eBook

Using Open Source in Your Code Base: A Beginner's Guide

Checkmarx

Table of Contents

Introduction

There are excellent reasons to incorporate open source code into your organization's own applications. It's usually free. It can help you implement functionality faster than you could if your developers had to write all of the code from scratch. It's often supported by a flourishing open source community that is constantly updating and improving the code.

Yet open source code also poses significant security, compliance, and other risks. While these risks can certainly be managed, some organizations aren't even fully aware that they exist, let alone prepared to respond to them.

If your organization uses open source code in any way, this eBook is for you. The following chapters explain what developers and business stakeholders need to know about incorporating open source into their own applications in order to do so securely.

As you'll see, virtually every organization can – and in many cases should – take advantage of open source as a way to build out its codebase. But they must implement safeguards that allow them to manage the risks that come with open source, and ensure that those risks are outweighed by the benefits that open source offers.

Open Source Is Everywhere – Even Your Codebase

To understand the risks associated with open source code, you must start by understanding just how widespread open source code is, even within organizations that don't think of themselves as participants in the open source community.

In his 2011 article for The Wall Street Journal, Marc Andreessen wrote that "**software is eating the world**." This, he explained, is because of the amazing pace of innovation in the tech industry, which is due in no small part to the open source movement. Open source has grown up alongside the rest of the technology market, and it is the reason why so many providers can create such rich services at such low costs. In the 10 years since he wrote his famous catch phrase, the industry has embraced open source to an even greater degree. And even the largest and oldest companies that once called open source "**a cancer**" have now **fully embraced the movement** – including Microsoft.

Where Did Open Source Come From?

Open source grew out of the free software movement, which can trace its roots back to the innovative culture that surrounded electronic and computer hobbyists in the 1970s. This was the same era that gave birth to many technology giants, including Apple and Microsoft. The free software movement was a rebellion against those giants who sought to control their software and limit the rights of its users to modify and redistribute the software as they saw fit. The "free" in the "free software movement" wasn't about offering products free of cost – it was about "free" as in freedom. In other words, you could charge for your program, but your source code had to go along with it, and the end users could do with it as they pleased.

The realities of "free software" didn't always align well with the ideology of the **Free Software Foundation**, however, and many wanted to remove the focus (implied or real) away from cost entirely. In 1998, Christine Peterson coined the term "open source" to place the emphasis on source code instead, and to reify that users should have the freedom to run, modify, and share software. Since 1998, the open source movement has far surpassed the FSF in size and influence, and the sheer number of **open source organizations** is testament to that fact. For a project to be truly open source, it must use one of the approved open source licenses maintained by the **Open Source Institute**.

Just How Prevalent Are Open Source Projects in the Real World?

To get an idea of just how prevalent open source is (and how it continues to move into spaces traditionally dominated by proprietary software), look no further than the web browser you're using right now. Firefox is open source. Google Chrome and Microsoft Edge are open source. Even Opera and Safari are based on open source projects.

Web browsers are also great examples of single open source projects that can be supported by multiple companies shipping their own products. For instance, Chromium is an open source project that was started by Google to be the base for Chrome. Chromium is now used at the core of Opera, Microsoft Edge, and dozens of other web browsers you'll likely never hear about.

Open source also dominates at the infrastructure layer of the entire technology industry, **with over 70% of web facing computers running open source web servers**. And you can't forget the poster child for open source, the Linux kernel, which is currently deployed on billions of devices world wide – and that's just Android smartphones. That doesn't even count its dominant position in the millions of servers at organizations like Facebook.

But What About Actual Developer Usage of Open Source for In-House Apps?

You will rarely find true developers who would not love to have the time to build everything that their applications need to function properly. The code that they build is a source of pride. Unfortunately, the reality is that developers are hired by businesses to solve problems, not to build their utopias. So whenever functionality is available that will fulfill their needs, they will use it. This can range from solutions like Drupal (a web content management platform that frees developers to focus on content and integrations) to development frameworks like Spring (which allows a developer to start coding business functionality instead of "plumbing"). In most cases, though, developers use a library to add desired functionality, such as adding a pre-built calculator to a website.

How Is Open Source Used and Distributed?

There are many different ways in which open source projects can be consumed. The following are some of the most common ways that they are introduced by developers, along with a brief description of how they flow through a DevOps or NoOps pipeline.

Source code is always available if developers want to build a component on their own, but in most cases, they download a pre-assembled component that is ready for use.

