Sonatype

PRECISION MATTERS

Why precise component intelligence is critical to automating modern software supply chains

Java, npm, JavaScript, .NET, Python, RubyGems (and beyond)

BY WAYNE JACKSON

CEO, SONATYPE, INC.

TABLE OF **CONTENTS**

Introduction

We don't build software like we used to.

In the days of development past, we built software the hard way. We either wrote it from scratch or copied and pasted code that was often impossible to compile.

Today, developers assemble software using processes that were born in modern manufacturing (lean, agile, kaizen, etc.). According to studies, 80% of a typical application consists of open source and other third party components.

"The answer is Software Supply Chain Automation - the application of principles that enabled the transformative advances at manufacturing pioneers like Toyota."

Sonatype helped pioneer this transformation by serving as a core contributor on the team that created Apache Maven, the world's most popular, component-oriented build tool. Sonatype also serves as the custodians of the Central Repository, the world's primary collection of open source Java components.

With the benefit of these experiences we were inspired to create the Nexus Repository Manager, the industry's first caching proxy for Java, which today helps more than 125,000 development teams organize, store, and manage their open source components and projects.

The combination of these innovations led to a remarkable proliferation of open source and open source adoption. Developers around the world gained easy access to a massive volume and variety of software 'parts', leveraging innovation that transformed the rate at which software could be produced.

But what about quality, maintainability, consistency, supportability, stability, scalability and other factors? How do we satisfy these aims while at the same time improving developer productivity?

The answer is Software Supply Chain Automation - the application of principles that enabled the transformative advances at manufacturing pioneers like Toyota.

This realization inspired Sonatype to study the principles of supply chain pioneers, most prominently W. Edwards Deming. His teachings led us to ponder the possibility that we could optimize software production in the

same way that Deming optimized traditional manufacturing.

Deming taught that a supply chain process could maximize speed and quality by doing three essential things:

- Maintain fewer and better relationships with suppliers
- Procure only the best components from those suppliers
- Track and trace the precise location of every component throughout the supply chain

In order to apply these principles to the software supply chain, we knew that we would have to realize several core goals:

- **Empower innovation** by equipping teams with the ability to precisely identify the highest quality open source components.
- **Scale fast** with component intelligence that is precise enough to enable automation at every phase of the software lifecycle.
- **Control component usage** with flexible policies that can promote granular decision support across varying teams, languages, and application profiles.

In pursuing these goals, we quickly learned that the core, most fundamental requirement was the ability to very precisely establish the identity of a given component.

Without precise identification, associating attributes that describe the nature of the component would be guesses, resolvable only by human analysis, making it impossible to empower, scale and control the software supply chain.

Beginning with the Java ecosystem, our data services team pioneered (and patented) Advanced Binary Fingerprinting.

Now, we are introducing Advanced Binary Fingerprinting for other ecosystems including npm, JavaScript, .NET, Python, and RubyGems.

Over the next few pages, we'll explain how our Advanced Binary Fingerprinting uniquely automates the software supply chain and accelerates innovation while also minimizing risk.

Exploring the Challenges

Until recently, component risks were most often ignored or accepted as a cost of doing business. Interfering with developers in any way meant unacceptable roadblocks to new features, new applications, and new markets.

But now, these risks are more widely known and understood. Of course, so are the long term consequences of poor hygiene and exploding maintenance debt - consequences that threaten innovation, and budgets, over time.

As organizations come to grips with the volume, variety, and velocity of open source components, as well as the challenges of managing a software supply chain, it is critical to answer the following questions:

- Where are components coming from?
- What components represent potential risk to the organization?
- Are any of these components used in current development?
- Do any applications in production include high risk components?

Behind these simple questions lie some remarkable complications. In fact, on closer inspection of the component ecosystems, the answers to these questions can quickly seem overwhelming.

For example, if you're developing Java applications, there are more than 1.36 million components in the Central Repository alone [State of the Software Supply Chain Report, Sonatype 2016], and that number is growing rapidly:

- More than 1,000 new projects added every day
- Approximately 10,000 new versions of project components added every day
- Companies requested these components more than 31 billion times in 2015 alone
- The typical organization uses more than 269,000 components per

year

As daunting as this might be, the Java ecosystem is considered to be relatively well structured and orderly. Dependencies are declared through hierarchies, there is a consistently applied coordinate system (group-artifact-version), and repositories are shared through reliable repositories like Maven Central.

