# Open Source and the Software Supply Chain

## A Look at Risks vs. Rewards

**Wayne Jackson, Sonatype**

**Abstract**. There is a dynamic shift occurring in the software development landscape. No longer are applications written, today most are assembled using open source components. The growing reliance on externally sourced, open-source components as core building blocks for modern application development, coupled with the complexity of the ecosystem, has ushered in new risks for the software supply chain.

This article will explore the licensing, security, and quality risks associated with component-based development and its direct impact on the integrity of the software supply chain.

### Introduction

For most of its history, software has been written—applications consisted primarily of custom developed code and internally developed components with only a small fraction of code sourced from outside the organization. Development efforts followed a "waterfall" methodology and projects spanned months or even years. The widespread use of cloud-based infrastructures and the rise of open-source technologies during the past decade have heavily influenced the software development landscape with startups and established organizations demanding increased flexibility and improved time to value in the way software is developed and delivered. As a result, modern software development and the resulting software supply chain have become increasingly component-based, where applications are assembled from existing components rather than written from scratch. Enterprise applications today are typically built using 75% to 80% open source components [1], with custom code comprising the rest. So, what does today's software development landscape look like and what are the risks to the software supply chain?

| Software Development Once Was… | Software Development Now Is… |
|---|---|
| Waterfall Methodology | Agile Development |
| Code-Based | Component-Based |
| Developed | Assembled |
| Independent | Collaborative |
| Proprietary | Open Source |

*Table 1:*

**First**, modern software development is increasingly component-based. The vast majority of these components are sourced from outside the organization.

**Second**, open source has become an integral part of modern applications. In most cases, externally sourced components are open source. Modern applications often rely on hundreds of open source components and frameworks.

**Third**, development organizations have embraced agile software development processes. The modern development process is rapid, continuous, and collaborative.

While development teams have embraced agile software development processes, the shifting software development landscape has also introduced new risks and requirements in the software supply chain. Applications can be composed of hundreds of components sourced from a myriad of open-source projects and these components can in turn, depend on other components, known as transitive dependencies. This creates an enormously complex software supply chain, where a single application may contain components originally published by dozens of individual projects.

To see just how far reaching externally sourced open source components are in the software supply chain, look no further than the Central Repository, the industry's primary source for open source components. The Central Repository receives 7.5 billion requests annually and is used by more than 60,000 organizations worldwide. Large organizations that rely on custom software for competitive advantage are the biggest consumers of the 400,000 components in the repository, but demand comes from every industry and geography.

Whether provided by commercial vendors or open source initiatives, components can introduce significant management, security and licensing challenges. Think of today's software as being assembled rapidly with a very complex supply chain like that of a car manufacturer. Like a car, the final product (an application) may contain hundreds or thousands of externally sourced components from dozens or hundreds of original suppliers. Each of these components has its own lifecycle, its own bug fixes and feature enhancements, and its own potential risks.

Like a car, a single flawed component could cause significant problems for the user. In the worst case, these problems could lead to security breaches, data leaks, stability, and performance issues, or legal actions related to intellectual property.

An easy example of a potential problem is in the area of security. Recent analysis by Aspect Security, using data from the Central Repository, uncovered widespread security vulnerabilities among the most commonly used open source components.

Often risks are caused by flawed components nested deep in an application's dependency tree where flaws are not easily apparent. Dependencies may allow flawed components to quickly infiltrate and undermine the software supply chain.

Users of the Central Repository regularly consume outdated, flawed or insecure components even years after newer fixed versions are available. An example of this can be seen when the United States Computer Emergency Readiness Team and NIST issued a warning in March 2009 that the Legion of the Bouncy Castle Java Cryptography API artifact was extremely vulnerable to remote attacks. Almost two years later in January 2011, more than 1,500 organizations downloaded the vulnerable version of Bouncy Castle from the Central Repository in a single month [3].

