# SOLVING YOUR OPEN SOURCE RISK WITH SOURCECLEAR

BY VERACODE

VERACODE

# SOLVING YOUR OPEN SOURCE RISK WITH SOURCECLEAR BY VERACODE

## RISKS ASSOCIATED WITH OPEN SOURCE LIBRARY USE

The demand on software development teams is greater than ever. With the cultural move towards DevOps, the implementation of CI/CD systems, and the desire to operate in an agile manner, developers are being asked to push out more software, in shorter periods of time, than ever before. In turn, developers are increasingly relying on open source libraries, or pre-built pieces of code available online, which allow them to add functionality to their code without having to build it from scratch.

The result is that software today is rarely made up completely of first-party code, but more often "assembled" from other sources. How much of a typical application is comprised of open source libraries? We are seeing numbers as high as 90 percent, with the remaining 10 percent first-party code used to stitch the libraries together and assemble an application.

*Background Knowledge*

**Direct vs Indirect Libraries**

When leveraging open source libraries, there are two categories of libraries pertaining to your application. **Direct dependency** libraries are all of the pieces of open source code that your developers are directly using and adding to the application they are developing. However, in the open source world, many times these libraries are also relying on other open source libraries for some of their functionality. And it's very common for those libraries to use even more open source libraries, and continue the chain – we've seen anywhere from two to over 10 levels of libraries being called, one after the other. The functionality, and potential issues, are then proliferated down the open source library chain to your application: these second layer removed libraries are called **indirect dependencies.**

### *Explosive growth of open source libraries*

And the use of open source libraries to assemble applications is accelerating. Not only are more people using open source libraries, but more individual developers, and even companies, are also on a mission to contribute to more open source projects. For CA Veracode, we're seeing more than 70 percent of our customer base leveraging one or more open source libraries in their applications. And when customers use open source libraries, they use a lot of them. On average, applications are leveraging 49 direct open source libraries – this does not count the indirect libraries, which can be in the hundreds for a single application.

### *Risk of open source libraries*

So what about the risks present in those applications? On average, applications contain seven open source vulnerabilities per application. While this doesn't seem like a big number, it only takes one to compromise an entire application. And of those applications that contain vulnerabilities, 44 percent of them contain critical vulnerabilities. So what are companies doing to protect themselves? Well, not always enough, as only about one-third of organizations do regular open source vulnerability scanning.

In the case of the Equifax breach, one vulnerability in an open source Struts2 component had huge impacts, including the resignation of the CEO.

So while the use of open source libraries is proliferating, so are the risks. With the reusable code and functionality also comes reusable vulnerabilities, and with very popular libraries used in thousands of applications, a single vulnerability, in a single piece of code, can suddenly makes thousands of applications vulnerable to the same exploit. If we look at Java applications alone, we're seeing 88 percent of Java applications are leveraging one or more libraries containing vulnerabilities.

There is also a pervasive false sense of security around open source libraries, and a general lack of understanding about the attack vectors that open source libraries can open up. While it is true that there are more "eyes on the code" when it comes to open source libraries, there is nobody ensuring that these security fixes are being disclosed, and, of course, nobody to inform you that there was a vulnerability patched. Ultimately, contributing developers are interested in the success of an open source library, not in the success of your business. It's up to the company implementing the open source library to ensure that it is safe to use, and that they are staying up to date on any known vulnerabilities in the library and their fixes.

Simply **using** open source libraries is **not a security threat to the business.** The real problem is **not knowing that what you are using contains vulnerabilities,** and that they are exploitable in your application.

## CHALLENGES IN SECURING OPEN SOURCE

When assessing the posture of your open source risk, you have to ask three key questions:

- Which libraries are we using, and do they contain any vulnerabilities?
- Does the vulnerable library actually do anything bad?
- Can I react fast enough to new vulnerabilities?

### *The unknown unknowns – does my code contain any vulnerabilities?*

With 70 percent of our customers leveraging open source libraries, and software being comprised of up to 90 percent open source code, the amount of open source code in use is proliferating rapidly. In addition, so are the problems that accompany that code. Development and security leaders alike are asking the question, "Does my code contain any vulnerabilities?" There is a need to both better understand the composition of software, and the vulnerabilities that are present. A bigger problem comes when you start assessing indirect dependencies as well (we defined indirect dependencies earlier in this paper), because the sheer volume of vulnerabilities can easily skyrocket out of control. Even though a developer could be pulling in only five open source libraries directly, those libraries could easily pull in hundreds of other open source libraries with them – including all of their vulnerabilities.

There is another fundamental issue that can be broken into two parts:

First, today the most used source of vulnerability data is the National Vulnerability Database (NVD), which contains tens of thousands of vulnerabilities across all different types of applications, including open source libraries. The NVD, unfortunately, is completely overrun with vulnerability submissions. (You can view their dashboard at any time[1].) At the time of this paper, the NVD is receiving about 200 new or modified vulnerabilities each week.

Second, everything submitted to the NVD has to be known and disclosed, but what about those vulnerabilities that few people know about because they haven't been broadly disclosed? Oftentimes, when a vulnerability is disclosed and assigned in the NVD, it has been patched for several months. For example, in August 2018, a new Remote Code Execution vulnerability for the Apache Struts library was disclosed and assigned in the NVD, however this vulnerability was actually found and patched back in April 2018. If anyone had been watching the commit logs of the library, they would have been aware of this potential vulnerability for months before the public was made aware of it. Malicious actors are targeting these libraries.

### *Not all vulnerabilities are created equally – is the vulnerability in the execution path?*

While there are different severity levels of vulnerabilities, there is also a second dimension that people don't, or can't, take into account – and that's whether the vulnerability is actually impacting their code at all.

