SANS

A SANS Product Review

Automating Open Source Security: A SANS Review of WhiteSource

Written by **Serge Borso**
Advisor: **Dave Shackleford**

Sponsored by:

WhiteSource

September 2018

Introduction

Many sources indicate that 60–80 percent of code in applications today is based on open source components.[1] This open source code often includes vulnerabilities that, if not managed properly, can expose organizations to potential breaches. This paper takes a close look at how the WhiteSource solution can handle the myriad of open source vulnerabilities through real-time detection and remediation.

With attackers increasingly focused on targeting vulnerable open source components in applications, detecting and remediating them is vital. WhiteSource's technology detects those vulnerabilities throughout the software development life cycle (SDLC), prioritizes which vulnerabilities to address first and provides remediation paths.

WhiteSource automates the process of open source selection, approval, detection of vulnerable or problematic components, and remediation through its open source security and license management solution. This automation provides substantial value in a DevOps/DevSecOps culture. Many organizations have adopted a DevOps approach to unify software development and operations, from coding/planning to monitoring/maintenance. Development teams would love to be able to manage code-related security concerns while addressing the bigger picture of automating the entire process

[1] "Report: Commercial Software Riddled with Open Source Code Flaws," www.linuxinsider.com/story/84469.html and "The State of Open Source in Commercial Apps: You're Using More Than You Think," https://techbeacon.com/state-open-source-commercial-apps-youre-using-more-you-think

of open source components selection, tracking and approval—but they rarely have the time. In an ideal world, security teams could enable development teams by streamlining application vulnerability efforts while incorporating risk management practices and prioritizing remediation efforts. So, we tested WhiteSource to see how it would measure up to these tasks.

At its core, WhiteSource is a security solution—a technology that integrates with the SDLC to enable security professionals to help development teams by providing information on the best open source components for a project and being able to identify and help prioritize vulnerabilities in the code. Having access to technology with the capability to identify components and their dependencies, enforce security and license policies automatically, and prioritize where developers and security teams should focus their efforts not only decreases time-to-market, but also increases the quality of code under development. All this while decreasing the amount of time and effort dedicated to vulnerability remediation after the code is put into production.

This paper will explore how WhiteSource's technology works, how it can be set up, and how to use it to automate the process of open source component vulnerability detection, remediation, licensing resolution and overall vulnerability management.

How It Works

There are several ways to integrate your codebase with the WhiteSource solution, from the very basic option of linking with a GitHub account to more robust choices such as installing the WhiteSource plug-in and integrating your package managers, build tools and continuous integration (CI) servers. WhiteSource also offers a REST API so customers can pull WhiteSource data into their homegrown systems. Users can access the API key, which is essential for all integrations, on the WhiteSource dashboard under the *Integrate* tab. Figure 1 shows the integration and API key screen.

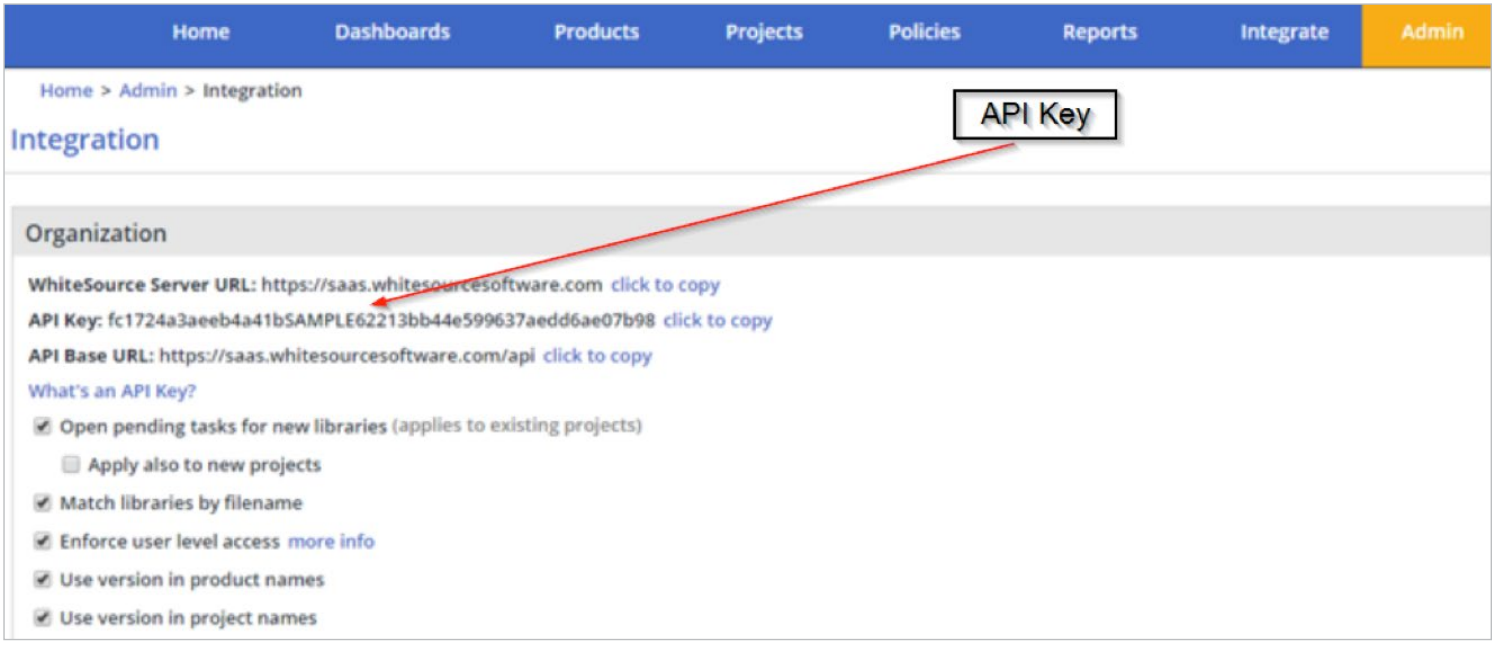


Figure 1. Integration and API Key

Integrating WhiteSource

For metafork, we started the WhiteSource integration by creating a new project and adding applicable libraries. As mentioned earlier, you can integrate the WhiteSource solution with your repositories, package managers, build tools and CI servers through a system of plug-ins, agents and services. Integrating WhiteSource with our build tools was a straightforward process. For example, when integrated and provided with our API key, the available Maven plug-in created, by default, a WhiteSource project for each module defined in our **POM** file. This flexibility results in options such as the capability to exclude and ignore modules or the entire Maven project. The Integration tab also provides access to more intricate features, such as specifying a user key to uniquely identify a user. After it's configured, this plug-in's job is to utilize the WhiteSource API to send open source usage information to WhiteSource in the form of hashes. WhiteSource uses the hashes and does not actually scan your code. We are then able to see relevant information in the WhiteSource dashboard. In addition to integrating with package managers and CI servers, the WhiteSource solution also integrates with issues trackers, JIRA and Work Items. Automating issue tracking without having to rely upon a new solution to track and address the issue is convenient.

After it was integrated in our environment, WhiteSource's technology performed a hash calculation (essentially creating a digital signature) for all of the components in a given project. From there, WhiteSource compared the unique signatures with the signatures in its own database and thus detected the open source components in our project. Upon running our build, WhiteSource automatically populated the dashboard with the real-time report in Figure 2, which shows all open source components detected as well as issues associated with those components.

Our Test Project, Metafork

For our testing purposes, we created a working use case and called our project "metafork." This project is comprised of 10 open source libraries, mostly Apache 2.0 licenses, and will ultimately be responsible for providing an API to track cryptocurrency price fluctuations. This use case will enable us to interact with the WhiteSource solution, explore its features and understand, from a security perspective, what the tool can really do. The concept of integrating metafork with the various features of the WhiteSource solution will serve as the backdrop for subsequent discussions in this paper.