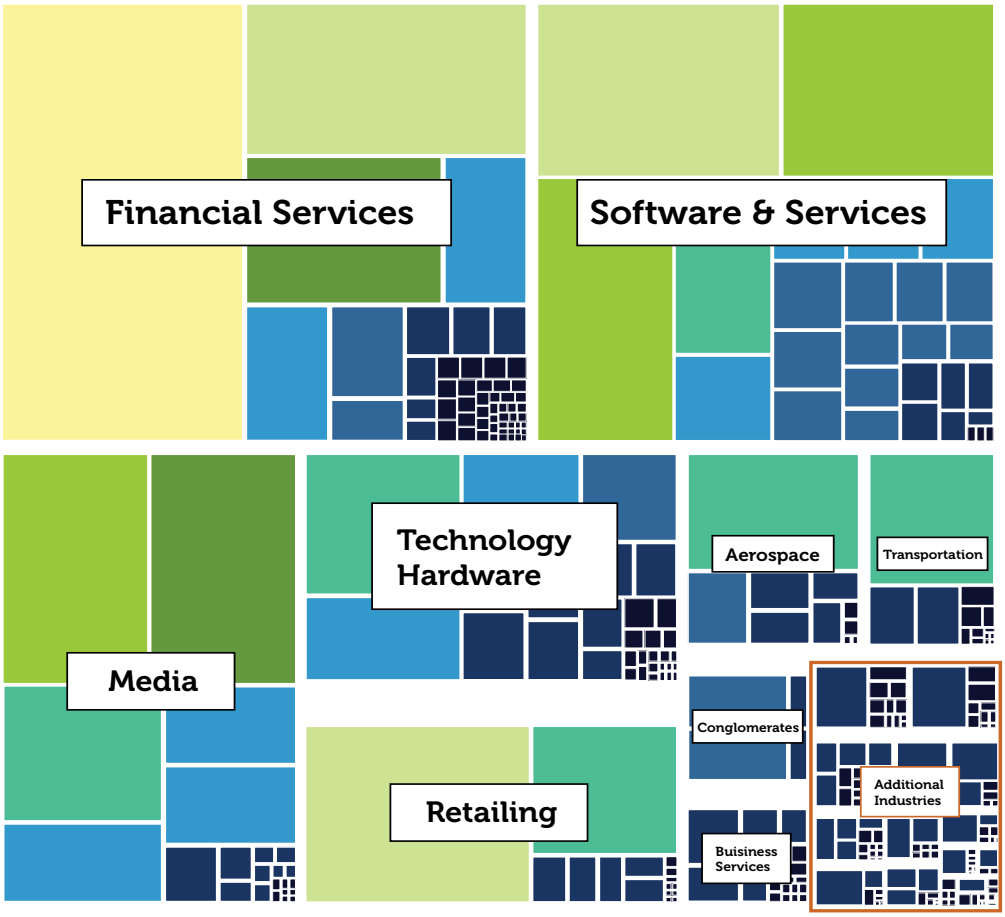


*Figure 1:*



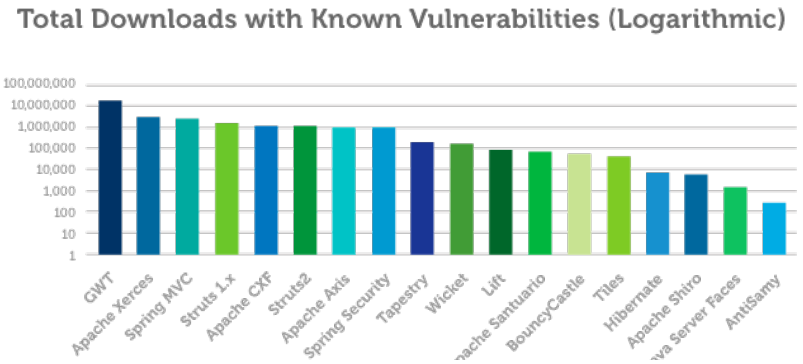


*Figure 2: From reference [1]*

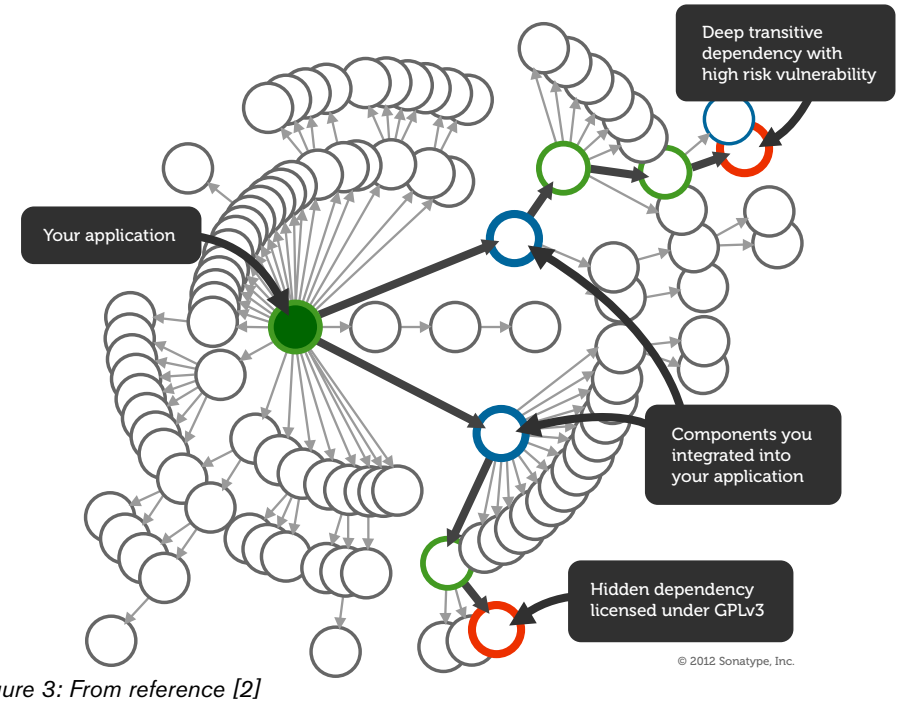


*Figure 3: From reference [2]*

Open source projects innovate rapidly and release frequently. However there is no update notification infrastructure for open source components. Therefore there is no easy way for component consumers to know when a new version has been released, much less which defects have been fixed.

Because open source usage generally occurs under the corporate radar, it is not uncommon for organizations to be unaware of which components are being used in their software supply chain or within key production applications.

Agile software development, incremental deployment and continuous integration have all resulted in many more builds over the life of a software project. Measurements of software quality and risks must be conducted in-band during the development and build process. Development teams are increasingly geographically dispersed and often include external contractors. Keeping disparate teams in sync and enforcing standards is increasingly important to minimize waste and risk.

To firmly establish both control and visibility across today's complex and agile software supply chain, organizations should take the following steps toward Component Lifecycle Management (CLM)—or the practice of proactively managing the use of components throughout the supply chain.

**Step 1: Inventory** – Gather information about your current component usage:

- Track component downloads and usage to understand consumption.
- Inventory internal component repositories to determine what is being distributed to development teams.
- Understand the software supply chain to determine which components and dependencies are being introduced to the organization.

**Step 2: Analyze** – Understand vulnerabilities in applications and repositories:

- Analyze key applications to uncover known security vulnerabilities.
- Analyze internal component repositories to discover vulnerable components.

**Step 3: Control** – Establish controls throughout the development lifecycle:

- Establish policies regarding security, the use of viral licenses and the out-of-date or out-of-version components.
- Eliminate or blacklist known vulnerable components in internal repositories.
- Establish mechanisms to prevent known flawed components from entering the organization.
- Implement controls in build and continuous integration systems to prevent inclusion of flawed components in software builds.