Software composition analysis (SCA) traditionally worked like this:

- First, the tool reads your files to determine which open source libraries and versions are being used.
- It then compares that list to a known data source, such as the National Vulnerability Database.
- Finally, it surfaces to your developers a list of all known vulnerabilities associated with those versions of the libraries.

But when leveraging an open source library, it's very common for a developer to only use a small subset of that library, for a very particular function or capability. Thus, it's very likely that a vulnerability in the library is never actually being called by your application, and in essence, is not exploitable.

With many tools out there, developers will receive an extremely large list of vulnerabilities for all open source libraries packaged in your application, and they will have to make a judgement call on what to fix first – and how much is worth fixing before pushing to production.

Most companies prioritize high-severity and critical vulnerabilities, but ignore lower severity vulnerabilities. However, you could decide to fix only the most critical of vulnerabilities, but there could be a noncritical one that is actually impacting your code and could be used to exploit the application. We argue that rather than fixing a high-severity vulnerability in a function that is not called by your application, you should prioritize a

[1] https://nvd.nist.gov/general/nvd-dashboard

medium-severity vulnerability that lies in the execution path and puts your customers at risk.

We've found that if developers can actually get this information delivered to them, their remediation time is reduced by up to 90 percent, due to their ability to prioritize and immediately fix the vulnerabilities that pose the highest likelihood of exploitation. That time is crucial.

### *Exploit attack window is shortening – can I respond fast enough?*

Developers not only have to release software faster than ever before, but they also have to respond to security threats faster than ever before. They also must have a tool that helps them maintain their development velocity.

Time between vulnerability disclosure and exploitation is counted in days, sometimes hours. As in many areas of IT security, the attacker is at an advantage.

Attackers are known to look through the commit history of open source libraries to find code changes that remediate a vulnerability. Oftentimes, developers make these changes without publicly disclosing the vulnerability to the NVD. This means, open source users are not alerted of security issues or urged to update their versions. Meanwhile, attackers can develop exploits based on the vulnerability and often start attacking applications with a "spray and pray" approach across the entire Internet. Once they successfully exploit a machine, they can access data or get a session pivot to other parts of the organization's network.

On the flip side, defenders need to be aware of not just this one vulnerability, but of all of the vulnerabilities in their open source code, with the added complexity that they are not all listed in the NVD. In addition, they need to work with the development teams to fix all vulnerabilities that put the organization at risk. Without the right solutions, this is a near impossible task.

### *Real-life example of the gap between discovery and disclosure*

Let's have a look at a real-life example: In August 2018, a public announcement was made of a remote code execution vulnerability in the Apache Struts open source library. However, this vulnerability was actually privately disclosed in April 2018, and finally in June 2018 the vulnerability was patched in the library – two months before the public disclosure. If attackers were monitoring that change, they would have seen the commit, explored if it was a security fix, and then focused their attention on finding applications with out-of-date libraries – thus giving them a place to narrow their focus and efforts.

Now let's look at a different example. The Equifax hack was the result of not patching a publicly disclosed vulnerability, in this case CVE-2017-5638 in the Apache Struts framework. While patching disclosed vulnerabilities is the first step you take to address the problem, it was merely days between the vulnerability being disclosed, and the vulnerability being exploited in an unpatched system. The larger your organization, the harder it can be to patch these systems in time, so companies need a leg up to get ahead of these disclosures. Unfortunately, as we've shown, you cannot rely solely on the CVE system and the National Vulnerability Database for open source security vulnerabilities alone.

Security and development teams need a way to level that playing field, stay ahead of vulnerabilities, and prioritize those that actually affect their code. Developers also need

something that provides automated and instant feedback, directly in their CI system – where and how they expect to work. This is the only way they can remediate issues in the moment, and maintain their velocity.

VERACODE

## HOW VERACODE CAN HELP

The threat that open source use has presented in the past couple of years has been growing, and it's not enough for us to help our customers detect and fix all vulnerabilities – we need a way to help get ahead of the problem. Customers have been asking for more than the bare minimum that most vendors provide. Veracode acquired SourceClear in 2018 to meet the needs of our customers faster. We understand that the use of open source is not going to go away, and believe the use of secure open source is a positive way for companies to accelerate their time to market. Simply using open source is not a security threat to the business; it's not knowing that what you are using contains vulnerabilities and that they are exploitable in your application that is the real problem.

### *The problem we solve with SourceClear*

SourceClear answers the three fundamental questions: Does my code contain vulnerabilities, do they actually impact my application, and can I react fast enough to new threats?

| | |
|---|---|
| **Does my code contain vulnerabilities?** | • Comprehensive database of both known vulnerabilities and undisclosed vulnerabilities<br>• Language coverage for all of your applications<br>• Detailed information on the individual vulnerability and its potential impact |
| **Do they actually impact my application?** | • Vulnerable methods feature indicates if vulnerable code is in the execution path (Java, Python, Ruby, & .NET)<br>• Dependency mapping of open source libraries, to understand the entire ecosystem of open source libraries that your application depends on to function – including direct and indirect libraries |
| **Can I react fast enough?** | • Integrated into your CI, automating open source vulnerability scanning on every build<br>• Proprietary Vulnerability Database, leverages data mining and machine learning, to get ahead of vulnerabilities before they are disclosed<br>• Monitoring deployed code for newly identified vulnerabilities, and sending alerts upon discovery |

VERACODE

*How you benefit from CA Veracode's solution*

AppSec programs are only as good as the rate that they are adopted and adhered to. In order to have a successful program, you must balance the needs of the business, the requirements of security, and the desires of the developers. Your company is looking to bring secure software to market as fast as possible, and they're relying on your developers to accomplish this.