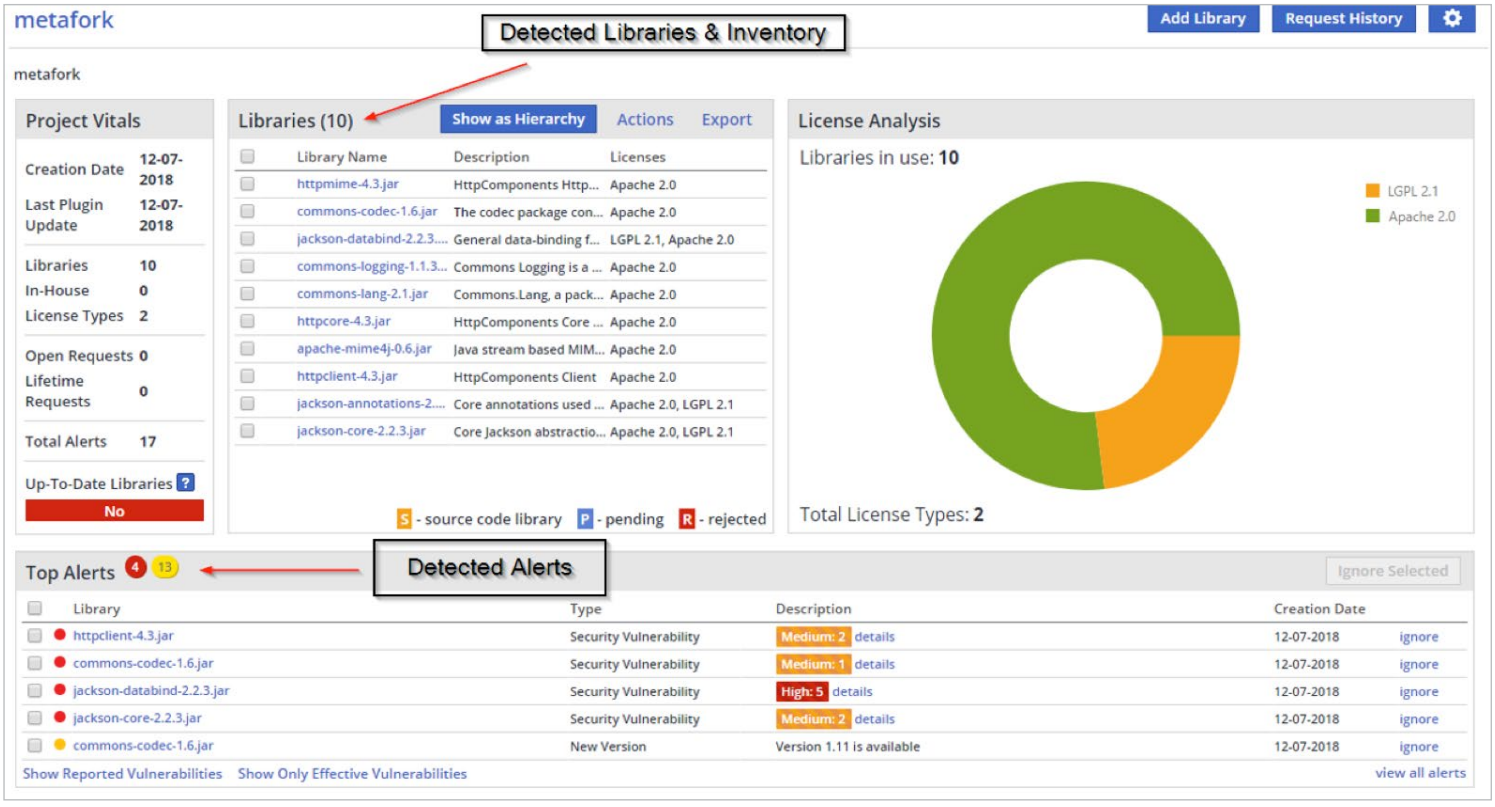


Figure 2. Initial Project View

With the build process complete, we can reveal the depth of the WhiteSource vulnerability database. After viewing the documentation, we noted that more than 300,000 vulnerable components were accounted for via aggregated sources such as security advisories, open source project issue trackers and common vulnerabilities and entries (CVE) details. As more vulnerabilities are identified in the wild, WhiteSource continually adds this information to its database.

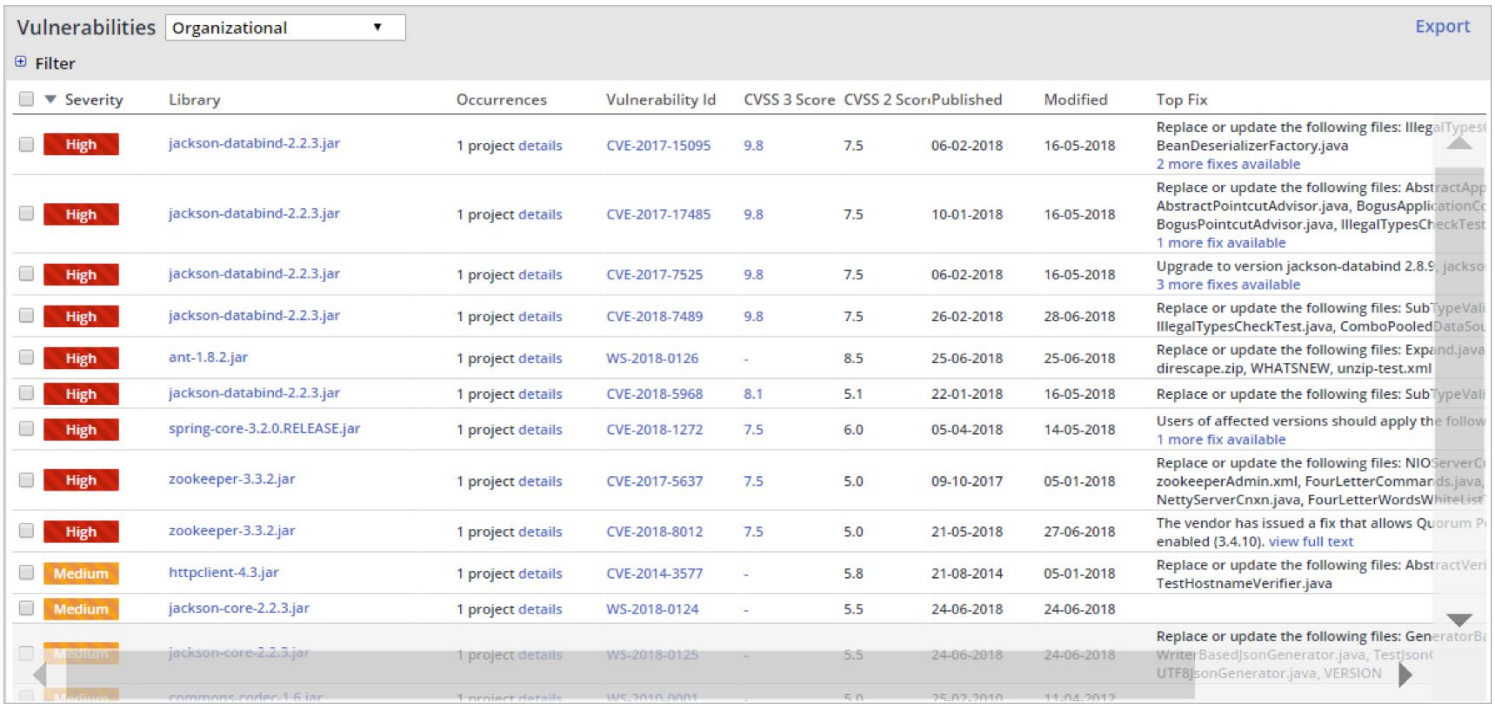
The dashboard shown in Figure 2 helps security professionals quickly ascertain product inventory, top alerts and license distribution. This information provides the teams responsible for application security with a unified dashboard showing the most relevant security information. The graphs are interactive, which allows users to drill into each component for more detail on any given section.

Eliminating False Positives

Producing accurate and actionable information is critical to managing vulnerabilities. A single false positive can invalidate an entire result set and waste considerable time, a situation not unfamiliar to security analysts and developers alike. The WhiteSource solution creates a unique hash of each component referenced in our project and correlates this hash with known components in the WhiteSource database, resulting in zero false positives. This action is performed each time someone runs a build so we can easily see affected versions and when fixes/updates have been applied correctly. At this point, WhiteSource is not actually scanning our code; instead, the solution calculates the digital signature and compares that signature to signatures in the WhiteSource database. The WhiteSource solution's default behavior is to look for differences in baseline signatures (the deltas), which leads to an increase in efficiency.

Figure 3 shows WhiteSource's vulnerability overview screen. At a glance, we see the most important information to help us understand what libraries and subsequent projects show identified vulnerabilities, the severity of the flaws and the corresponding Common Vulnerability Scoring System (CVSS) scores. This screen also provides a direct link to the CVE details and the top-rated fix to address the issues.

Figure 3. Vulnerability Overview upon Initial Integration



A mature threat and vulnerability management program should take application vulnerabilities into account by identifying, validating, assigning remediation tasks, fixing, rescanning, reporting and gathering metrics. The information in WhiteSource's vulnerability overview forms the basis for the data organizations need to meet their AppSec vulnerability management objectives.

The WhiteSource database contains information on open source components such as the associated license, version and version history, and general information associated with the component, as well as copyright information and other data. The database aggregates information from hundreds of open source repositories and supports more than 200 programming languages, which is no small feat.

WhiteSource Features and Functionality

At this point, we have successfully integrated our project, explored the most basic dashboards and have an understanding of our inventory as well as the initial vulnerability/risk profile. This next section focuses on the features and functionality of the WhiteSource solution and provides insight into how to get the most out of the platform.

Open Source Components Detection—Inventory

Our project inventory consists of a multitude of libraries that WhiteSource breaks down into their basic components: version, language (type), description, associated product(s), SHA1 hash and the like (see Figure 4).