Moving beyond Java, into other languages and ecosystems, we find much less structure and organization.

Using JavaScript as an example, there are:

- 43 million JavaScript files (and growing)
- Roughly 6 million unique components
- Relatively no hierarchical structures – dependencies are generally embedded, not linked

If the above doesn't clarify the issue, take a look at just one example: jQuery. This component has been embedded (and often modified and renamed) in 72,000 different packages. In this figurative jungle, how do you know which jQuery is the definitive one? Which files not named jQuery actually are jQuery? And which version of jQuery are you analyzing?

Only when these answers are known is it possible to empower developers with actionable metadata so real-time judgements can be made about the acceptability of a given, specific component. In other words, informative component intelligence is essential, however it must be paired with precise identification as well.

Alternatives for Identification

Before we proceed further, it is important to understand the various approaches that are commonly used to examine and identify components. It is also important to understand why these approaches are not precise enough to empower, automate and scale modern software supply chains.

File Name - components are identified by evaluating file names alphanumerically.

Why this approach is not sufficiently precise:

- **False positives** - filenames such as util.jar or core.js are very common (like looking in a phone book and trying to identify a specific person with last name of Smith).
- **False negatives** – files are constantly renamed, especially with JavaScript, and therefore components evade comparison results.
- **Incomplete** - while naming will identify struts.jar or jquery.js (two popular components), it will not reveal the specific version of the component.

File Hash - components are identified by examining a unique identifier called the hash or checksum.

Why this approach is not sufficiently precise:

- **False negatives (unknowns)** - Developers often recompile components and regularly post-process (minify, compress) files. Any change, no matter how minor, will result in a different hash that precludes identification by this method.

NAMESPACE / METADATA - components are identified by looking at packaging instructions in files or manifests (e.g. pom.xml - Apache Maven, build.gradle - Gradle, package.json - npm).

Why this approach is not sufficiently precise:

- **False Positives -** Duplication of coordinates is a common problem. Just because the package manager thinks it has a file with the right namespace, doesn't mean it's not an altered one, from another repo.
- **Unknowns -** packaging systems often insert components that are unable to be identified.
- **Multiples -** multiple packaging systems are commonly used which simply compounds the issues above.

Source Code - components identified by scanning source code to find fingerprints that match other known (inventoried) source code (code snippets).

Why this approach is not sufficiently precise:

- **Availability** - when source code is not available to scan, it is (obviously) impossible to identify the code and its associations.
- **False positives** - source code matching is based on snippet analysis which results in numerous false positives. This is because common coding patterns exist, making different pieces of source code appear similar.
- **Redundant positives** – a given piece of source code, especially in open source, often exists in multiple projects and, within a project, likely exists in many versions of a component.
- **Speed** - source code scanning is a slow process and does not provide feedback fast enough to align with the requirements of the modern software supply chain.

In summary, all of the aforementioned approaches are prone to errors and none of them are able to precisely identify open source components at supply chain scale.

Innovation in Supply Chain Data Science

The inability of other approaches to precisely identify Java components led Sonatype to invest the necessary time and capital to invent Advanced Binary Fingerprinting. Recognizing that precision is equally important (and maybe more so) in other ecosystems, Sonatype committed to extending that invention through the creation of new techniques, processes, and algorithms in data science.

The three goals of this effort were to produce:

- A definitive database of components
- A definitive database of component versions
- A definitive coordinate system for components and versions that could be used to reference component metadata (intelligence) for reliable decision making, primarily through automation

After countless hours of research, iteration, and refinement, we have realized our goal – precise identification of components in diverse ecosystems. The conceptual methods used are similar to DNA sequencing of the human genome. The underlying algorithms alone required 40 trillion iterations!

On a practical level, here are two examples of the result:

Java

An application is shipped as a Java Enterprise Archive (EAR) artifact. The EAR contains three Java web applications (WAR). Each of these contain numerous JAR files, including one with a Java JAR file created by patching (forking) an open source project and recompiling it to create a new custom JAR. With Sonatype Advanced Binary Matching, we can identify all nested files successfully. In addition, our patented algorithm will trigger the similarity result and indicate a close partial match to the forked open source including the project version.