Veracode's Software Composition Analysis solution, by SourceClear, provides you with:

- **Better developer adoption:** SourceClear was built first and foremost for developers. It plugs directly into the systems they are already using, returns results in a way they expect, and allows them to prioritize what to fix first.
- **Better inventory and awareness of open source risk:** Because of our extensive database of not only known vulnerabilities, but also unknown/undisclosed ones – you can be assured that you get the full view of your risk posture.
- **The ability to maintain development velocity:** With integration into the CI, prioritization of vulnerabilities with vulnerable methods, fix and update advisories, and lightning-fast feedback, your developers are checking their code while they're working on it. In this way, they ensure that they are getting ahead of any problems and not waiting until it's too late and dealing with a lot of unplanned and unscheduled work.

But how do you know that your program is working, and that it's successful? You will want to track a few of these metrics as you roll out your program:

- **Reduction in total number of vulnerabilities in your application:** By eliminating vulnerabilities early and often, you will be able to reduce the overall number of vulnerabilities in your applications delivered to production.
- **Elimination of all vulnerabilities that actually impact your code:** The highest priority for your organization should be critical vulnerabilities, and those that actually impact your code.
- **Number of times code is scanned:** We have evidence that customers who scan more frequently have far fewer vulnerabilities in their applications, and remediate those vulnerabilities at a much faster rate.
- **Number of applications and teams leveraging open source vulnerability scanning:** Your adoption rate is an important measure, and indicator of the future success of your program. Full coverage is critical to ensuring you are releasing secure applications to market, and protecting your customers' data.
- **Time to remediation:** Knowing how fast your developers are remediating their vulnerabilities is an important stat, since it's a very clear indicator of how security is impacting overall developer velocity.
- **Time to approve:** Without technology scanning your code every time it is touched, your security team has to go through extensive work vetting each individual library for developers to use. Reducing this to zero and having the scan occur during the application build will accelerate software development.

VERACODE

## VERACODE SOURCECLEAR DETAILS

Our teams have worked hard to develop a Software Composition Analysis (SCA) solution that works for organizations attempting to better secure applications against open source risk without reducing the development velocity of the business.

### *SourceClear's role in your DevSecOps program*

One of the true hallmarks of DevOps is the ability to fit tools and processes into what works for your businesses – as such, you have the flexibility to fit SourceClear wherever you want it in your SDLC. We currently support two methods of scanning your source code: via integration into your CI system for automated scans, or integrated into your developer's workstation for manual command line interface (CLI) scans.

### *SourceClear is easy to get started*

SourceClear is comprised of two parts, an agent-based scanner that scans your source code, and a cloud-based reporting and management platform.

The agent is typically deployed into a CI system, where development teams run it on every build. We see the most success when it is run during integration testing.

We also support the agent being deployed directly on a desktop and used in a CLI, or as a plugin to a package manager.

*Background Knowledge*

**Package Managers**

When it comes to using open source libraries, software developers rely on tools called package managers to determine which libraries and which versions of libraries are needed by the software they are creating. These package managers make many important decisions on behalf of the developer behind the scenes, in most cases without them ever realizing. For instance, for every one library that a developer chooses to use, package managers can add as many as ten more libraries to the final code, creating complex dependency trees. The result is that when an application built with open-source code is finally ready to be shipped, it's usually made up of 90 percent of code that came from other people.

### *SourceClear in the CI is so important*

When an organization has implemented a CI system, the general consensus is that everything moving to production passes through the organization's CI system. Thus, if you put your security checks into your CI system, then you know that everything moving to production is passing through the security scan. Many competitors in this space try to assess open source vulnerability dependencies in different ways, but usually miss the mark in terms of accuracy and coverage. The result is solutions that scan internal repositories, which could be potentially bypassed, or attempt to scan the source code text and try to guess at which libraries are being used. With SourceClear, we scan at the time of build, which gives us the most accurate architecture of your code and how the vulnerabilities affect your code.

VERACODE

*Support*

SourceClear supports a number of languages. With the exception of Java, the SourceClear agent requires that you use one of the following supported package managers for the language your team is developing in – these cover 99% of the requirements out there.

We support the following languages:

| Languages | Package Managers |
|---|---|
| Java | Maven, Gradle, Ant, or imported Jars |
| Javascript | NPM, Yarn, Bower |
| .NET | Nuget |
| Python | PIP |
| PHP | Composer |
| GO | Trash, Glide, GoVendor, GoDep, GoGet, Dep |
| Ruby | Gems |
| Objective-C | Cocoapods |
| Scala | SBT |

**You can always find the latest up-to-date supported languages list at:**
https://www.sourceclear.com/docs/what-does-sourceclear-support/#package-managers

We support a number of systems to initiate your open source vulnerability scans:

| CI & Build Systems |
|---|
| Atlassian, Bitbucket, CircleCI, Codeship, Gitlab, Jenkins, TravisCI, Maven, and Gradle |

Don't see your CI system listed? While we have listed a few here that we have written specific scripts for, our system is extremely flexible when it comes to integrating with CI systems, requiring only a single line of code to get started – thus we have not found a CI system that we can't get SourceClear up and running with yet.

| Desktop |
|---|
| Command Line Interface (CLI) for local scanning |

Integrated into your local environment to manually scan for vulnerabilities and return the results to our platform – ensuring that you know which vulnerabilities you have.

| Issue Tracking |
|---|
| Atlassian and Github |

Our integration with Atlassian JIRA and GitHub allows your developers to create issues/tickets when new vulnerabilities are discovered in their applications. Since your developers can choose which ones to make tickets for, this gives them the flexibility to prioritize their most critical vulnerabilities.