Inventory Report - metafork

Filter

By Licenses Value ^Apache 2.0|, Apache 2.(Filter

Library Name	Type	Description	Licenses
commons-lang-2.1.jar	Java	Commons.Lang, a package of Java utility classes for the classes that are in java.lang's hierarchy, or are considered to be so standard as to justify existence in java.lang.	Apache 2.0
apache-mime4j-0.6.jar	Java	Java stream based MIME message parser	Apache 2.0
jackson-core-2.2.3.jar	Java	Core Jackson abstractions, basic JSON streaming API implementation	LGPL 2.1, Apache 2.0
httpclient-4.3.jar	Java	HttpComponents Client	Apache 2.0
commons-codec-1.6.jar	Java	The codec package contains simple encoder and decoders for various formats such as Base64 and Hexadecimal. In addition to these widely used encoders and decoders, the codec package also maintains a collection of phonetic encoding utilities.	Apache 2.0
jackson-databind-2.2.3.jar	Java	General data-binding functionality for Jackson: works on core streaming API	LGPL 2.1, Apache 2.0
jackson-annotations-2.2.3.jar	Java	Core annotations used for value types, used by Jackson data binding package.	LGPL 2.1, Apache 2.0
httpcore-4.3.jar	Java	HttpComponents Core (blocking I/O)	Apache 2.0
httpmime-4.3.jar	Java	HttpComponents HttpClient - MIME coded entities	Apache 2.0
commons-logging-1.1.3.jar	Java	Commons Logging is a thin adapter allowing configurable bridging to other, well known logging systems.	Apache 2.0

Figure 4. Inventory Overview

When the WhiteSource solution determines that one component has a vulnerability, it is this inventory detection mechanism that can help us easily identify other portions of our product base that the flaw affects. This quick detection and identification provides security personnel with a means to ascertain the breadth of a given vulnerable component.

Vulnerability Detection and Remediation

WhiteSource detects open source components with known vulnerabilities in your products and provides remediation information by utilizing its expansive database sourced from the open source community. In the case of metafork, the 10 unique libraries that comprise the project generated 17 alerts, stemming from known vulnerabilities to outdated libraries and licensing issues.

Figure 5 shows a detailed drill-down into a specific vulnerability identified in our test project. The link to the CVE details is handy in instances when more robust information is required to understand the vulnerability. The CVSS score helps to convey the criticality of the finding while the section below it highlights the fixes most appropriate to addressing the issue.

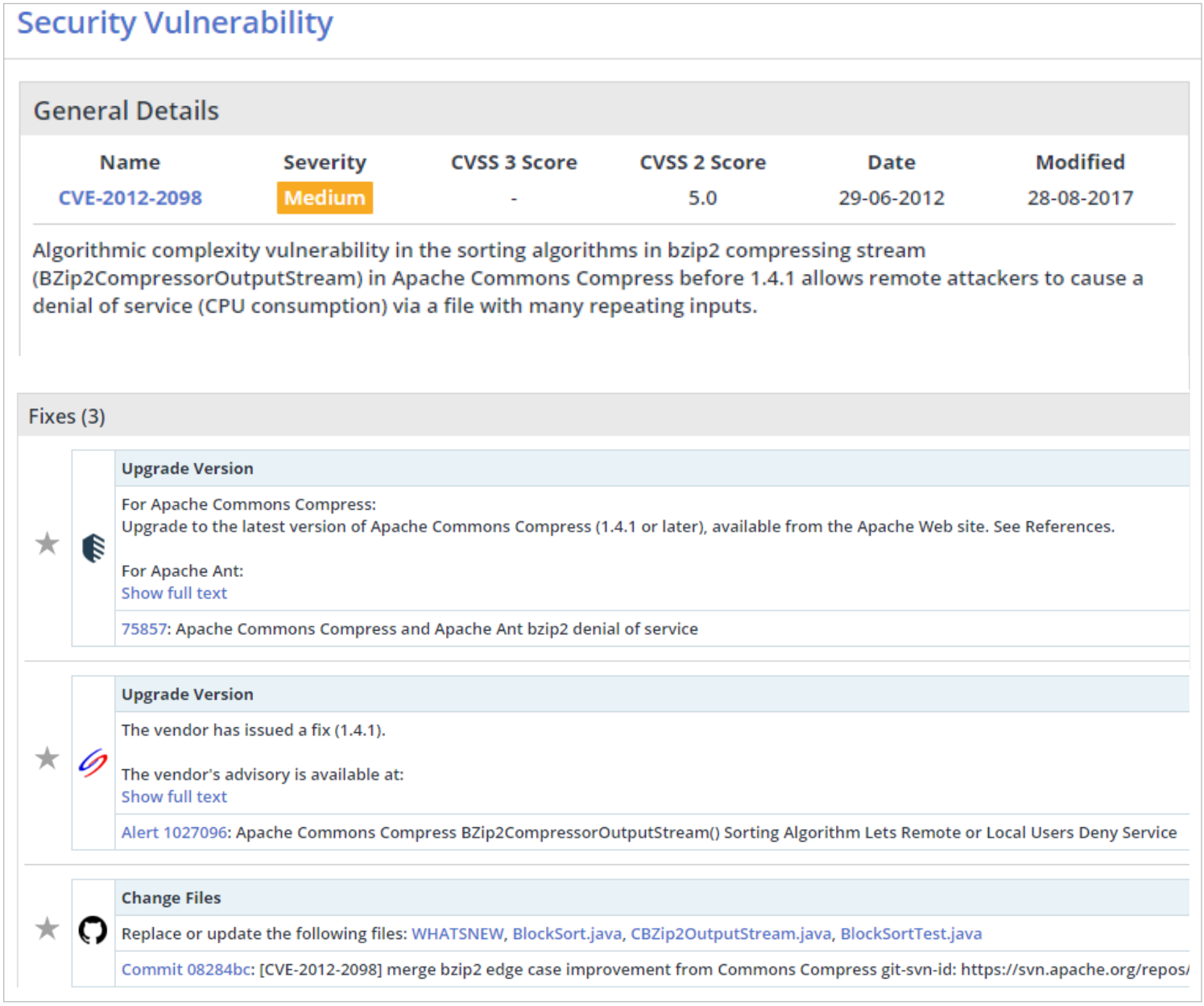


Figure 5. Vulnerability Details

In addition to sourcing vulnerabilities from multiple databases, guaranteeing zero false positives and alerting in real time, WhiteSource provides suggested remediation steps that typically involve updating the affected software when possible. Alternative methods of dealing with vulnerable software include replacing the affected files or updating versions.

License Compliance

Depending on your organization, you may have license restrictions dictating which types of licensing are appropriate for the components of your applications and which licenses require additional approval for use. Conversely, a common situation when dealing with open source is the lack of any license being associated with a given library. You'll be able to see what, if any, licenses your code is associated with after you run your integration plug-in and import your project. The WhiteSource solution will detect the open source libraries in your project(s) and subsequently assign an open source license. In-house or commercial components will display as unknown or otherwise unspecified licenses in the dashboard. See Figure 6 for an example of how WhiteSource identifies licenses.

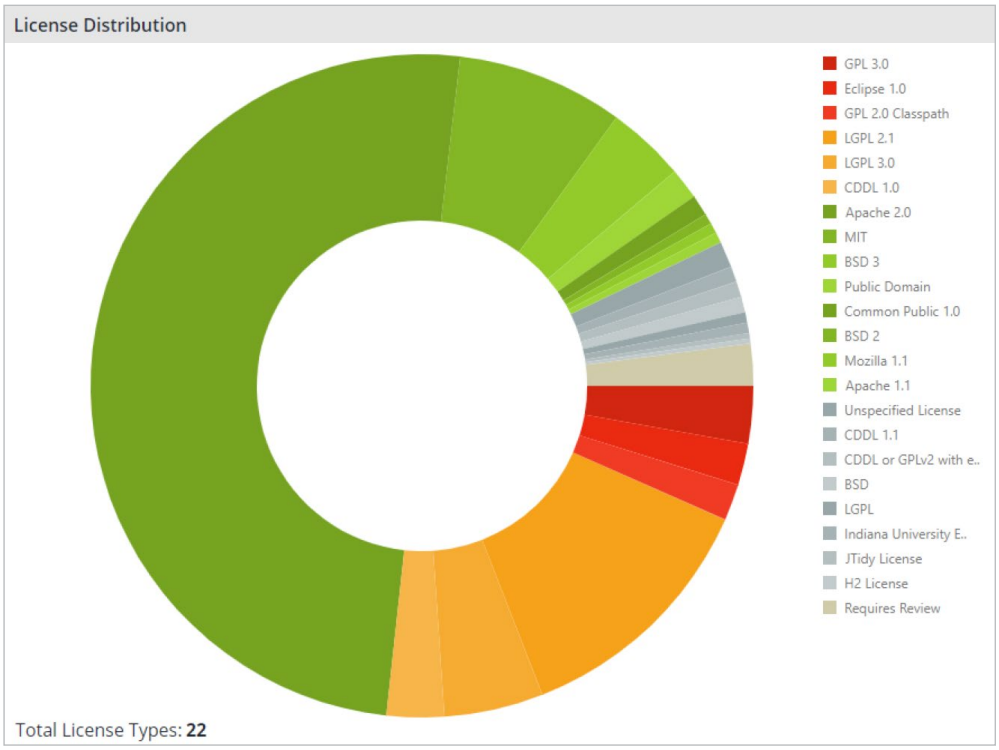


Figure 6. Identified Licenses

The dashboard displays actionable alerts until we either mark the libraries as "in-house" or perform our due diligence identifying the open source nature of the libraries. WhiteSource provides the option to select a license and see which libraries and corresponding projects use the license.