JavaScript

An application is shipped as a self-extracting zip file. The file contains a number of JavaScript files, including one with a dependency for jQuery. However, this file was renamed, and has therefore lost any identifying information to the jQuery dependency. Using Sonatype's proprietary matching, the component is identified, and traced back to the original jQuery dependency, including the specific version.

Precise Metadata

Once you have the ability to precisely identify components, the next step toward true software supply chain automation is assuring that the metadata that describes the attributes of a component are sufficiently precise. This allows decisions about a component's acceptability to be adjudicated through automated processes.

As with component identification, searching through various databases for metadata such as security disclosures by the name of a component will inevitably produce poor information. Recognizing this, Sonatype invests heavily in a team of experts that conduct proprietary research to develop deep intelligence on components. These experts leverage public and private data feeds, mine popular repositories (like GitHub), and monitor project web sites, but never rely on those information sources without thorough curation.

For licensing, that means not just relying on the license declarations made by the project but also examining headers within the source code. For security defects, that means finding the root algorithm flaw and documenting paths to remediation or alternative resolution. For other attributes, that means both quantitative and qualitative assessments of architectural and project integrity.

Through our stewardship of Maven Central and partnerships with other public repositories like NuGet Gallery and Npmjs.org, Sonatype actively catalogues all publicly available component resources. Furthermore, our unique platform allows us to reference and validate public data with data from more than 125,000 installs of Nexus Repository.

Enabling Supply Chain Automation

Precision is the only way to empower teams to make better decisions, so that they can scale faster with controls that are flexible enough to reflect the policies of the organization in the context of the applications that are being developed.

Every organization, every team, and every application is fundamentally unique. Therefore, you need to ask yourself the following question: at what point in the software lifecycle do you want to examine the attributes of a component?

- Every time a developer downloads a new component?
- Whenever a development team produces a build?
- During pre-release testing?
- When applications are ready for production?

Which is best?

The answer, of course, is that all of them are important because the world of modern software development is not one size fits all. Indeed, every situation is different and therefore it is critical to have component intelligence tools that are flexible enough to add value at every stage of the DevOps tool chain. This includes the developer IDE, build systems, repository managers, CI/CD tools, image constructors, delivery orchestrators, and production runtime environments.

After deployment, preserving a bill of materials and monitoring the ecosystem for new information, such as the discovery of a new security defect, is also imperative.

In Summary

When it comes to using open source components to manufacture modern software,

the bottom line is this – precise intelligence is critical. Tools that lack precision cannot scale to the needs of modern software development. Inaccurate and/or incomplete data will leave organizations to deal with vulnerabilities, licensing, and other quality issues that lead directly to higher costs and reduced innovation.

Sonatype Data Services and Advanced Binary Fingerprinting power a unique solution that enables organizations to:

- **Empower innovation** by equipping teams with the ability to precisely identify the highest quality open source components.
- **Scale fast** with component intelligence that is precise enough to enable automation at every phase of the software lifecycle.
- **Control component usage** with flexible policies that can promote granular decision support across varying teams, languages, and application profiles.

With precise identification on your side, you have the power to error-proof the software supply chain. This means eliminating, with certainty, the risks and inefficiencies that diminish innovation. This also means unlocking the full potential of talented developers so you can innovate faster and compete more effectively on a global playing field.

We welcome any questions that you might have and we encourage you to sample the incredible value of Sonatype Data Services and Advanced Binary Fingerprinting. To take the next step, try our free tool, Application Health Check, or schedule a demo today.

Sonatype helps organizations build better software, even faster. Like a traditional supply chain, software applications are built by assembling open source and third party components streaming in from a wide variety of public and internal sources. While re-use is far faster than custom code, the flow of components into and through an organization remains complex and inefficient. Sonatype's Nexus solutions apply proven supply chain principles to increase speed, efficiency and quality by optimizing the component supply chain. Sonatype has been on the forefront of creating tools to improve developer efficiency and quality since the inception of the Central Repository and Apache Maven in 2001, and the company continues to serve as the steward of the Central Repository serving 17.2 Billion component download requests in 2014 alone. Sonatype is privately held with investments from New Enterprise Associates (NEA), Accel Partners, Bay Partners, Hummer Winblad Venture Partners and Morgenthaler Ventures. Visit: www.sonatype.com

Sonatype Inc. • 8161 Maple Lawn Blvd, Suite 250 • Fulton, MD 20759 • 1.877.866.2836 • www.sonatype.com