**You can always find the latest up-to-date supported systems list at:**
https://www.sourceclear.com/docs/what-does-sourceclear-support/#ci-tools

## *NOBODY DOES IT QUITE LIKE SOURCECLEAR*

While we help our customers protect themselves from open source risks, there are a number of vendors that do the same basic discovery and reporting. However, SourceClear's focus is on coverage and actionable results that help development teams maintain velocity while ensuring more secure applications are delivered to production. The following are the technology aspects that set us apart from the rest of the competition.

**NVD vs SourceClear Proprietary Database**

We mentioned earlier that the NVD, although very robust, is not able to keep up with the sheer volume of vulnerabilities being disclosed and/or updated daily. One of the problems is that the NVD is for all software vulnerabilities disclosed, not just open source vulnerabilities. Thus, these open source library vulnerabilities get stuck in a log jam behind everything else. The NVD has a pretty slick dashboard on their website showing how many vulnerabilities are coming in every day, week, and month. This screenshot, taken at the time of writing this paper, shows that there were 247 CVEs processed in that week alone.

**NVD Dashboard**

**CVEs Received and Processed**

| Time Period | New CVEs Received by NVD | New CVEs Analyzed by NVD | Modified CVEs Received by NVD | Modified CVEs Re-analyzed by NVD |
|---|---|---|---|---|
| Today | 0 | 7 | 66 | 0 |
| This Week | 247 | 249 | 209 | 28 |
| This Month | 851 | 990 | 582 | 33 |
| Last Month | 1143 | 1546 | 1275 | 23 |
| This Year | 12665 | 10910 | 10407 | 156 |

However, there is one more issue not being addressed. The only way that a vulnerability makes it into the database is if a software developer or an independent security researcher submits it to the NVD. It is not uncommon for vulnerabilities to be fixed, but never disclosed or submitted to the NVD. We mentioned this example earlier, but as a reminder, the Apache Struts Remote Code Execution vulnerability that was disclosed to the public in August 2018 was actually patched back in April. That means for four months, anyone not on the latest version of the library was potentially vulnerable to this exploit. This, by the way, was the same type of vulnerability that led to the Equifax breach the year before.

To combat this, we have developed our own database that includes all of the open source vulnerabilities in the NVD, as well as our own list of vulnerabilities in open source libraries that have not yet been disclosed to the NVD. In many cases, the vulnerabilities we find and record have either not been disclosed yet and are in the time between patching and full public disclosure, or in some cases, there was never any intent to disclose the vulnerability and its fix. There is a third category we track, which are "Reserved CVEs." We

take the Reserved CVE IDs from the NVD and then find the vulnerabilities in the public repos, in order to give you a head start on the fix prior to full public disclosure.

So how do we discover these vulnerabilities that are out in the wild and are either unknown or undisclosed? That's where our data mining, proprietary machine learning, and security research process come into play.

**Machine Learning**

Due to the incredible amount of hard work that has gone into developing, improving, and fine tuning our machine learning technology, we won't go too far into the technical details. But we feel it's important to talk about what we use it for and, in general terms, how it works, so that you can feel confident that our data helps provide a better and more holistic view of the open source vulnerability landscape.

The goal of our machine learning technology is to automate the identification of potential security vulnerabilities from commit messages and bug reports. In open source projects, bugs are typically tracked with issue trackers, and code changes are merged in the form of commits to source control repositories. Thus, if an organization is able to monitor all of these repositories and review each new bug issue and commit message, they could identify potential vulnerabilities. However, there are tens of thousands of open source repositories, with hundreds of thousands of bug-tracking issues and commit messages to comb through, with new ones hitting every day.

Our system uses natural language processing and real machine learning to identify potential vulnerabilities in open source libraries with a high level of accuracy. By analyzing the patterns found in past commit messages and bug-tracking issues using machine learning, our model can identify when new commits or bug issues resemble a silent fix of a potential vulnerability. These potential vulnerabilities are then raised to our security research team. These silent fixes can be a silent killer for your data protection.

**Security Research Team**

The security research team is responsible for taking all of the data that our machine learning system sends us, and reviewing each potential vulnerability to ensure that it is in fact real. If there are any false positives that return, the team adds this feedback into the system to better tune the algorithm over time.

The security research team ensures that any vulnerabilities discovered are not included in the NVD already, or not already included in our proprietary vulnerability database. If neither are true, then this is determined to be a new undisclosed/unknown vulnerability.

The team assigns the new vulnerability an internal score, unique identifier, and adds it to our proprietary database. Upon being added, security alerts are sent out to any customer using the library.

**License Database**

In addition to tracking security vulnerabilities, which is the main focus of our technology, we also offer the ability to identify a limited set of open source licenses that can pose both business and financial risk to organizations.

We are able to help companies identify open source licenses like Apache, MIT, MPL, BSD, CCDL, and GPL. These are the most requested types, and we are working to identify and add more license types that our customers are looking to find.

**Scanning Agent**

The SourceClear agent is deployed with a single line added to your CI system, or within your CLI, by performing a CURL command, and pulling the most up-to-date agent from CA Veracode. On subsequent scans, the command checks if there is a newer version of the agent and pulls that version down to your environment.

The agent is most effective when scanning your system during the time of an application build, since it is able to create a call graph of the application to understand how it is composed, and how data and controls flow through your application. This allows us to get better coverage of your code, and identify vulnerable methods in supported languages.