For our proprietary libraries, we can mark those as "in-house" by setting a rule based on the library name or other unique indicator. To create this rule, select the *In-House* option under the *Admin* tab on the dashboard. The default behavior of the WhiteSource solution is to identify all open source licenses, making it easy to see which licenses are not in compliance while providing the capability to remedy the situation. Figure 7 shows how to add the in-house rule for custom licenses.

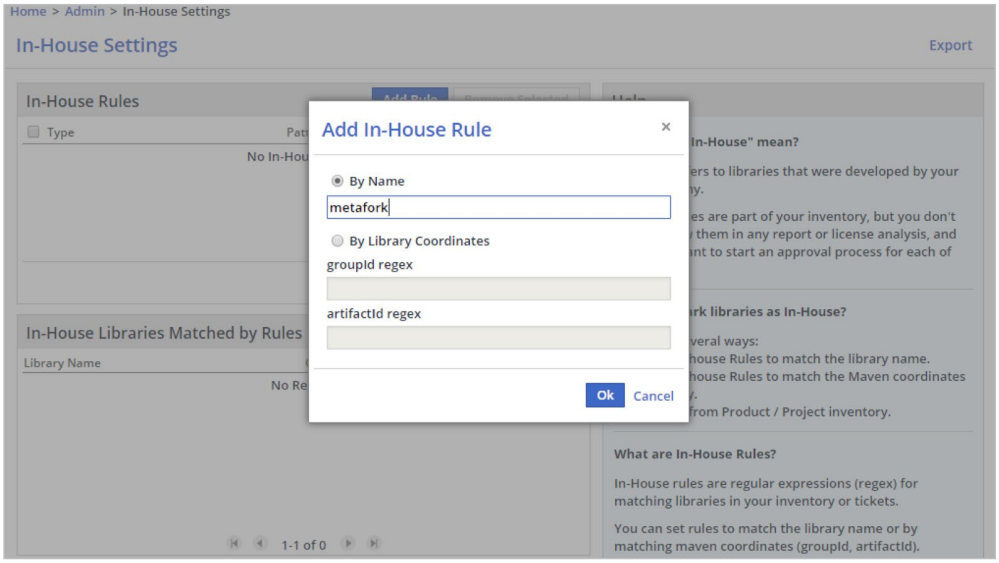


Figure 7. Adding a Rule for Custom License Categorization

Policy Enforcement

Automation can enhance an organization's speed to market and, when done properly, can increase accuracy, quality of code and even the control of the process. WhiteSource gives enterprises the capability to enforce policies and approval processes automatically, by allowing security teams to approve, block or reassign specified open source components. Figure 8 shows the Policy view in WhiteSource.

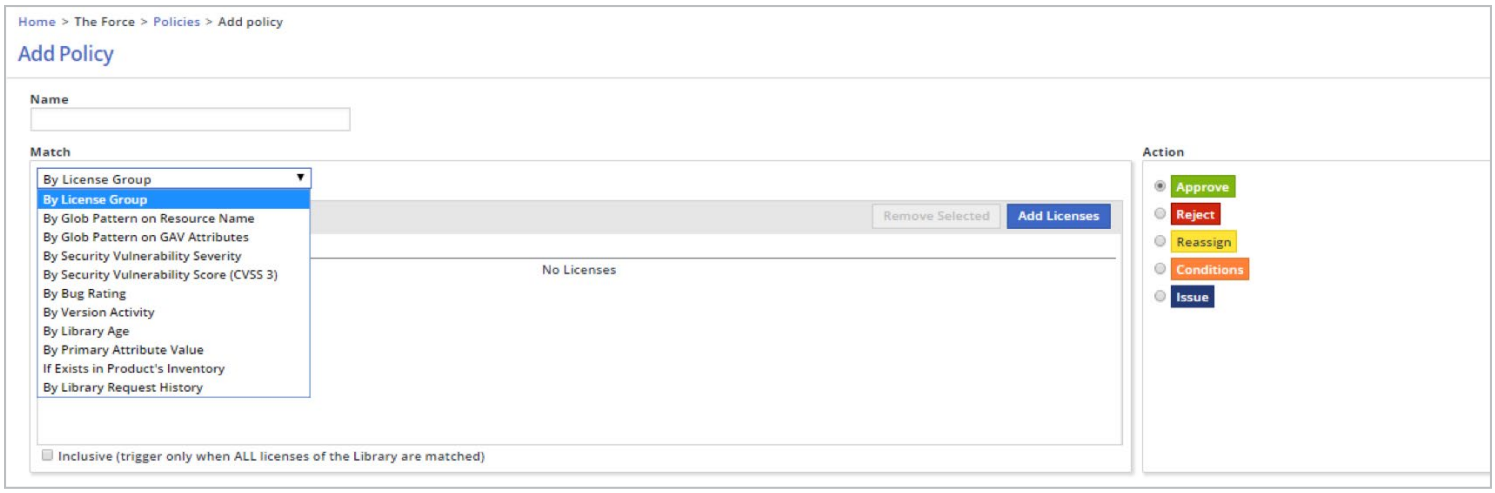


Figure 8. Policy View

Policies can be configured at an organizational level (encompassing all products and projects), at product/project levels or even according to an attribute type to suit the granular needs of an organization, product and project respectively. To set up a policy, click on the prominent *Policies* navigational link to access the Policies page, add a new policy by denoting its name and how the technology should match the condition required to enforce the policy; then define the intended action. For instance, to create a policy to create a Jira issue when WhiteSource detects a library encompassing a vulnerability with a CVSS score greater than 5, we would set the applicable fields as shown in Figure 9.

