul Chandler
Well, it does sound simple, “free” software, what could go wrong? It turns out open source is a lot more complicated than most people think.

Let's talk about the risks first. There are what I call garden variety risks. The software is downloaded from the internet for free without any of the typical licensee protections that you'd have in negotiated commercial software license agreement. For example, the software is as-is without any IP infringement indemnity and no commitment for support or maintenance. All of those risks are borne by the user or by the company that employs the user.

You could look to the open source project for ongoing support, but the project can shut down at any time and leave the code orphaned. So you need to get comfortable with those risks.

In addition to the garden variety risks, there are also risks that are unique to open source. These risks generally fall into two categories, license compliance and security vulnerabilities.

Julian Dibbell
Okay, again, lots to unpack there. I think if you've heard anything about risks related to open source software, it's probably the license compliance risk, right? So let's talk about that first.

Paul Chandler
Sure. So free may mean no cost, but it doesn't mean you could do whatever you want. Open source software is not public domain software. There are license terms that users need to comply with, and there are many different open source licenses in circulation. So what are the key things you need to remember?

Well, first, open source licenses vary widely, but they generally fall into two groups, permissive and copyleft, and I think of this in terms of easy compliance and hard, or more complicated, compliance. Permissive open source licenses, such as the BSD or MIT license, are relatively

simple. You can modify and distribute the code as much as you want, so long as you include notice of license conditions and attribution to the licensor. Notice and attribution.

Copyleft licenses such as the GNU Public License, also called the GPL, on the other hand, require that if you distribute the code or any work based on the code, you must provide recipients the source code under the same copyleft license terms that you received that source code under—with the same requirement applying to all downstream recipients of the code or any work based on the code. I just said the license will apply to distributed works based on the copyleft code, but what that means varies by license. The requirement that the license apply to or infest other software based on the code, is why these licenses are also sometimes referred to as viral. Proponents of copyleft licenses don't favor that moniker, by the way. As you could imagine, just keep going with this metaphor, this infection can have major implications for companies. For example, depending on how you integrate open source copyleft code with your proprietary code, the license may require you to (1) distribute the source code for your proprietary code; (2) not charge a license fee or impose your own license terms for your proprietary code; and (3) grant to the public a license to your entire software patent portfolio that might read on what you're distributing. These requirements typically get triggered by distributing open source code. However, it's not clear what that means, in many cases. Many popular open source licenses did not define key terms or even specify what the governing law is, so that creates uncertainty about what compliance means. Even if you're not distributing the code, even if you're only using it in a SaaS or cloud-hosted model, some open source licenses have triggered copyleft requirements based on that.

And finally, some companies try to engineer their code to carefully avoid copyleft effects, but as we know, software evolves and changes as it's used, and those copyleft effects may get triggered later on down the line as the way the code is engineered changes.

Julian Dibbell

Okay, so there are a variety of different kinds of open source licenses and they all have different terms, some more permissive, some less. So that gives you options, and on the other hand, you may be using many of these together. Is it possible for the terms to be in conflict? How would that happen and what are the implications for the licensee?

Paul Chandler

Yes, that's a big point. Yes, conflicts are possible, and it's another known risk. This happens when an application leverages multiple open source packages for the project and the packages are covered by incompatible open source licenses. What do I mean by that? Well, remember I mentioned earlier that a common requirement of a permissive license is notice and attribution, but a common requirement of copyleft license is that "thou shalt impose no other license terms," so on the one hand, you have a license that says you must give notice and attribution. On the other hand, you have another license that says only its terms apply, and that's a conflict.

The important thing to remember is that this is incompatibility in the *legal sense*, not in the technical sense. The components in question may operate together just fine, and that means that the technical people, the programmers, and developers will not be aware necessarily of the

conflict, but the legal department sure will if it looks into this. You also need to keep in mind that this can be a multidimensional problem. What do I mean by that?