**Call Graph**

You can run the scanner interactively from a command line or automatically as part of a continuous integration process. By integrating with your build process, you always know exactly what code is being used. With each scan, it generates a call graph of your application and generates a complete and accurate dependency graph that describes with absolute precision what versions of open source libraries are being used. Using both the dependency graph and call graph, the scanner then performs control flow analysis to determine, with the best level of precision possible, if your application is actually using open source code in a vulnerable way.

**Vulnerable Methods**

Most SCA solutions simply look at your application's dependency file to determine what versions of which open source libraries are being pulled into the application. They then compare that list to a vulnerability database, usually the NVD, and pull back a list of libraries that have reported vulnerabilities in them. However, just because a vulnerability exists in one part of the code, doesn't necessarily make the entire library vulnerable. Often, the vulnerability only compromises the larger application if it leverages the part of the library that holds the vulnerability, and has data and control flowing through that part of the code. Essentially, this is like saying that a house is not secured and

locked up because the shed out back isn't locked up. Yes, we want the shed to be locked up, but it's not actually impacting the house at all. Vulnerable methods work the same way.

We solve this problem with our agent, and the way we discover open source dependencies in an application. The call graph creation that happens by the agent (discussed in the previous section) allows us to see how data and controls flow through your application – which includes determining if that data is flowing through the vulnerable part of the open source library being used. If it is, we indicate back to the developer that this is in fact a vulnerable method, and it is causing your application to be vulnerable to exploits. When we scan with vulnerable methods, we find that up to 90 percent of the vulnerabilities reported are not likely to actually impact the code. While we definitely recommend developers stay on top of the latest up-to-date version of every library they use, in reality development and security teams have to work together to make trade-offs between security and speed. With vulnerable methods, developers can tackle the vulnerabilities that are actually likely to make their application vulnerable first, reducing their risk by the most in the shortest amount of time.

Today, vulnerable methods is available for Java, Ruby, Python, and .NET.

*Background Knowledge*

### Reducing noise is critical

One of the biggest complaints about SCA solutions we hear from prospects and customers, especially when talking to security leaders, goes something like this...

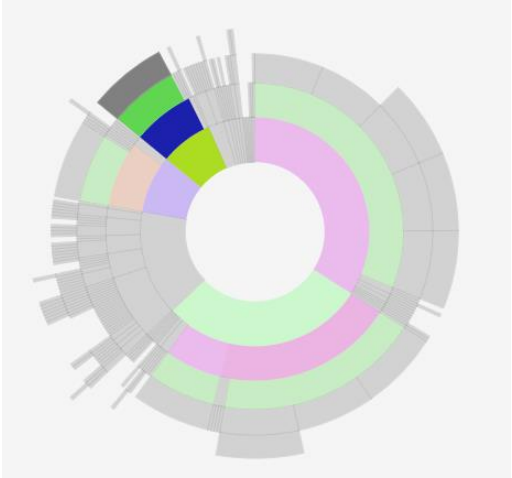
> *"We are having a really tough time with adoption, and I don't know what to do. When we scan our application with our current SCA solution, and I send those results to the developers, they get upset because they feel like I'm wasting their time. They have to go through each finding, and figure out how it's impacting their code, so that when they upgrade to the newest version of a library, they can determine if the new library will change any of their functionality. However, they come back and tell me that the vulnerabilities I'm sending them, although present in the library itself, aren't actually in their application and aren't affecting the integrity nor security of their application at all. So they see me as someone who doesn't understand how software works, and refuse to work with the tools we have."*

Does that sound familiar at all? If you're using any other SCA solution on the market, your developers are thinking this, even if they haven't expressed it yet. That noise is frustrating for your development team, because their goal is to ship applications as fast as possible. The great news is that you can actually help them do this, by providing them with a solution that shows not only the inventory of vulnerabilities, but which ones they should prioritize as actually affecting your application.

**Dependency Graphs**

Earlier in the paper, we described the difference between direct and indirect dependencies. A lot of the work our engine does centers around mapping out your application and all of the open

source libraries that the application depends on: both directly pulled in by developers, and indirectly pulled in from those direct libraries. Other SCA solutions on the market struggle with this dependency mapping, often reporting many duplicates of the same vulnerability for a single application. This redundancy in reporting can lead to extreme bloat of results reporting, generate many false positives, and give your development teams the difficult task of identifying what's real versus what's not.



This image is a visualization within our platform of a dependency graph. The empty circle in the middle is your application, and all of the sections around it are different direct and indirect libraries. In this specific example, all of the colored sections are libraries containing vulnerabilities that affect the application either directly or indirectly.

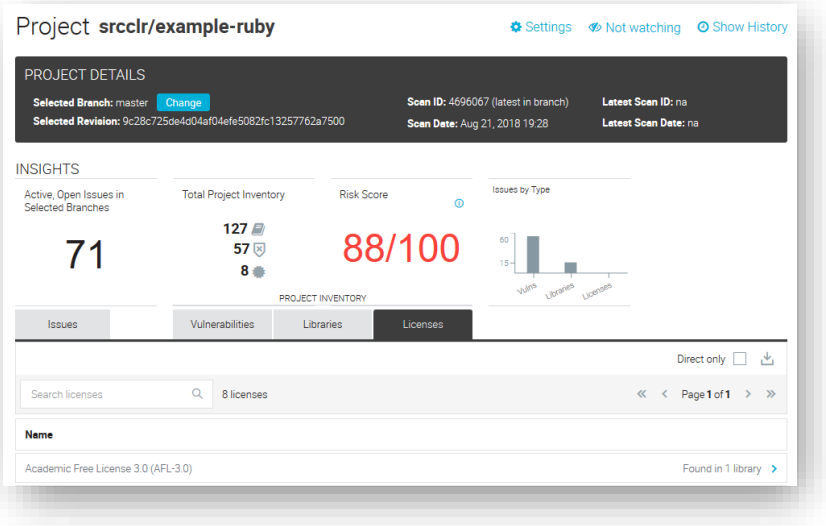
It is important to realize that, again, just because your development teams are only using 10 open source libraries, they could actually be pulling in hundreds of different libraries indirectly. Our scanner identifies all of these, the versions being used, and any vulnerabilities that they contain. And for supported languages, it identifies the call stacks and traces the vulnerabilities through your application to identify those that actually impact your application and leave it open to exploits.