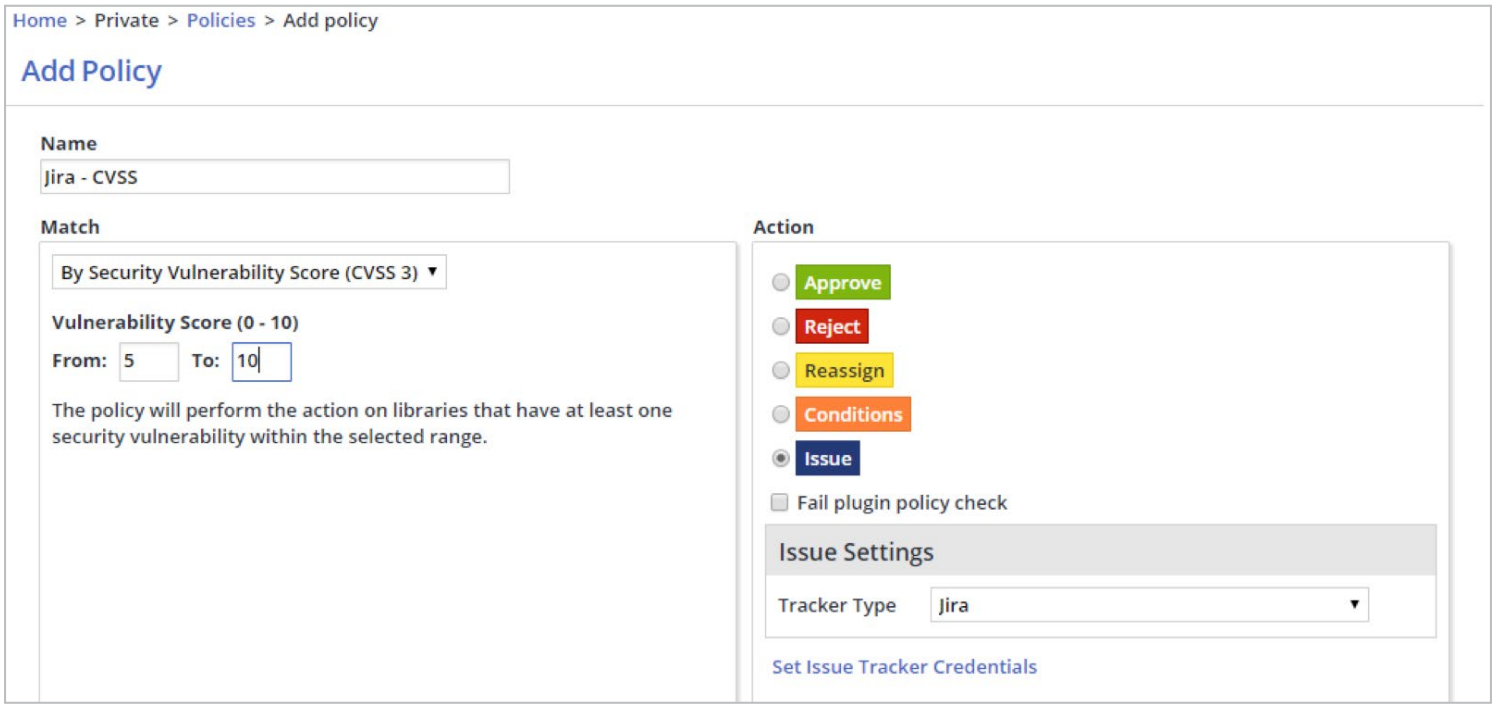


Figure 9. Creating a Policy

This policy provides all the information needed in a ticket. For example, every new vulnerability found that meets the stated criteria (CVSS greater than 5) will open a ticket with the vulnerability name, all links and a suggested fix for that vulnerability, making it easy to resolve and remediate the vulnerability. Leveraging this capability facilitates quality coding standards and enables automating pass/fail/issue tracker creation for a build. It simplifies the process and provides notifications when policy criteria are met, enabling the right teams to enact remediation steps as warranted and with minimal effort.

Alerts

An integral part of producing and maintaining a project is managing open source components throughout the life cycle, including post deployment. WhiteSource handles this management by continuously monitoring the BoM of the latest build of the application in the event that vulnerable components are identified and matched in the database. WhiteSource provides a variety of means to alert on vulnerabilities, including email alerts, failing a build or leveraging the dashboard to provide a quick view of a project's state. Figure 10 illustrates the dashboard view of organizational alerts, where we can choose to ignore certain alerts, determine the cause of an alert and drill down into project-specific alerts.

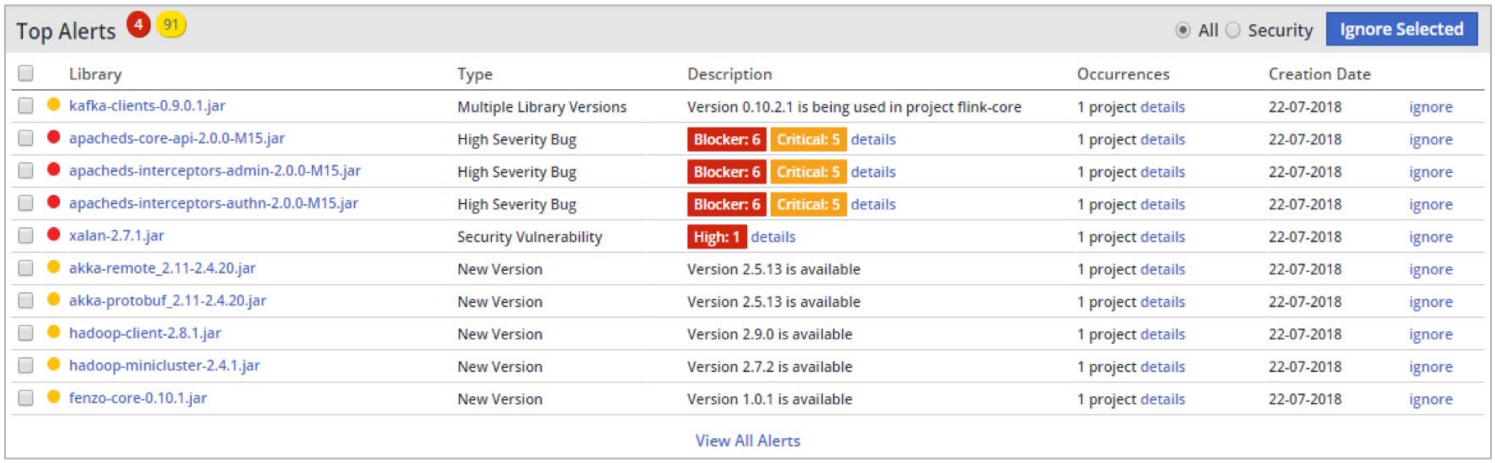


Figure 10. Alerts Dashboard

WhiteSource also issues alerts when newer versions of a library are made public, when bugs are identified or a component has multiple licenses associated with it. As with other WhiteSource dashboards, the alerts overview does a good job of indicating the key elements of what caused the alert and provides options for drilling into the details or ignoring the alert.

Reports

Using the navigation tab for *Reports*, we can generate reports using the dashboard. WhiteSource provides more than a dozen options for reports, which takes the guesswork out of creating and tuning specific criteria. Of special note are the due diligence and risk reports. The risk report shows the overall risk score per organization or per product. The information contained in this report can be used to quantify the security of a given application or business unit as well as track how code quality and developer emphasis on security have improved security posture over time. Such metrics assist a business

in understanding and evaluating its risk posture and determining where additional resources may be required. Last, the quality (for instance, how well maintained/active the code is) of a given component is ranked based on the inclusion of up-to-date libraries and the specific vulnerabilities noted in the "Quality" section of the risk report. You can drill down to get details on the CVSS score, the description of the issue and a recommended fix. Figure 11 shows a risk report depicting this information.

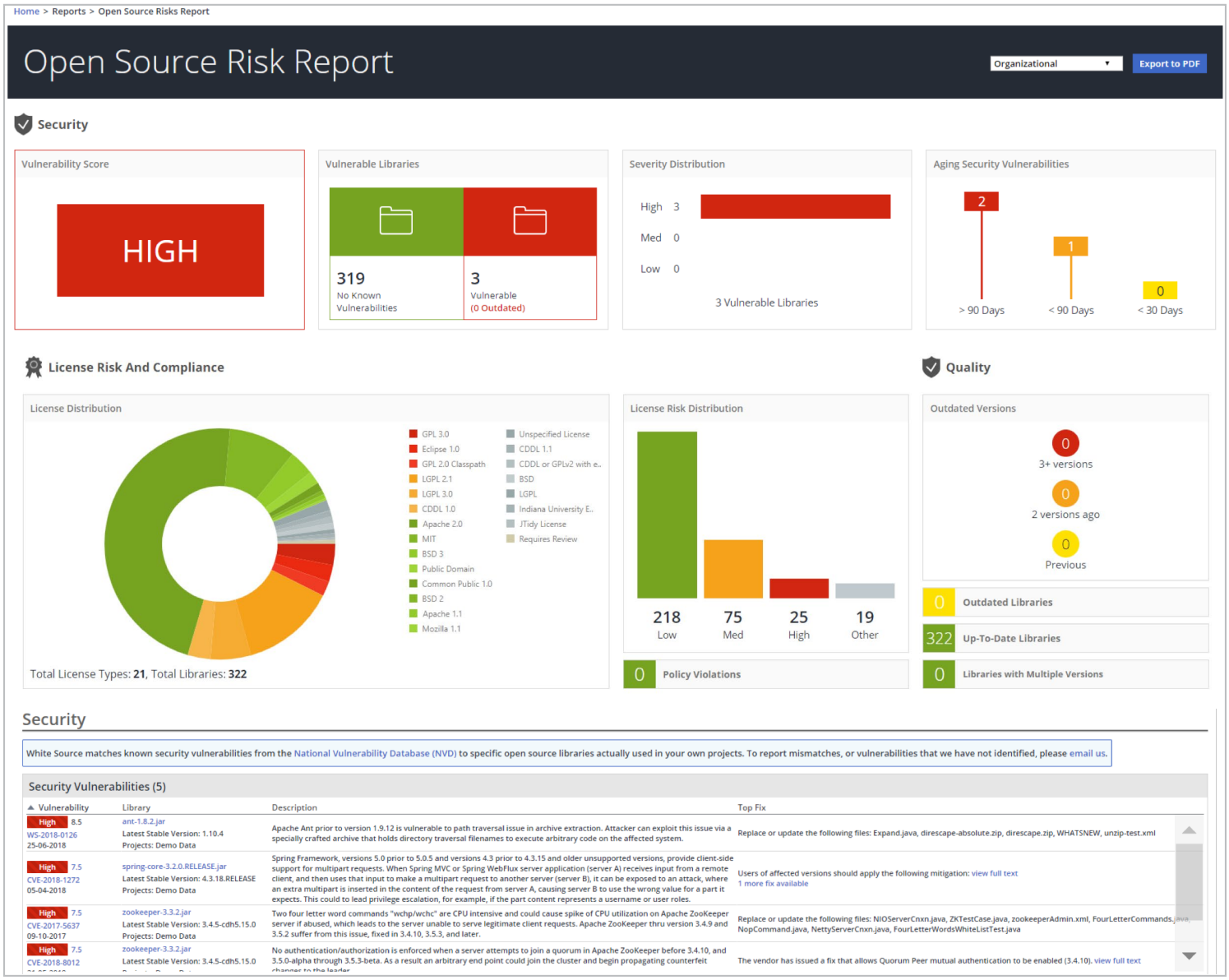


Figure 11. Risk Report, License/ Compliance and Security Details

The due diligence report provides homepage reference links in addition to a listing of every library, its associated license, risk rating, homepage reference links and author/ copyright information, if applicable, on a per-project basis. All of this aggregated information is available in various locations in other reports or dashboards. When executed as a due diligence report, the resulting XML or Microsoft Excel output can be handed to an auditor or investing team conducting their own due diligence.