- **Module:** A module is the basic building block of any application. It is usually a single file that contains multiple functions. A calculator would be a great example. In this case, the functions would be: add, subtract, multiply, and divide.
- **Class:** A class is a module that can be instantiated into an object. This means that it can keep state on its own. One example of this would be a calculator that can keep a running subtotal.
- **Package:** This can mean different things in different contexts, but in many cases, it just refers to the component as you downloaded it. Strictly speaking (especially for languages like Python and Java), a package is a collection of modules or classes that share the same name space. In Python, packages are hosted in a single directory with a shared __init__.py file, for example, which ensures that they all start in a consistent way.
- **Library:** Libraries are probably the most common way in which application developers leverage open source projects. A library is composed of modules, classes, and even packages that together create a group of complementary functionality. A great example of this is jQuery, which is used by 73% of JavaScript-based websites. This simplifies common functionality, such as event handling and loading data from external services.
- **Framework:** At their core, frameworks are simply collections of libraries that have been pieced together to provide a starting point for developers. Frameworks usually rely on libraries from multiple other open source projects, and they are often opinionated, meaning that they want things done in a specific way. As long as what you need to do falls within those preset guidelines, everything is fine. The reason why so many frameworks exist is that different organizations have different styles, and not everyone likes to work the same way. For example, the top JavaScript frameworks all originated from different companies: Bootstrap came from Twitter, React came from Facebook, and both Angular and Polymer came from Google.

Open Source and You

If your company does any in-house software development – meaning you build your own applications, integrations, websites, or any other kind of software resources – it's nearly certain that you are utilizing at least some open source code.

So you need to answer a few questions: Do you have an inventory of open source code? Do you know which versions of which open source components you use? Do those versions have known security vulnerabilities? Which software licenses are you including, and what kind of exposure does that create for you? The GPL and Apache licenses (two of the most popular open source licenses) can have very different impacts on how much of your own code may end up being open source if you ship applications to consumers.

All of these questions can be answered by using a software composition analysis (SCA) tool, which can identify, categorize, and report on the open source code that has been imported by your development team. Introducing SCA functionality into your DevOps toolbox will enable you to flag known critical security defects and then use those as part of the deployment criteria for production. An SCA tool like Checkmarx's could save your organization from hitting the news cycle for the wrong reasons.

What, How, and Where Open Source Gets Pulled into a Codebase

Now that you know how pervasive open source is, let's dive deeper into how developers often incorporate open source code into applications that they are building.

The vast majority of software developers in the industry today are paid to solve business problems. Regardless of whether they work for small independent software vendors or Fortune 500 companies, solving such problems is now one of their primary responsibilities. Given the time and the opportunity, many software developers would write as much functionality into their applications as they possibly could from scratch. However, that can be very time consuming: first, they have to debug and fix it, and then, they have to maintain it (or better yet, enhance it).

Third-Party Extensions Are the Answer

To increase productivity and save a great deal of time, developers often use code written by third parties rather than rebuilding the same generic functionality across multiple applications. While there has always been a market for commercially licensed and supported extensions (including modules, packages, libraries, and frameworks), the vast majority of the third-party code used today is open source. This means that there is no marketplace and no purchase order; rather, a few extra lines of someone else's code are simply imported into the software.

This can cause problems with licensing and disclosure if it is not accurately tracked and monitored. That is why software composition analysis (SCA) products are worth their weight in gold. These solutions find all of the third-party packages that are in use, then identify the corresponding licenses. They can even show if they are out of date or if known security vulnerabilities have been reported against them.

It's All Based on Open Source

Even the lowest level of an application stack (the language and runtime engine) is often open source. The most popular languages in use these days are all open source, or at least have open source distributions. Go, Python, PHP, Ruby, and JavaScript are all open source by default, and even languages that are traditionally commercially supported have open source distributions like OpenJDK (for Java) and gcc (for C/C++).

After you've chosen your language, you'll likely want to ensure that you have some structure in place so that you won't need to declare all the basic functionality like dependency management and data management. Well over half of all Java applications use the Spring framework as their starting point. PHP uses Laravel, while JavaScript uses React and Bootstrap, among others.

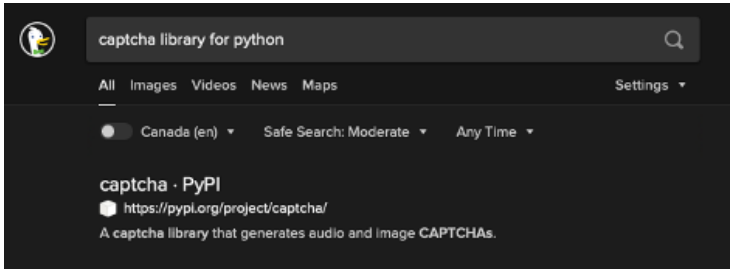
Frameworks and languages form a solid foundation for any application, but the bulk of open source influence can be found in the staggering number of modules that are available as packages and libraries which can be easily integrated into applications.

How Easy Is It to Find Open Source Modules and Libraries?

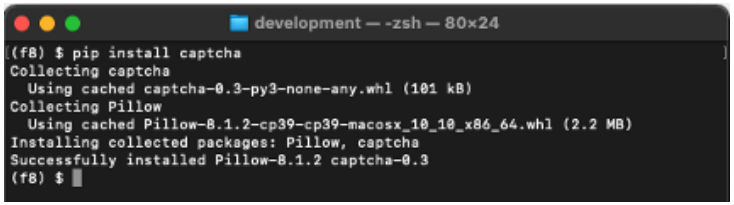
Any web search for any type of functionality will often return results that link to places like GitHub, GitLab, PyPI, and many other sites. So how do you find what you need?

Let's say that you want to make a particular form a little more secure by including a CAPTCHA. If you don't know where to start, just head over to your favorite search engine and enter, "captcha library for Python."

In our case, the first result is an open source library that can be installed via pip (pip is the standard utility used in the Python ecosystem to install modules).



Installing this module is as simple as typing, "pip install captcha."



Now, with just a couple lines of code, a whole new set of tested and proven functionality is added to the application in minutes.

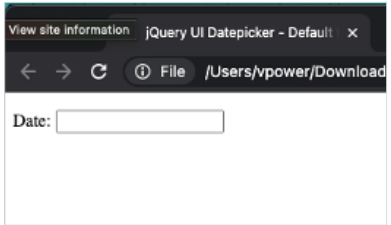
```
from captcha.image import ImageCaptcha
image = ImageCaptcha(fonts=['/path/A.ttf', '/path/B.ttf'])
data = image.generate('1234')
image.write('1234', 'out.png')
```

For another, more real-world example, let's say that you have a web application that needs to be able to pick a date from a calendar. To show you how to do this, we will use the jQuery library, which has a great deal of functionality and is easy to use.

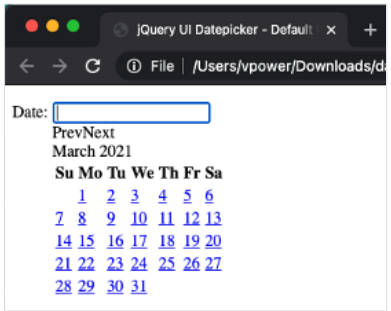
The first step is to add the jQuery modules to the web page in question. There are two stylesheets and two script files that need to be imported. These are added between the head tags. The next step is to define the datapicker function, which activates the appropriate pieces of the jQuery library. The final step is to define where to put it on the page using an input field. The code looks like this:

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <title>jQuery UI Datepicker - Default functionality</title>
 <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/
 jquery-ui.css">
 <link rel="stylesheet" href="/resources/demos/style.css">
 <script src="https://code.jquery.com/jquery-1.12.4.js"></script>
 <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
 <script>
 $( function() {
   $( "#datepicker" ).datepicker();
 } );
 </script>
</head>
<body>
<p>Date: <input type="text" id="datepicker"></p>
</body>
</html>
```

The finished web page looks like this:



When you select the date input, it will present a calendar. You can stylize it, of course, but this example shows the simplicity that open source libraries can provide:



As this example shows, it's very easy to add open source to your codebase. And it's very likely that your developers are using open source in ways like this, even if they don't otherwise make use of open source applications.

As we'll see later, even basic and minimal use of open source code introduces potential risks. Staying ahead of them requires visibility into how and where your organization borrows from open source projects.

Why Developers Use Open Source in Their Projects – and How to Manage the Risks

The previous two chapters explained how developers add open source to their projects, but they only touched briefly on why developers do this. This chapter fills that gap by focusing on the why of using open source code, and explaining when it does and doesn't make sense to do so.

If you're a developer, incorporating open source code into your project is like ordering a meal kit instead of cooking from scratch. It saves you some time and effort. But it also reduces your level of control over the final product, and it could lead to issues that you don't foresee.

That's not to say you shouldn't use open source (or, for that matter, meal kits). There are lots of great reasons for developers to take advantage of open source within their projects. But it's important to be aware of the potential drawbacks and risks, and to have a plan in place for addressing them.

Why Add Open Source to Your Codebase?

Most organizations with in-house development teams maintain their own code bases. Their developers write most of the code for those codebases themselves.

However, they may choose to add third-party open source code to their codebases, for several reasons.

The most obvious is that it's often faster and easier to incorporate a feature or build an integration using third-party code than it is to write it yourself. Why spend a week building out a new service from scratch when you can grab the code you need from an open repo on GitHub? Why reinvent the wheel if someone already invented it for you and wants you to use it for free?

Incorporating open source can also be handy in situations where an in-house development team doesn't have the expertise necessary to implement a certain feature or service itself. Maybe you need to add a new feature to a legacy application written in a language that no one knows well, for example, so you decide to pull the code you need from an open source project instead of teaching yourself the language.

Developers may also choose to work with open source code for reasons that could be described as political. Borrowing code from third-party open source projects can help build bridges with those projects. Contributing code back builds even stronger bridges. If a company wants to establish itself as a participant in the ecosystem surrounding a certain open source project, in other words, working with its code can be a way to do so.

The Drawbacks of Using Open Source Code

While the reasons for adding open source code to a company's internal code base are sound, there are potential drawbacks to be aware of.

One is that, to **quote engineer Steve Belovarich**, "when you adopt open source tools you often don't know what you are getting."

In other words, when you use open source code, you may understand approximately what it does and how it fits within your codebase. But unless developers spend days poring over each line of the code – which is unlikely because that would defeat the purpose of borrowing code in order to save time and effort – the team won't fully understand exactly how the open source code works. That's a risk, because it means the business becomes dependent on something that its developers don't fully understand.

Reusing open source code may also lead to licensing problems. There are dozens of different open source licenses, all of which have their own rules on how open source can be incorporated into other codebases. Some licenses let developers do anything they want with open source code. Others require attribution of the original authors of the code. Some impose restrictions in situations where the code is used publicly, but not if organizations only use it internally.

Because of this complexity, it's easy to run into licensing issues by reusing open source code in a way that violates whichever license governs it. And while it can be easy to assume that the licenses won't really be enforced, that's **not always the case**. Businesses face a real risk if they borrow open source code willy-nilly without understanding the licensing terms, and without keeping track of where the code exists within their own codebases.

Likewise, security issues can be a concern when using open source code. Open source is no less inherently secure or insecure than proprietary code. But when developers use open source without fully understanding how the code works, and without knowing which vulnerabilities may be lurking within it, they risk introducing security problems into their codebases.

When and When Not to Use Open Source Code

Deciding whether or not to make use of open source code, then, boils down to assessing whether the benefits outweigh the risks – as well as your organization's ability to control those risks.

If adding open source to your codebase will save a significant amount of time, effort, and money by significantly reducing the amount of original coding the development team has to perform, it's probably worth it. It may be less so if it will only save developers from a day's work.

Likewise, if you trust the source of the code and you're confident that it is secure, it makes more sense to lean on it. It's generally better to use open source code from a major project than it is to pull code from a random GitHub repository.

Most important of all is the level of visibility and control that developers have over the source code they use. Regardless of how much they trust the original authors of the code or how well written the code is, it's impossible to have full confidence that it won't introduce security, licensing, or other issues without performing your own vetting.

How Hackers Can Infect Open Source Codebases

As the preceding chapters explained, there are a variety of ways to add open source to a codebase, and many good reasons for doing so.

But there are also critical security risks associated with this practice. Again, you can manage these risks, but only if you understand how hackers can use open source code as a vector for causing harm to your organization.

Most open source projects welcome contributions from anyone. That's one of the key strengths of open source development as a whole – the fact that any developer can help build it.

But this permissiveness can also breed risks. If open source projects don't adequately vet new contributors and validate their code, their open-door policies can become a vector for hackers to sneak malicious code into their repositories. This is bad not just for the projects themselves, but also for any third parties that incorporate vulnerable open source code into their own codebases.

Here's a look at how hackers can exploit open source projects, and what that means for developers who depend on open source code to help build their own projects.

Hacker Infiltration of Open Source Projects

Unlike most proprietary codebases, which are not accessible to the public at large, open source projects typically allow anyone to contribute code to them, and even go out of their way to make it easy to do so. After all, part of the reason why platforms like GitHub have become massively popular for hosting open source projects is the ease with which users of those platforms can access open source code, modify it or extend it, and push their changes back into the main codebase.

This doesn't mean that anyone can instantly contribute any code to an open source codebase without any sort of vetting process in place. Most projects carefully review proposed contributions from coders who haven't worked with the projects before to ensure that the contributions meet the project's standards for code quality and security. They also look at the backgrounds of the new contributors to check that they have an established track record of solid contributions and coding experience. This vetting process helps prevent bad code from sneaking into codebases.

However, the rigor of the vetting process for new contributors can vary widely from one open source project to another. Large, mature projects led by veteran coders tend to enforce high standards. But smaller projects, or those that are not managed well, may do a poorer job of keeping track of who they allow onto their teams of contributors.

That means that coders with malicious intent may be able to slip past the gatekeeping process that is supposed to protect open source projects.

Keep in mind that hackers need not contribute plainly malicious code in order to infiltrate open source projects in this way. They could merely propose a change that creates a configuration condition (like improper input validation or an overlooked bounds check) that enables an exploit against the application.

This means that malicious contributions aren't always easy to detect. Even experienced coders may fail to notice the vulnerabilities that hackers have intentionally buried within code they propose to contribute to an open source project.

Malicious Insiders

Complicating matters further is the fact that, once a developer has been accepted as a trusted member of an open source community, his or her activity within the codebase may not be monitored very closely.

That means that a hacker could potentially make valid contributions initially in order to gain the trust of peers, then start adding malicious code to the codebase without being closely watched.

If you think scenarios like this sound rare, think again. GitHub reports that 20 percent of the bugs within code stored on its platform were **planted by malicious actors**. Similar issues have occurred on the **NPM repository**, where hackers uploaded malicious open source applications with names similar to legitimate ones in order to trick developers into using them.

Malicious Code's Impact on Third Parties

It's not just developers and users of the open source tool or platform itself who are harmed when hackers find their way into an open source project. Third-party developers who incorporate code from an open source project into their own, internal codebases are also at risk.

In other words, if you are building an application or tool for use inside your own company, and you import some open source code to help implement it, you run the risk that an exploit or vulnerability lurking inside that code will make its way into your own application.

Protecting Against Hacks in Open Source

What can you do to protect your business from the risks associated with hacker infiltration of open source codebases?

You could choose not to use open source, of course, but that would mean shooting yourself in the foot. Open source is a great resource that, when used properly, allows developers to build applications faster and more cost-effectively than they could if they had to implement everything themselves.

A better approach is to use open source when you need it, but to be sure that you get open source code from trusted, mature projects. As noted above, these projects are more likely to perform the thorough vetting necessary to keep hackers out of their codebases.

Conclusion: Securing Your Open Source Code

This eBook has highlighted several core truths about the way organizations use open source:

- Virtually all organizations that develop software of any kind rely, in part, on open source code.
- Many organizations, however, aren't fully aware that they are using open source code extensively. Or, if they are aware, they often have little visibility into which code they are using and how they are using it.
- Open source code varies widely with regard to how secure it is and what the implications of its licensing terms are.
- Without the ability to know when and how you are using open source, you run the risk that the open source code you rely on introduces security, compliance, licensing or other risks to your organization – and you may not even know those risks exist.

In short, we've learned in the preceding chapters that open source carries risks, and you can only manage those risks if you know how and where you are using open source code.

This is why organizations that use open source in any way should leverage Software Composition Analysis (SCA) tools, like Checkmarx. Checkmarx allows you to scan your entire codebases to build an inventory of areas that include open source components – even if your developers have modified those components since originally borrowing them from open source projects. It also identifies where the open source code came from, and which security or licensing issues are associated with the code.

With a tool like Checkmarx, you can take full advantage of open source while enjoying the confidence that your applications and business are safe from the risks that open source can introduce. In other words, you can have your cake and eat it too – you get the fast, efficient development operations that open source enables, while still keeping your code secure and compliant.

Contact us for a Free Demo!

Checkmarx

Checkmarx is constantly pushing the boundaries of Application Security Testing to make security seamless and simple for the world's developers while giving CISOs the confidence and control they need. As the AppSec testing leader, we provide the industry's most comprehensive solutions, giving development and security teams unparalleled accuracy, coverage, visibility, and guidance to reduce risk across all components of modern software – including proprietary code, open source, APIs, and Infrastructure as code. Over 1,600 customers, including half of the Fortune 50, trust our security technology, expert research, and global services to securely optimize development at speed and scale. For more information, visit our website, check out our blog, or follow us on LinkedIn.