VERACODE

## SOURCECLEAR UI OVERVIEW

The SourceClear User Interface was designed with development velocity in mind. All of the important information is readily accessible, and easy for developers to navigate around and get actionable information from as fast as possible. While a good part of our solution centers around the ability to integrate both scanning and reporting into your CI system, a lot of information is available in the platform.

### *Reporting*




The SourceClear platform is divided into a number of constructs:

**Vulnerabilities:** Each individual vulnerability that has been found in one of the libraries connected to your application.

**Projects**: The body of code being scanned by the SourceClear agent.

**Workspaces**: A collection of projects. You may put any number of projects into a workspace. Some companies equate a workspace to a development team, or to an entire application.

**Libraries**: The names and versions of the libraries that are connected to your application.

**Licenses**: Listing of the open source licenses that are currently associated with the open source libraries being leveraged by your code.

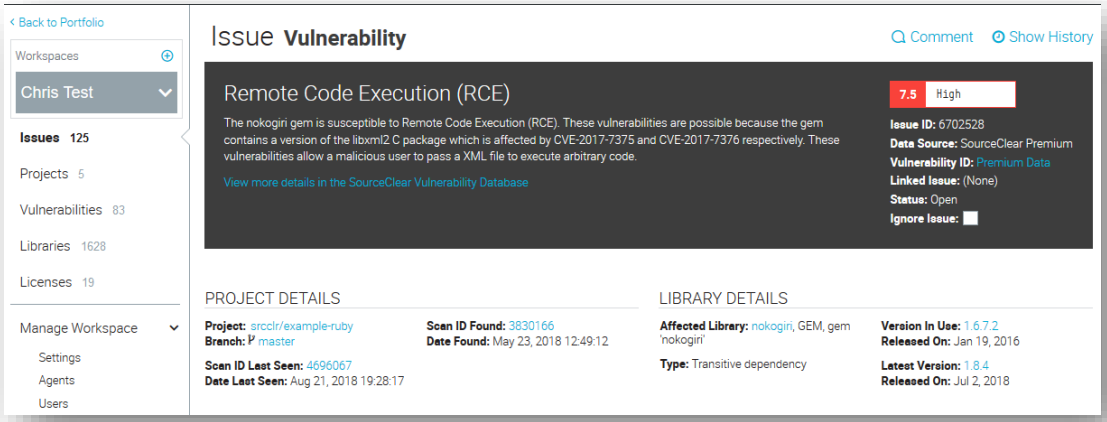
**Issues**: Platform items that are related to something that needs to be addressed. Usually an issue is created for each vulnerability found. However, an issue could also be an out-of-date library, or a library with a risky business license.

You can view the results of scans through any of these lenses (Projects, Libraries, etc...). Each project has a number of insights available, as seen in the screenshot, including the number of active open issues, an inventory overview, and Risk Score of the project.

VERACODE

## *Vulnerability Details*

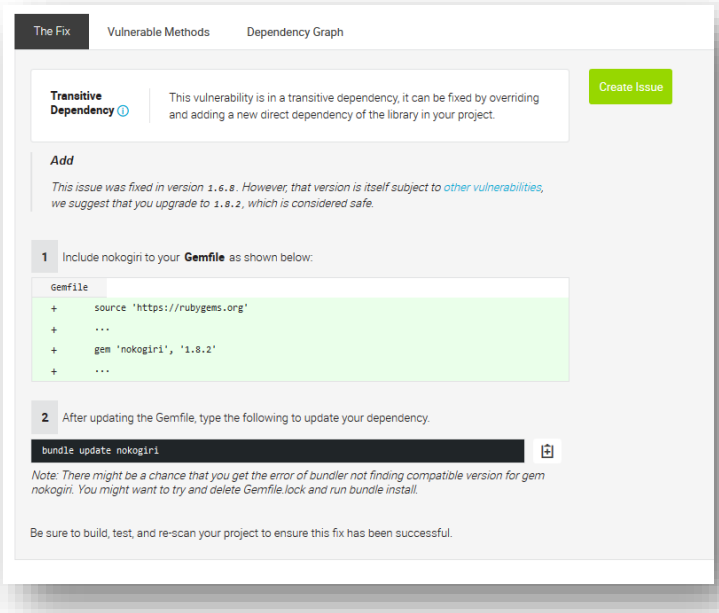
We provide a lot of information on each vulnerability that is reported after a scan. We supply not only a description of the vulnerability, but also how it is related to the project it was discovered in. We surface which branch it was discovered in, when the vulnerability was found in your project, when it was last seen, the current status of this issue, the data




source, and whether it is a direct or indirect dependency. In this example, you can see that a Remote Code Execution (RCE) was discovered on the master branch of the example Ruby project. The vulnerability was found on May 23, 2018, and seen last on Aug 21 of the same year. Due to the critical nature of the vulnerability, it has a score of 7.5. Additionally, the data source shows as “SourceClear Premium,” which means this was a vulnerability that we found, that was not reported to the NVD. Remote Code Executions are extremely dangerous vulnerabilities that have led to some of the most serious data breaches in history –and if your code contained this vulnerability, you would be aware of it with SourceClear.