Effective Usage Analysis Technology

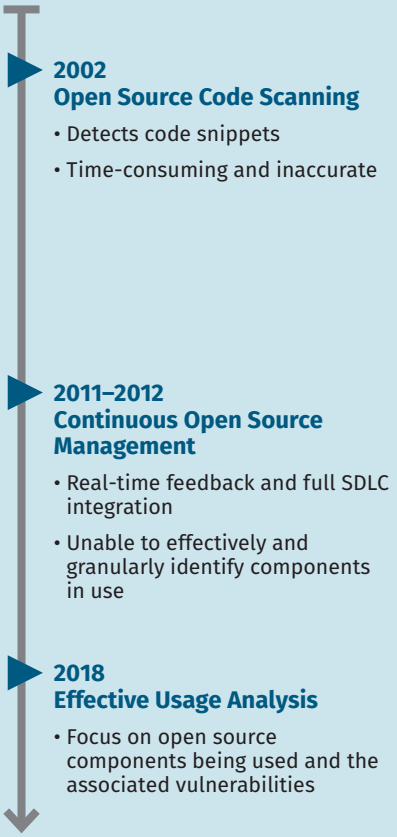
At this point, we have audited the metafork project with WhiteSource. We set up our organization, product and projects; created an inventory; identified vulnerabilities and corresponding remediations; took a look at license compliance; configured enforcement policies and alerting; and created reports. This setup and use of features provides application security teams with the tools to monitor and address usage of open source components as new builds are created and progress throughout the SDLC.

The use of open source components is on the rise, and with it, the number of vulnerabilities that, if not managed properly, can expose organizations to potential breaches. Because the number of potential vulnerabilities makes it impossible to address all issues, prioritization of the vulnerabilities is key to securing the applications. The technology required to provide the necessary tool set is mature and was recently updated to include WhiteSource's newest implementation of software composition analysis (SCA) technology: Effective Usage Analysis.

A Brief History of Software Composition Analysis

To understand the latest technology, we must understand its beginnings. Original SCA essentially attempted to match code snippets to find references to open source components in a given application, which proved to be time-consuming and inaccurate, generating lots of false positives. Second-generation SCA gained prominence around 2011–2012 and featured significant improvements such as real-time feedback and full SDLC integration, while boasting fast scanning and accurate results in the context of matching on the open source package being used. When we use an open source component, this second-generation technology assumes we use the entire referenced open source codebase/package. This method of identifying referenced code produces an entire list of all the vulnerabilities associated with a component even when it does not affect our code. Just as first-generation SCA solutions were prone to false positives, this second-generation solution is unable to effectively and granularly identify which components are being executed. While the second-generation solution does a great job of identifying the open source components in a project, it does not typically have the capacity or insight into where or how a given component is being used and to what extent.

To address this issue, WhiteSource has developed the third generation of SCA technology, focused on actual and effective open source usage. It understands the way a user consumes open source components within the application and correctly maps that usage to vulnerabilities. The technology identifies all vulnerable functionalities and then provides insights about whether the application is actually making calls to the affected code, thereby making the vulnerability an "effective" vulnerability. The value of this technology is evident: Security teams are able to provide development teams with the evidence they require to address and prioritize the effective vulnerable code, greatly reducing the time associated with research, updates and remediation.



Software Composition Analysis

This new technology, Effective Usage Analysis, changes how SCA tools provide information on *what* you use to *how* you use it. It is well-documented that security professionals are bombarded with security vulnerabilities alerts. This technology enables security teams to reduce the number of vulnerabilities that require response by going beyond noting that code presents a vulnerability to indicating whether that vulnerability is called. From there, Effective Usage Analysis prioritizes effective vulnerabilities, saving time and enabling security professionals and developers, alike, to focus their energies on relevant, impactful vulnerabilities.

The goal is no longer to match 100 percent on a given reference to a package but rather to identify which functions in a given library an application is actually using. This approach provides insights into the impact of each vulnerability on the product's risk. This shift in accuracy of output produces actionable information to best determine how to prioritize tasks and workload.

When a vulnerability is discovered and the vendor or developer releases an update to address it, most security professionals wrongly believe they can just update the code in their applications and move on. Such an oversimplification does not do justice to the level of effort required to find the sections of code that need to be fixed, address backward compatibility, perform unit testing, and complete QA—let alone the weeks of time potentially associated with replacing or updating the affected code. Instead of dedicating the cycles to pursuing this complex path, we could bypass or comment out the calls to the vulnerable code, which would not require rebuilding the entire application. Having the capability to focus on what parts of a vulnerable library an application is actually using gives the power back to the security and development teams to prioritize workflows without the risk of increasing backlog and wasting cycles remediating a vulnerability that poses no security risk to a project.

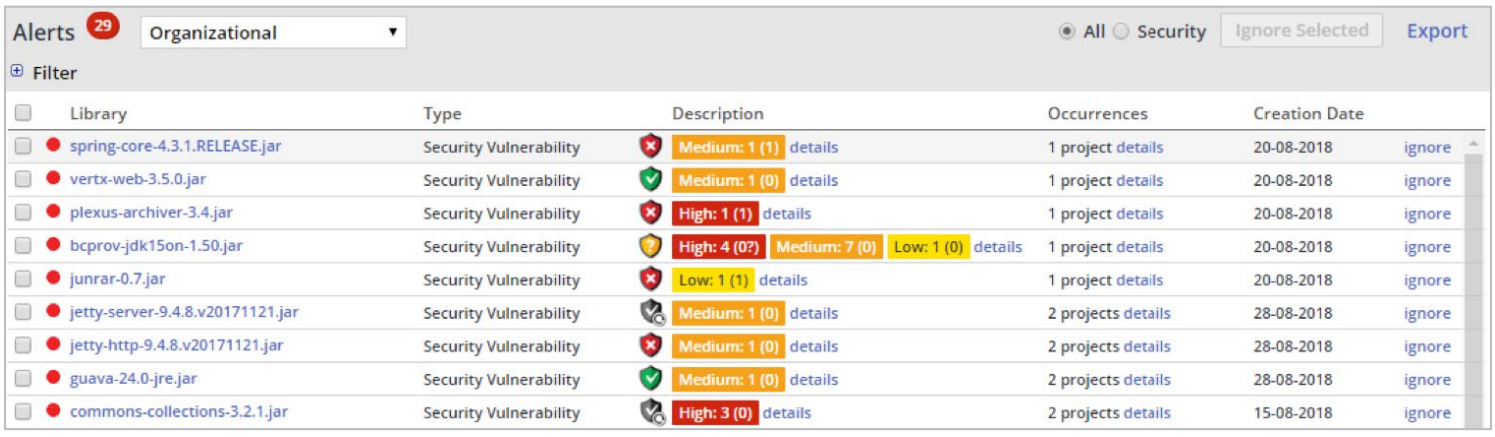
Source Code Analysis

The implementation of this new technology in the WhiteSource solution is configured via parameters (**`enableImpactAnalysis=true`**) in the File System Agent configuration file. This portion of the WhiteSource technology relies on source code analysis to produce output but is integrated with the current solution. In order for the Effective Usage Analysis technology to understand what components of code are being used, it must actually scan the codebase. (Unlike the hashing process, here it actually performs a traditional scan).

Performing this code analysis can be slow. Effective Usage Analysis technology overcomes this problem by leveraging the base technology to pre-form more robust scanning only on previously identified vulnerable components. In this way, the technology has no need to scan all code, resulting in a potential delay in processing. Instead, it builds upon the underlying technology to focus on producing relevant output per the previously identified open source components. This method of scanning would likely scale well to large projects without negatively affecting speed.

Once configured for your needs, the Effective Usage Analysis output is prominent in several sections of the dashboard, including the Home, Product and Project main screens and within Alerts and under Security. Figure 12 shows the iconic shields of the Effective Usage Analysis in the reporting dashboard.

Figure 12. Effective Usage Analysis in Reporting Section



Aside from the File System Agent configuration required for integration, users don't have to make any other changes to make use of the technology. The Effective Usage Analysis technology presents varying color shields in the context of the displayed scope:

- **Green** No effective references to vulnerable code are identified in a given component.
- **Yellow** Analysis could not establish effective references of reported vulnerable code.
- **Red** Effective references from code can be mapped back to the identified vulnerabilities reported for a component.
- **Grey** The technology identifies that the latest analysis of the library may be outdated.