Well, one dimension covers conflicts that occur between open source packages used in a project, the ones that the customer approves for developers to use. But it's also usually the case that open source packages themselves rely on other open source packages to operate, and those other open source packages can be referred to as dependencies, and those packages, in turn, have their own dependencies and open source packages that they rely on. So software license conflicts can arise from dependencies at any level within a package, and as you can imagine, this can get quite complicated. So what's the worst case? Well, the worst case here is that an open source license conflict will prevent the client from selling its product as it intended, or the client may incur substantial remediation costs to fix the problem down the road when it's discovered.

Julian Dibbell
Well, so that's a pretty bad worst case and we've just gotten through the licensing risks, and of course, there are more as you mentioned. Let's talk about some of those. Security vulnerabilities I think you mentioned. What's going on there?

Paul Chandler
Well, security is a big item in the news these days. It seems like we're reading about hacks every day there's a new system that's hacked and there are a number of issues to consider. So as you said earlier, the source code for open source is available for anyone to study, including hackers. Remember, Julian, you said earlier, many eyes on the code. So hackers may find security vulnerabilities and for packages with wide adoption that means hackers can exploit these vulnerabilities to engage in large-scale cyberattacks.

Also Julian, like you said, open source is not just the licensing model; it's a development model, and that means that the model involves trusting complete strangers to make positive contributions to the software as its developed, and any time there's a need for trust and the digital world, there's an associated vulnerability. Open source is downloaded from the internet. Anyone can download the package, sabotage the files, and put them back on the internet for someone else to download, and this can take many forms. Someone could switch the license terms (for example, replacing a copyleft license with a permissive license, which would mean that the user wouldn't be complying with the right license terms). Another way this can come about is programmed developers can sabotage their own code. This isn't just a theoretical risk; it's happened repeatedly. We've seen people put malware into open source to protest, for example, Russia's invasion of Ukraine. We've also seen the creator of a very popular open source packages, Fakers and Colors, sabotage his own code because he became unhappy that corporations were using it for free.

Then there's what is sometimes referred to as typosquatting. Here, a hacker downloads the code, injects malware into it, and puts it back on the internet for others to download, but with a slightly modified name to trick careless developers. So, for example, instead of a package called **J**ulian.js it may be Ju**ll**ian.js or **j**ulian.js. And of course, the hacker isn't just necessarily doing this

once. It may create a multitude of packages with lookalike names to fool unsuspecting developers. This is a significant problem today, in part because we don't have the equivalent of an iTunes store for open source, where a central authority reviews and verifies what's available for download.

Julian Dibbell
Yet with all of these risks, as scary as they may be, open source software is not hardly confined to the margins of software production these days. It's mainstream; it's literally woven into the fabric of modern technology in our lives. I think a recent study found that something like 70% of software products were found to have pieces of open source software incorporated into them.

So with all the risk, it's hard for businesses to resist the advantages of using open source software because those go straight to the bottom line—the low or no licensing costs, the basically free labor you get out of the community of programmers. So it's hard to resist, and we see more and more deals where open source software is in the mix.

Let's talk about those deals. How do open source software issues arise in the context of technology transactions? And maybe let me start by asking Paul, what kinds of deals are we talking about? Where are we most likely to see questions about open source software coming up?

Paul Chandler
Well, we see these issues to some extent in M&A and even in standard software licensing agreements—by standard I mean commercial. But, open source issues are usually implicated most directly in software development deals. A client will come to us and say "we want X software developer by Y developer for Z dollars." But, just because there may be an agreement on price and scope, that doesn't mean you're done. And sometimes this happens in dramatic fashion. For example, we had a client whose business folks had agreed on price and general function for a large digital platform development, several hundreds of millions of dollars in costs, and they were quite surprised as the deal went on to find out that the developer planned to build the platform mostly out of no-cost open source software. So naturally there is a question with them: Why are you charging me hundreds of millions of dollars?

Julian Dibbell
As lawyers, then, how do we help our clients approach these deals?

Paul Chandler
Well, these days, almost every development deal we see involves open source to some degree, so we know it's going to be an issue even if the client doesn't initially mention it. Lawyers can best protect their clients by understanding the open source risks themselves and then educating their clients on the materiality of the issues and getting involved early in the deal formation process before open source selections become solidified. This is the lawyer's best chance to educate the client on open source risks before negotiations start, and, in that way, it's less likely that these issues will be dropped in the heat of the battle.

Julian Dibbell
Important point. And, in educating our client as you suggest, what are the tendencies that you see that we need to be pushing back on and pushing them away from, like what are the some of the most common open source software misconceptions and missteps that you have seen in your deals?

Paul Chandler
We see a number of misconceptions repeatedly come up and go down. A quick summary of them: One is, hey, it's free. There's no worries. The client looks only at the cost savings for the initial up-front cost but not at the full cost of ownership. And, by that, I mean the cost of license compliance, which can be a real headache, and also providing long-term support. The clients minimize the risk because they don't fully understand the risks or they focus on the lack of open source litigation at this point, but they focus only on the particular software license but not on code quality or the security vulnerabilities in the code. That's being too narrow-minded. Or, they think of open source in terms of a one-size-fits-all solution. For instance, they might say, "well, we've always dealt with open source this way," and they don't focus on what the particular project is intended to do, what its objectives are. Finally, we see a lot of clients that have the attitude of "set it and forget it." They don't put in place some kind of ongoing monitoring of open source use as development progresses, and that leaves them open to risks, because there wouldn't be any process to pick up new uses of open source or engineering changes that could impact license compliance issues, among other issues.

Julian Dibbell
Okay, so how do we translate this into actual legal advice? Diving into the process of the deal itself, what questions do we need to start with?

Paul Chandler
We, as lawyers, need to understand what the client gets for its money. Is it getting custom code, open source or something else? And, in a development deal, preliminary due diligence is always key, but now it's even more so with open source. There is no one-size-fits-all approach. You need to know what you're dealing with, and good questions bring out the issues, but they also allow you to avoid problems and propose appropriate legal terms or mitigating strategies.

For example, while there are a lot of questions that could be asked, here are a few key ones: Why are we doing this? What is the purpose of the project? Is it to develop back office software, a product to be distributed? Is it a fee-based commercial product? Is intellectual property important? This is probably the most important question to ask, because, as I mentioned, open source license requirements are more likely to be triggered by distributed software than those used for back office software, for instance.

Another question is, where will the software be run? Is it internal? Will it be cloud-based all or in part? Will it run on third-party outsource computers? How will it be maintained? What's the plan for it to be supported, and also who are the parties involved? Does the client have a significant patent portfolio? Do each of the parties understand open source risks and issues? Is there sophistication there? Do they have open source policies with "go" and "no go" lists for licenses?

The answers to these questions will let you find potential problems and look for solutions. For instance, there may be a different license that could be used, or you could switch to a different package, but you won't have the potential to find the solution if you don't know that the problem exists.

Julian Dibbell
Alright, so then diving into the negotiations themselves, what are the contractual approaches that customers should be considering for mitigating open source software risks in their software development agreements?

Paul Chandler
This is a really interesting area, and I'd say it's one of the most evolving aspects of open source legal work. Providers and customers are getting a lot better at understanding open source risks, but, of course, the resulting contractual terms vary depending on the parties, the nature of the deal and which party chooses the open source in question, in my experience. In the past, the customer would often say absolutely no open source can be used on this project without my consent, or they might have simply forgotten or ignored the issue altogether. Similarly, providers often would say any open source is provided, as is our warranties and indemnities don't cover open source, and we're not including open source in our maintenance and support obligations. Now, those positions are often not realistic, in part because open source is everywhere and, in many cases, the providers' competitors are giving protection. These days the terms in contracts regarding open source are much more nuanced to manage risk.

There can be a preapproved list of open source and uses, and I'd say more often than not customary warranties of compliance and indemnities do not explicitly exclude open source software, but it really depends on the parties on the deal and, and you know, depends on what's appropriate for the parties at the time. We are also seeing more open source-specific contractual provisions come into play. For instance, an obligation for the provider to provide information rights (e.g. what open source packages are used, what versions, how they're used) and that is very important many times to facilitate the clients being able to comply with open source licenses that are relevant. We talk to the customer and the provider as clients and look for what's appropriate in a given deal.

Julian Dibbell
Okay. So you're talking about our standard contractual provisions for managing risk warranties and divinities, et cetera. Are contractual solutions always just about those kinds of risk allocation provisions, I mean, what else do you see?

Paul Chandler
It's not always the case. We've seen some pretty creative practical approaches to deal with open source issues. For example, we had a client that was trying to sell a software package that included copyleft open source, and, in order to avoid an argument that our client distributed the copyleft code and, with it, all the integrated proprietary code, our client persuaded the buyer that it was in the interest of both parties for our clients to transfer only the proprietary system code and let the buyer itself download the open source copyleft code and rebuild the product.

Doing that also avoided an automatic grant of a patent license. There was skepticism at first, but, when people understood the open source issues at hand, they were very in favor of a creative way to avoid them.

Julian Dibbell
That's a great point and shows the importance of both understanding the issues and then thinking creatively about them. I want come back to a topic you hinted at earlier, Paul, the question of Providence. Do you know where this software came from and the risks that arise from that? I mean that's becoming something of a hot topic, isn't it?

Paul Chandler
Yes, it's a hot topic, and it's a deep topic, and the issue is twofold. First, you have the problem of making sure the open source packages were downloaded from the official project website, not just from the equivalent of an internet bulletin board.

But, even if the developer downloaded the open source from an authorized location, there's no guarantee that what was downloaded was the open source package the developer intended. Remember, we talked about typosquatting earlier and the risk of developers being tricked into downloading lookalike packages. And, just like with license conflicts, this is a multidimensional problem. Mistakes and mis-downloading can occur with packages that the client approves for the project, but mistakes can also occur with open source packages at any level all the way down the dependency chain. So, as you said, this is currently a hot topic, and companies are looking for ways to secure the software supply chain. There's also a drive to do this because of the recent executive order on software bill of materials, so-called SBOMs. We expect to see a lot of activity in this area as things evolve. So far, we're not seeing it come up and deals as much, aside from perhaps contracts that involve federal government agencies and aside from a general requirement for developers to provide SBOMs with their deliverables.

Julian Dibbell
Paul, finally, I have to ask, how do you see the role of outside counsel in addressing open source software issues in connection with the deal? What can we do to help?

Paul Chandler
I think that's a really important question to understand how you can best utilize us or outside counsel in general. We can answer questions for clients and help clients educate their business clients and stakeholders in understanding open source issues. We can help analyze the proposed open source licenses for a client's project, identify and assess risks, and propose ways to mitigate those risks, for instance.

Depending on the project, mitigation might be not using the open source package in question for your project. That might mean that you use a commercial or a proprietary alternative instead, or it might mean looking for an alternative license for the open source or asking the developer of the open source for an exemption. Or, it might mean engineering the project so that you don't distribute that particular open source. Or, as I alluded to earlier, we have the user download the package to avoid the issues. We could also propose contractual protections that appropriately allocate risk between the parties. Finally, when the contractual protections don't

cover everything, we can identify and assess residual risks from the open source usage. For instance, the risk of typosquatting, dependencies in security vulnerabilities and how to address those risks might mean through monitoring and management of open source, use of open source policies, and coordinating with the company's security vulnerability policies, among others.

Julian Dibbell
Thank you, Paul. Great insights. Listeners, if you have any questions about today's episode or an idea for an episode you'd like to hear about, anything related to technology and IP transactions and the law, please email us at techtransactions@mayerbrown.com. Thanks for listening.

Announcer
We hope you enjoyed this program. You can subscribe on all major podcasting platforms. To learn about other Mayer Brown audio programming, visit mayerbrown.com/podcasts. Thanks for listening.