## *Fix Recommendation*

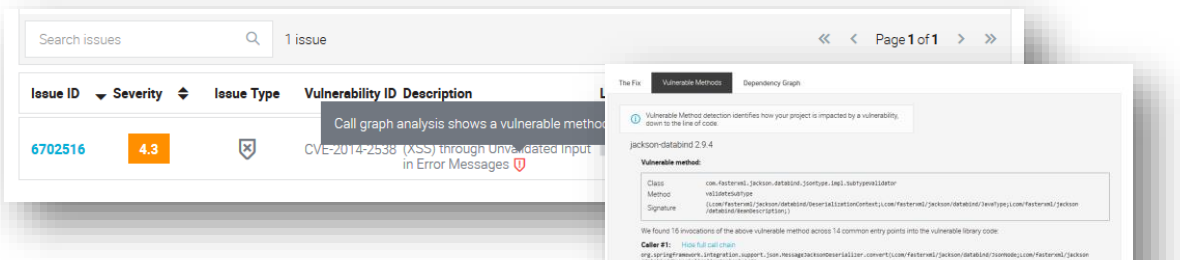
Actionability is extremely important to our customers. Simply finding vulnerabilities is one thing, but providing some help on how to fix them is extremely key. Most fix recommendations are straightforward, such as simply updating the library. However, we do not just recommend you update to the library where the vulnerability was fixed, especially if that library contains vulnerabilities. Instead, we provide the safest version of the library as a recommendation for you to update to.



Another important fix recommendation comes when you have indirect dependent libraries in you project. Our system helps identify this indirect library, recommends the safest library to update to, and provides the code to pull the library directly into your code. By pulling it directly yourself, you are able to bypass the indirect dependency without waiting for the authors of the direct libraries to update their code.

## *Vulnerable Methods*

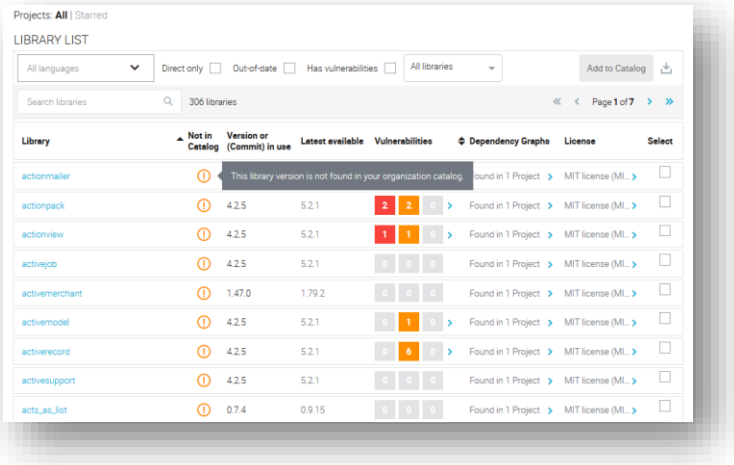
The most actionable piece of data we provide is an indication of whether a vulnerability is actually impacting your code as a vulnerable method. In the UI, on your issues screen, we add an icon if a vulnerability discovered is part of a vulnerable method that is actually




putting your code at risk for compromise. As part of the vulnerable methods reporting in the UI, we also provide a stack trace that shows you exactly how the data flows through your code and through the vulnerability. This way, you can determine if you should update the library, or mitigate the vulnerability by replacing the method with a different one, or protecting the data in a different manner.

## *Library Catalog*

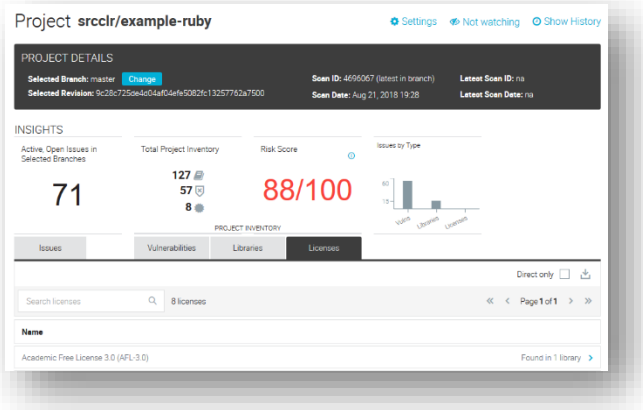
While not to be confused with an open source repository whitelist/blacklist, our library catalog feature allows security and development leaders to keep a list of libraries that have been identified as safe to use by the organization. While we don't restrict the use of libraries based on this catalog, we do indicate in the UI whether the library is part of the organization's library catalog. This catalog can also be reviewed, to see what version is the latest and how many vulnerabilities are in that version of it.



VERACODE

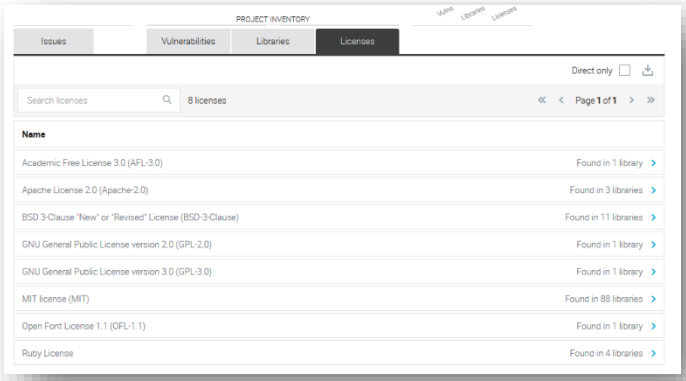
## *Risk Score*

The Risk Score on the project page is our internal calculation of the associated risk that the project contains – the higher the number, the riskier the project. This takes into consideration the number of vulnerabilities found in the project, the criticality level of those vulnerabilities, and, most importantly, whether those vulnerabilities are called as a vulnerable method in your project.



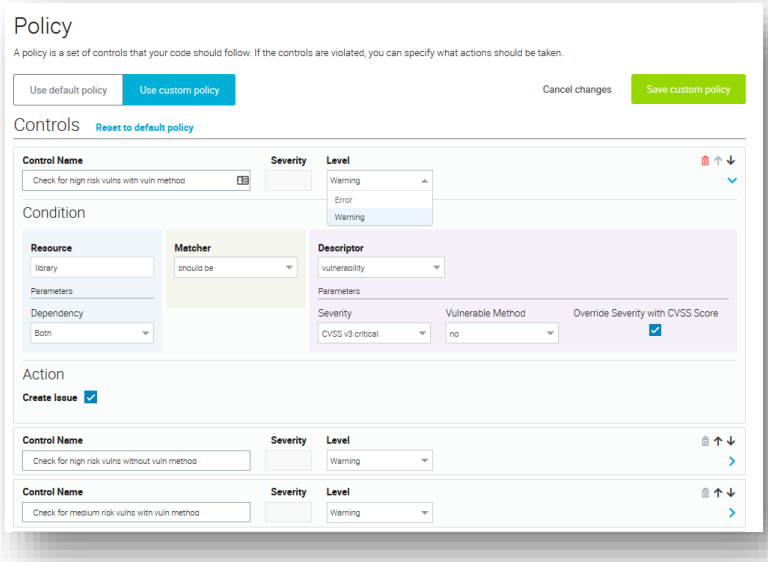

## *Licenses Found*

Again, while not the primary focus of our solution, our customers ask us to provide information on what types of open source licenses are being invoked by the project. We currently check for over 180 of the more popular libraries, and we are adding more of them to our system all of the time. If there is a library we do not cover, we encourage you to let your Security Program Manager know so that we can get it added to our list.




## *Policies (Rules, Filters, & Actions)*

In SourceClear, we have a section called “Policy,” which is essentially a collection of Rules, Filters, and Actions that determine what happens every time the agent scans your projects. The controls are completely customizable, across many different rule sets. For example, you may only want to report on Very High or Critical vulnerabilities as reported in the NVD. And when these come back, you can determine what level is sent back to your CI, which will either be a warning or an error. In your CI system, your developers can set up what happens when either one of these flags are presented. In most cases, an error will be used to break the current build, which prevents software from continuing to other downstream activities

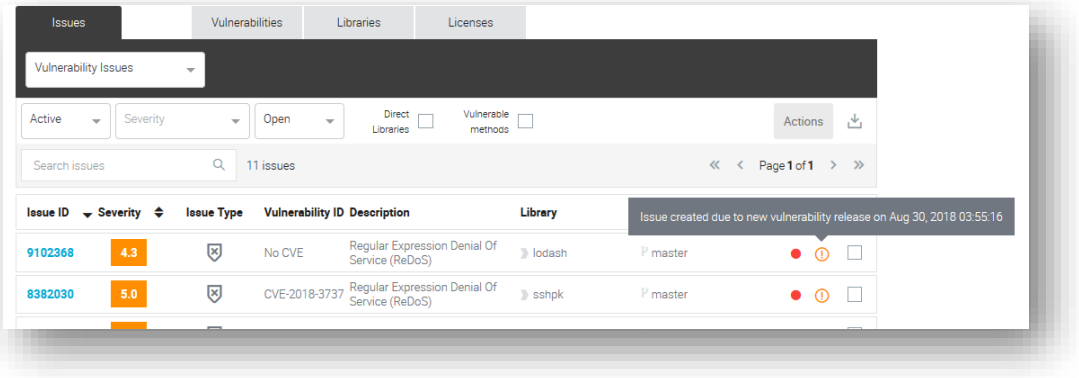


without being fixed. A warning will allow the pipeline to continue but still surface a message to the developers and report the issue in the CI system.

You can assess a number of dimensions, including whether the vulnerability contains a vulnerable method or not. It's important to note that these are meant to allow developers to surface results as they see fit.

### *Security Alerts*

Vulnerabilities can be discovered at any time, and it's important that we keep our customers up to date with the latest information regardless of whether a library is being continuously scanned or not. With Security Alerts, if a vulnerability is added to our database after you have scanned a library, we push the results to your workspace, and indicate that it is a new vulnerability discovered, regardless of the last time you scanned that library. Additionally, Web Hooks can also be set up to POST information anytime a new




vulnerability is discovered after a project has been scanned.

In this screen, this issue was created on August 30, 2018 – however, the project was actually scanned on March 2, 2018. This is crucial for the security of our customers, as it allows them to be notified of new vulnerabilities without having to scan – ensuring you are always up to date on the latest security posture of your applications.

## MATURE YOUR APPSEC PROGRAM WITH CA VERACODE

There are a number of different places where you could start your application security program, and a lot of different paths to mature your program – but there are not a lot of companies that can help cover your needs from end to end. When assessing your options for your AppSec partners, you need to look for a company that can cover the entire software development lifecycle (SDLC), with a strong focus not only on first- and third-party code, but also the ability to actually implement a mature program. CA Veracode is the market leader in application security, and our years of experience have shown that those companies that evaluate their first-party code, plus open source libraries, and do so early, midway, and late in the SDLC have the best coverage. With CA Veracode, you can ensure a scalable, cost-effective AppSec program that helps make security part of your competitive advantage.