This output guides security efforts in focusing on the vulnerabilities. Figure 13 shows the entire Effective Usage Analysis dashboard.

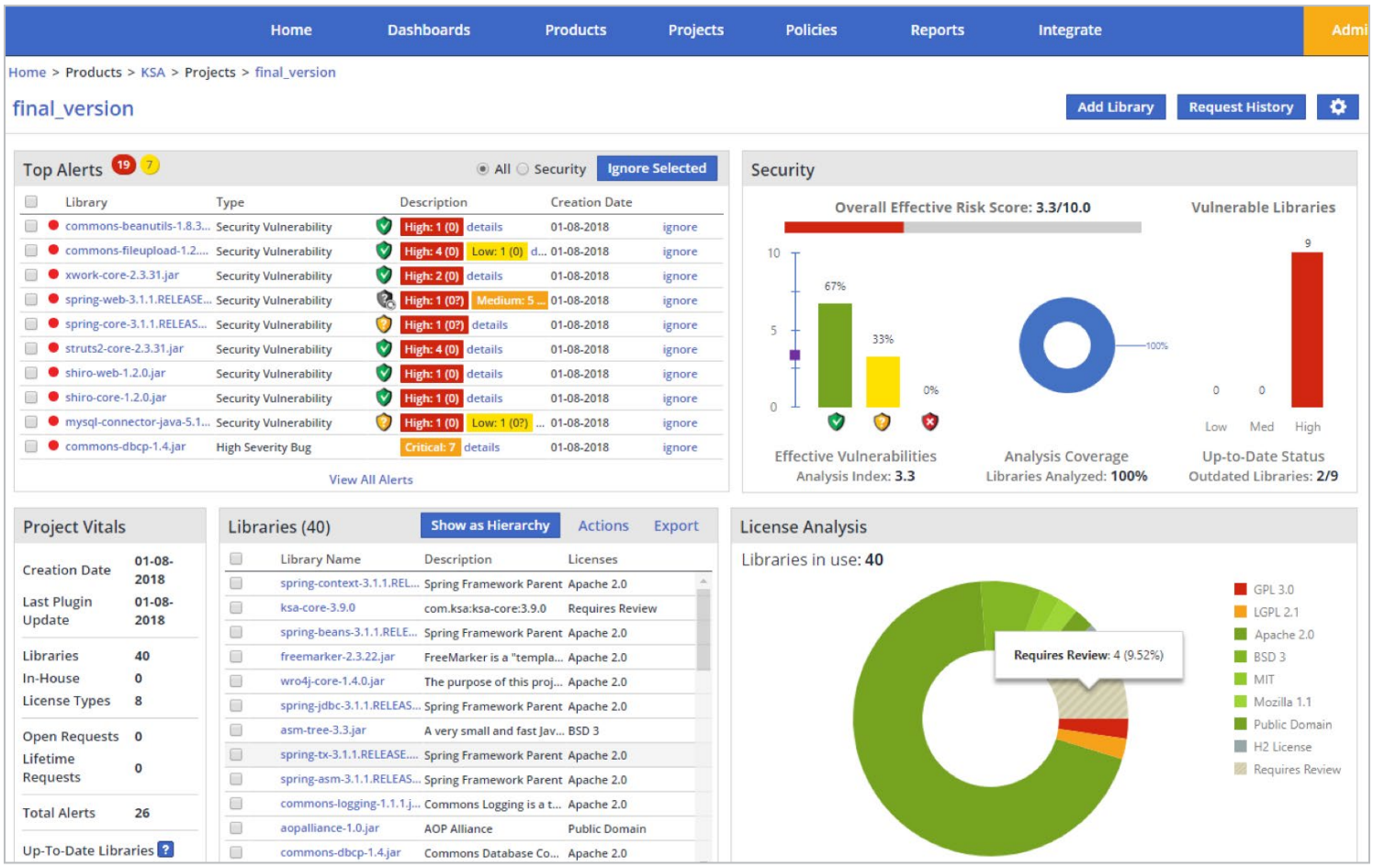


Figure 13. Dashboard View of Effective Usage Analysis

The Effective Usage Analysis report helps security professionals quickly ascertain risk and allows for some human logic to prioritize code changes. For instance, viewing the severity of the vulnerable library provides a glimpse of the risk, but the shield color integrated in this view helps us understand whether the vulnerable library is effective and ultimately, what the real risk is to our project and users. Moreover, we can prioritize what to address based on the criticality (CVSS score) of

the vulnerability or the sensitivity of the application. Figure 14 shows the Effective Usage Analysis shield indicators displaying mostly green icons, denoting that the vulnerable code is not actually effective in the given application/library.

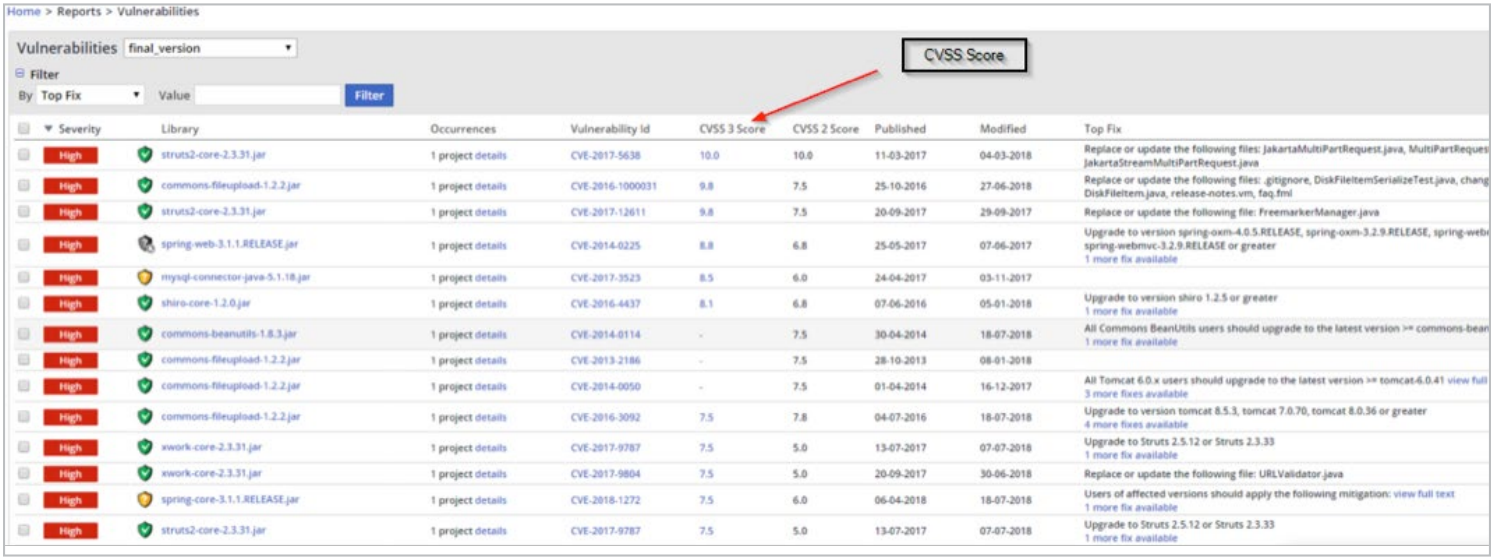


Figure 14. Effective Usage Analysis Shields in a Vulnerabilities Report

Users can look at Effective Usage Analysis output from various perspectives—Organization, Product and Project—by selecting the appropriate dashboard to view. Leveraging this new technology in this capacity enables teams to prioritize fixes and manage time constraints. Most teams will view the output on a product level, to address effective vulnerabilities based on the product's criticality (see Figure 15).