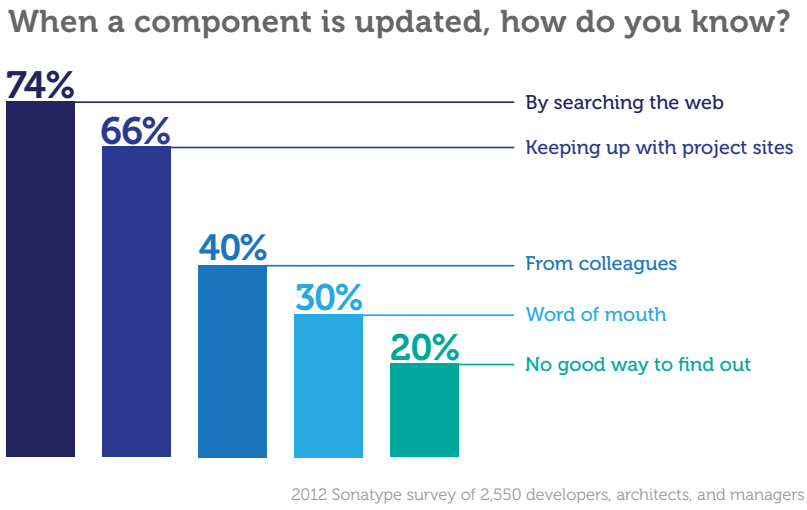


*Figure 4: From reference [4]*

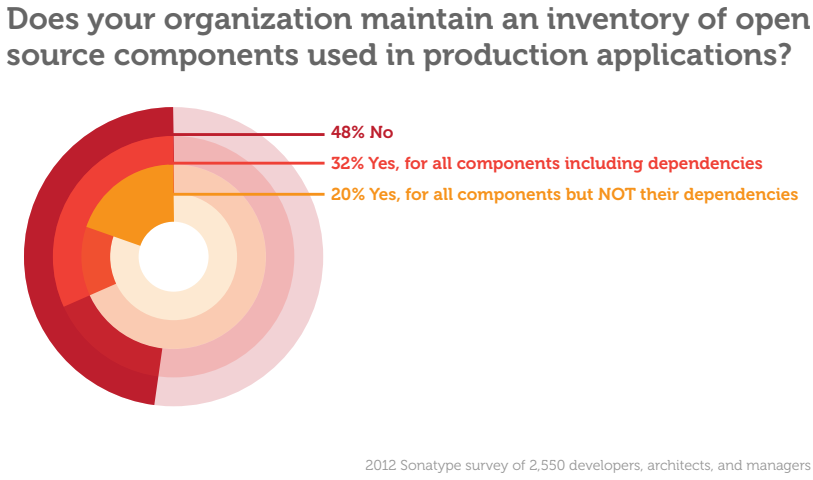


*Figure 5: From reference [4]*

**Step 4: Monitor** – Maintain awareness of component updates:

- Maintain an inventory of all components and dependencies used in production applications.
- Continuously monitor application bill-of-materials for updates and newly discovered vulnerabilities.

Properly managing the use of open source components throughout the software development lifecycle will enable organizations to ensure the integrity of the software supply chain and focus on the cost savings and wealth of innovation open source software can bring.

## ABOUT THE AUTHOR

Wayne Jackson is CEO of Sonatype, a provider of component lifecycle management tools. Prior to joining Sonatype, he was the CEO of open source network security pioneer Sourcefire, Inc., which he guided from fledgling startup through IPO in March of 2007 to a peak valuation of over $750 million. Before joining Sourcefire, Mr. Jackson co-founded Riverbed Technologies, a wireless infrastructure company, and served as its CEO until the sale of the company for approximately $1 billion. While at Riverbed, he built strategic relationships with industry leaders Palm, Oracle, IBM, Symbol and Microsoft, growing the company from startup to category winner in less than two years. Mr. Jackson holds a B.B.S. in Finance from James Madison University and has completed the Executive Education program for Corporate Governance at Harvard University.

**Phone: (301) 684-8080**
**E-mail: wjackson@sonatype.com**

## REFERENCES

1. Aspect Security, "The Unfortunate Reality of Insecure Libraries," March 2012
2. Sonatype Inc., "Executive Brief: Addressing Security Concerns in Open Source Components," March 2012
3. The Central Repository (2012)
4. Sonatype Inc., "2012 Open Source Software Development Survey," April 2012