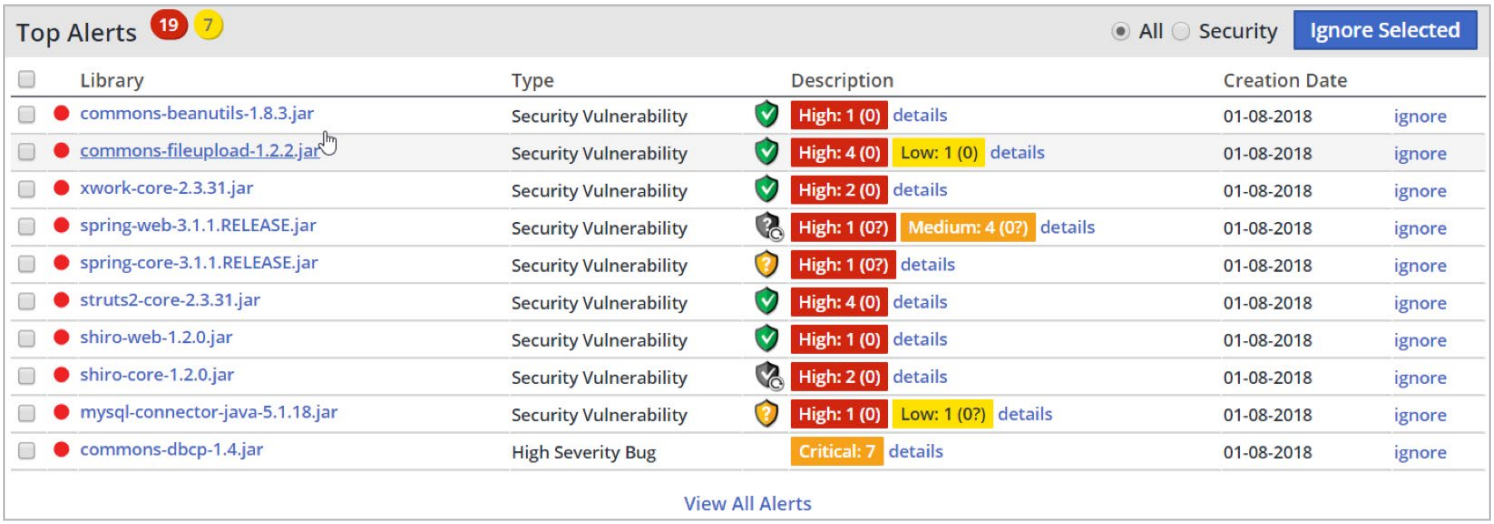


Figure 15. Effective Usage Analysis Product-Level Perspective

Effective Usage Analysis clearly depicts where references to vulnerable code reside in a given library. If no vulnerability exists, then there is no attack surface, which means there is not a known path to application exploitation. Manually performing the research to come to this conclusion can be time-intensive for security teams. But with WhiteSource's Effective Usage Analysis, determining whether a vulnerable component needs to be addressed is aided by component and code verification, yielding a level of confidence that the application is not subject to the identified vulnerability. The amount of time to manually complete such research would be significant, although the focus may be less on vulnerability validation and more on trying to understand transitive dependencies and how to patch code without breaking functionality.

Trace Analysis

Developers often struggle to research open source vulnerabilities because they may not be familiar with certain codes or know how those codes use all the transitive dependencies. WhiteSource's full trace analysis helps identify alternative remediation options, such as changing methods or blocking the vulnerable functions. This aspect of the new technology provides security professionals and developers with the exact path from their code to the vulnerable function.

Development and security teams can use the trace analysis to pinpoint the path to vulnerabilities. When utilized, the full stack trace is referenced, including a path to the impacted file, the class (or applicable point of reference depending on the language) and the entry point to the open source code for clear and actionable information. See Figure 16.

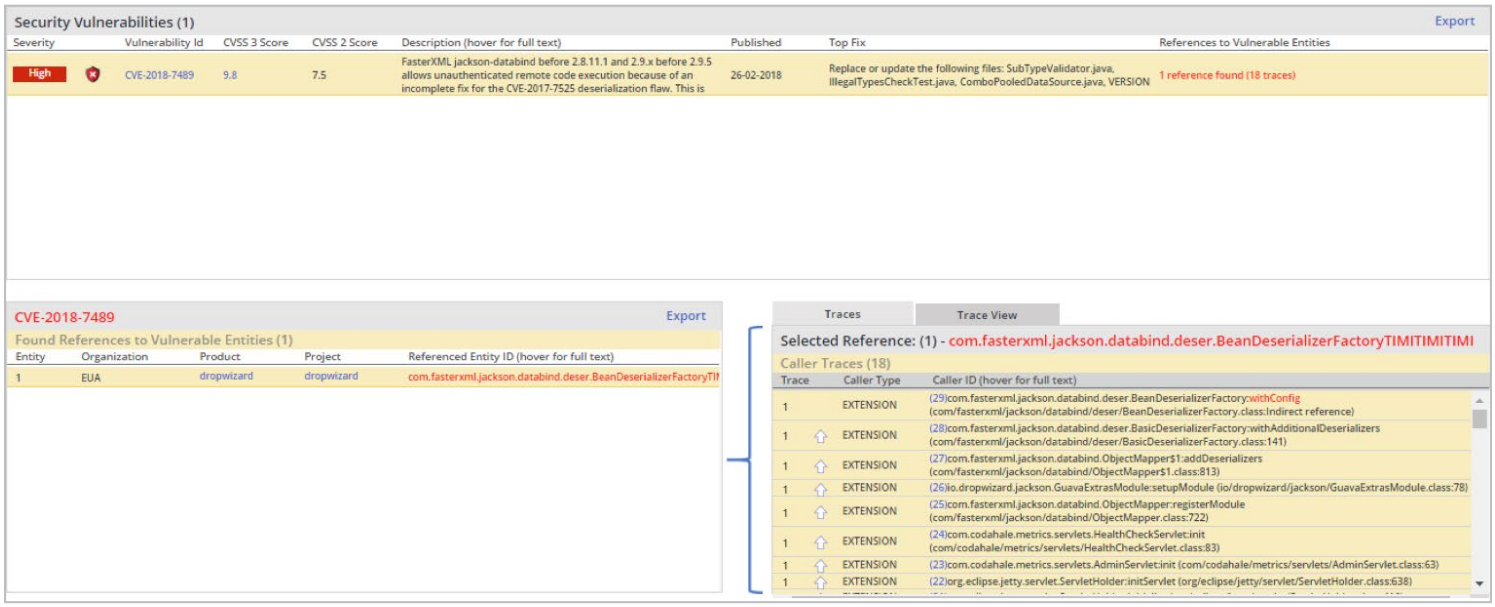


Figure 16. Trace Analysis Output

The trace analysis associated with the red shield provides developers with a quick path to discovering the affected component. Logging is an essential part of debugging, and the trace view depicts the output of the Effective Usage Analysis technology and provides a log or reference point to what component is affected—down to the line of referenced code. This granular information also includes the dependency path to research in order to address the vulnerability. Ultimately this trace analysis guides the developer to the issues while providing the most pertinent information in a clean interface. Security personnel also now have detailed information to delve into the finding and gain a deeper understanding of the root of the vulnerability, which helps with research efforts and potentially even understanding the risk of the flaw.

Because Effective Usage Analysis performs source code analysis on only the vulnerable components of our codebase, it performs quickly. It also eliminates false positives due to exact hashing on open source components. The technology is integrated with the WhiteSource dashboard and requires a simple confirmation change to make use of it. The visual indicators on the dashboard guide security and development teams to focus on only the vulnerabilities affecting the product, reducing wasted time.

Conclusion

In addition to thorough SDLC integration, the WhiteSource solution excels at prioritizing vulnerability remediation by showing affected components or vulnerabilities that development and security teams can use to hone in on the vulnerabilities they need to address first. This prioritization is accomplished via the latest major update to the WhiteSource solution, which is its next-generation software composition analysis feature: Effective Usage Analysis. Other elements that round out the solution as a go-to option include:

- The database containing 300,000+ known vulnerable components
- Out-of-the-box reporting functionality
- Support for more than 200 programming languages
- Strong commitment to the open source community
- Forward-looking technology

The WhiteSource solution is a boon to security professionals and integral as part of a robust vulnerability remediation program. The world's reliance on open source software seems to be ever-increasing, and when the next Heartbleed or Apache Struts (Equifax) magnitude of vulnerability is publicly released, the focus on open source security afforded by the WhiteSource solution could prove to be the difference between exploitation and weathering the storm. In addition, the solution possesses great viability in the open source community as a tool to increase efficiency and make developers' lives easier.

About the Authoring Team

Serge Borso, a SANS community instructor and analyst, teaches the Defending Web Applications Security Essentials and Web Application Penetration Testing and Ethical Hacking courses for SANS. As owner and principal consultant of an information security organization, he leads penetration-testing engagements and has helped dozens of organizations improve their security posture. Serge's accomplishments include developing vulnerability management programs, creating security awareness training solutions and implementing a biometric security system for online banking. An active member in the InfoSec community, he serves on the board of directors of the large, active Denver chapter of Open Web Application Security Project (OWASP). Serge holds several security certifications, including CISSP, GPEN, GCFA and GWAPT.

Dave Shackleford (advisor), a SANS analyst, instructor, course author, GIAC technical director and member of the board of directors for the SANS Technology Institute, is the founder and principal consultant with Voodoo Security. He has consulted with hundreds of organizations in the areas of security, regulatory compliance, and network architecture and engineering. A VMware vExpert, Dave has extensive experience designing and configuring secure virtualized infrastructures. He previously worked as chief security officer for Configuresoft and CTO for the Center for Internet Security. Dave currently helps lead the Atlanta chapter of the Cloud Security Alliance.

Sponsor

SANS would like to thank this paper's sponsor: