BLACKDUCK®

Five Considerations for Securing Your Software Supply Chain

As noted in the annual "Open Source Security and Risk Analysis" (OSSRA) report by Black Duck, the use of open source and third-party software continues to increase at a rapid rate. This means that your development team undoubtedly uses open source in some capacity and to some degree within your software development life cycle.

And if your team functions as a provider of software, the open source, third-party, and commercial components you rely on to build your applications make you part of a supply chain. It's important to note, though, that this supply chain doesn't stop with your team—it extends beyond your development and delivery activities and reaches all the way to the end user of your applications.

Your supply chain comprises everything that touches an application or plays a role in its assembly, development, and deployment. It also includes proprietary code and components created by your development team, as well as the infrastructure used to build and deliver that software to your end user.

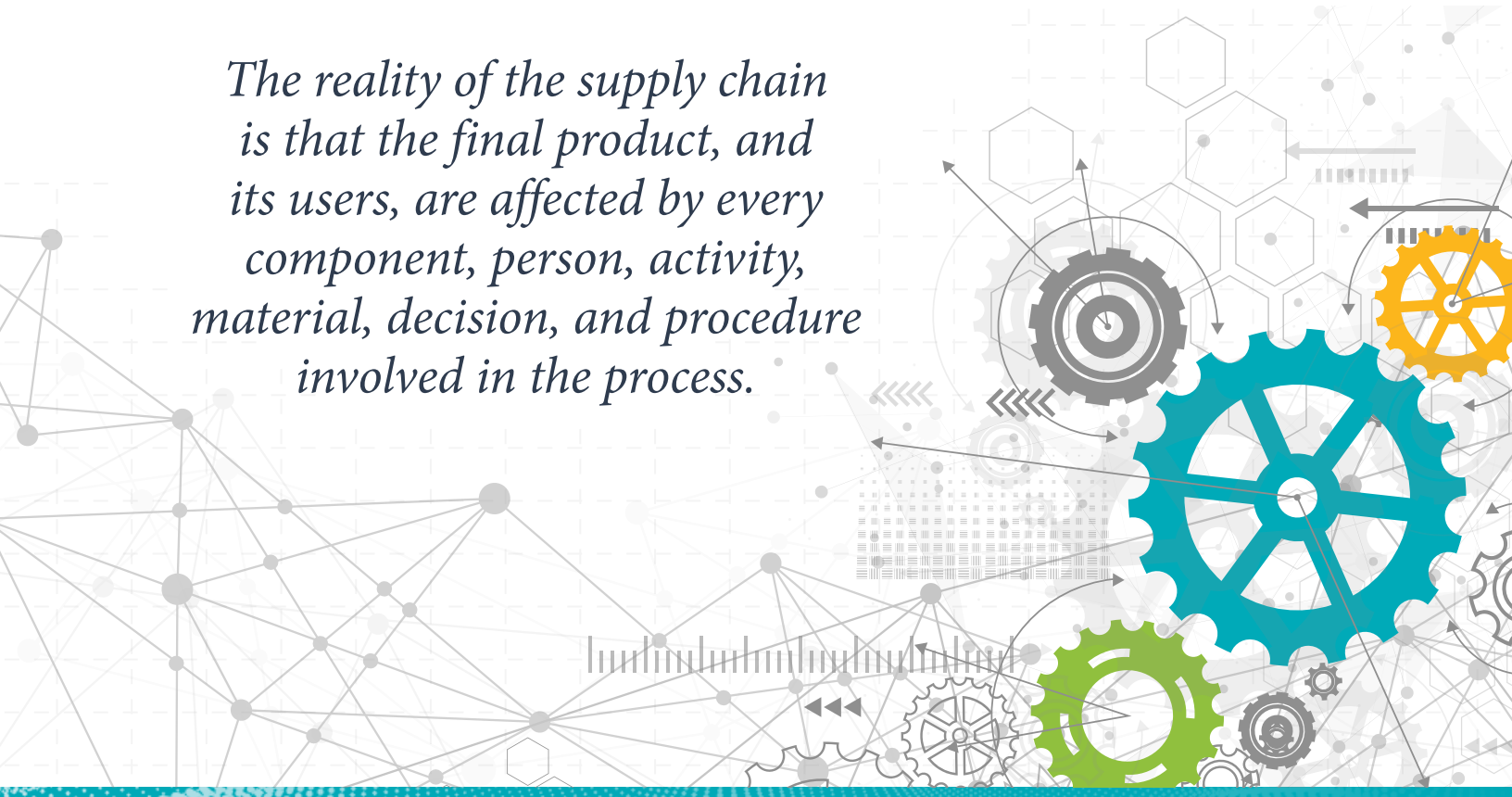
Examining an example of a traditional supply chain can help add clarity to the software supply chain discussion. Consider automobile manufacturing. This industry's supply chain begins with raw materials suppliers, who provide metals, cotton, leather, wood, and so on to parts manufacturers. These manufacturers take the raw materials and produce parts like screws, fabrics, nuts, and bolts. These components are then passed along to automotive parts and systems manufacturers, who fashion the car parts needed to build a vehicle (engines, transmissions). Finally, these parts are sent along to original equipment manufacturers, who build vehicles in an assembly line and ship them for sale to consumers.

When you consider the myriad inputs involved in this manufacturing process, you can begin to understand the many places where risk can be introduced. What is the quality of the parts sourced to build this car? How well did the auto manufacturer assemble these parts? How does this automobile actually perform when the user gets behind the wheel? How does it respond to incidents like accidents and nonideal driving conditions?

The reality of the supply chain is that the final product, and its users, are affected by every component, person, activity, material, decision, and procedure involved in the process. Weaknesses in any part of this workflow introduce risk, and the only way to mitigate this risk is to completely understand the entire supply chain. The same applies to the software supply chain.

In the simplest terms, an organization cannot accurately or completely manage its risk without full visibility into its supply chain. With so many moving parts, all it takes is one unidentified weakness or design flaw to create opportunities for exploit.

The reality of the supply chain is that the final product, and its users, are affected by every component, person, activity, material, decision, and procedure involved in the process.



The software supply chain today

More than three-fifths (61%) of U.S. businesses have been directly impacted by a software supply chain threat over the past year, according to a report from Capterra, an online marketplace vendor owned by analyst firm Gartner. The Capterra report points to open source software as a key source of supply chain risk, and notes that 94% of U.S. companies use open source in some form.

There clearly are effects that insecure open source can have on businesses both inside and outside of the software development industry. For example, a zero-day vulnerability in the widely used Apache Log4J utility allowed attackers to execute arbitrary code on vulnerable servers.

But open source isn't the only risk to supply chain integrity. The SolarWinds breach resulted from a single leaked password. With it, hackers were able to gain access to its systems and leverage routine updates to access thousands of customers' sensitive data. In the CodeCov supply chain breach, hackers used secrets found in Docker images to install a back door and gain access to customers' data.

In response to the uptick in security breaches, President Biden issued an executive order (EO 14028) instructing U.S. agencies to establish guidelines for how software vendors selling to the federal government must secure their software. Aimed at bolstering the United States cybersecurity profile, this order prompted a nationwide re-examination of organizational security practices that stretches well beyond the companies specified in the order.

The order also sparked a collective investigation of software security practices. A key part of EO 14028 is suggested enhancements to the enterprise software supply chain. The order added another layer of consideration to existing software security practices: that organizations scrutinize their existing activities through the lens of supply chain security. The result is increased requirements that must be addressed when outlining security measures.

Key considerations for software supply chain security

On a fundamental level, any effort to secure the supply chain must consider both how to safeguard applications from upstream risk, and how to prevent your organization from generating downstream risk.

In an effort to provide a more concise starting point for tackling supply chain security, Black Duck has identified five areas of consideration that should drive your security activities. Answering these five questions should yield important insights into areas of success and weakness in your supply chain security efforts and inform your security activities.

On a fundamental level, any effort to secure the supply chain must consider both how to safeguard applications from upstream risk,

and how to prevent your organization from generating downstream risk.

Question 1

Is the open source you use secure?

Understanding open source risk

The prevalence of open source is a fundamental concern when addressing supply chain security. While open source is no more or less risky than proprietary code, failure to adequately secure it introduces great risk to your organization's overall security. Your developers very likely use open source in nearly everything they create, and that means large portions of your applications are composed of code that you did not write. The annual OSSRA report confirms this: Nearly 100% of the codebases examined for the report contain open source, and nearly 80% of the code in those codebases is open source.

Open source by definition comes from an uncontrolled source: developers outside your organization. Securing your supply chain means maintaining visibility into everything your applications are composed of, especially when it comes from outside your organization.

It's important to note that developers include open source in their work both intentionally and inadvertently. The same is true for your vendors; their developers may unknowingly incorporate open source into their projects. While this is not inherently a bad thing, it does introduce an avenue for upstream risks to find their way into your applications. It is your responsibility to track open source components, licenses, and vulnerabilities, and their associated risk, in your supply chain. But given the scale of this undertaking, manual tracking is insufficient and far too labor-intensive.

Understanding legal risk

Often less publicized than security risks, legal implications stemming from license violations and conflicts can easily translate to issues with mergers and acquisitions, vendor disputes, and distribution problems. For software vendors and distributors, the legal risk stemming from supply chain weaknesses poses a threat to an organization's reputation and financial security.

Mismanagement of open source is often to blame for legal risks, as the presence of open source can make the identification and monitoring of intellectual property rather murky. Such was the case in the infamous Cisco vs. the Free Software Foundation lawsuit, in which Cisco acquired firmware that contained noncompliant open source software. By inheriting licensing conflicts, Cisco exposed the organization to legal and reputational implications that could have been prevented. This underscores again how everything within the supply chain has the potential to affect an entire operation. It is critical to think of software risk as business risk and treat it accordingly.

It is your responsibility to track open source components, licenses, and vulnerabilities, and their associated risk, in your supply chain.

Understanding operational risk and risk outside of known vulnerabilities

Operational risk can be introduced when teams use outdated components, components with no recent development activity, or components from projects without a sufficient developer community to actively maintain the code. Aside from leading to code quality, reliability, and maintainability issues, operational risk is a gateway to security risk. If there are no developers finding and fixing bugs in a project, there are no developers finding, disclosing, and fixing security defects. This is especially true when leveraging projects with poor or unknown reputation and history. These types of projects yield low-hanging fruit for threat actors.

Risk can also come in the form of a malicious package, which is a package that contains malware disguised as a legitimate package. They are used to infiltrate the software supply chain via open source libraries or third-party dependencies and perform harmful actions once activated. Malicious packages can pose serious risks to the integrity and security of applications. Once a malicious package's malware infects a system, it can potentially steal sensitive data, disable security software, modify or delete files, and even take over an entire system or network, spreading to other devices to further increase its damage.

Question 2

Are you required to provide a Software Bill of Materials?

There are many avenues through which software can enter an application. Intentional and unintentional (i.e., transitive dependencies) additions can enter via

- Package managers
- Copy/paste and AI-generated snippets
- Languages such as C/C++ that lack package managers
- Container images, dynamic link libraries, and other third-party binaries or libraries

All of which creates a broad risk surface and obscured visibility. Without a complete, dynamic view of what's in your applications, neither you, your vendors, nor your consumers can confidently determine what risk you're exposed to. Maintaining a Software Bill of Materials (SBOM) is a best practice and the cornerstone of a successful supply chain security program. It is also sometimes required for regulatory compliance. An SBOM is essentially an "ingredients list" of your codebase, and it can be the key to establishing visibility, but to be accurate and complete, it needs to be compiled at the right times.

It's not enough to simply analyze package managers for dependencies, and it's not enough to rely on SBOMs that were compiled days, weeks, or months ago. For an SBOM to be effective, it needs to be accurately populated with the dependencies buried in code, regardless of language, dependency type, or version. And it can't stop at open source; custom and commercial code needs to be included and tracked as well.

Take the Apache Log4J zero-day vulnerability as an example; organizations with up-to-date SBOMs were able to determine their exposure and begin mitigation activities within hours of disclosure, without the need to go back and scan their entire application portfolio.

Without a complete, dynamic view of what's in your applications, neither you, your vendors, nor your consumers can confidently determine what risk you're exposed to.

Question 3

Is the code you write secure?

Since open source represents the majority of application code, it's no surprise that it also represents the majority of the attack surface. However, it's still crucial to ensure that the code your developers write protects sensitive data and systems from cyberattacks.

Security defects and weaknesses that are inadvertently coded into applications open the door to attacks such as buffer overflows, SQL injection, and cross-site scripting. Should a system breach occur, these security defects leave sensitive data vulnerable to exposure. A threat actor could exploit these weaknesses to inject malicious code, and then have an avenue to infiltrate the systems maintained by the organization operating the software.

Efforts to avoid these issues usually include code reviews in hopes that a few more sets of eyes on source code will help identify issues. However, most software developers are not trained in secure coding, and many security flaws are far too complex to spot and trace during manual code review. Static analysis solutions that can automatically review code execution and data paths as well as identify security weaknesses are essential tools in ensuring that you can trust that you're not introducing any downstream risk.

That said, organizations should always be focused on continuously improving the security of their products. And the best way to avoid security weaknesses is to never create them in the first place. Organizations should invest in training their developers on how to code securely, and create security champions to lead the charge in instilling a culture of supply chain security.

Security defects and weaknesses that are inadvertently coded into applications open the door to attacks such as buffer overflows, SQL injection, and cross-site scripting.

Question 4

Is your development and delivery infrastructure secure?

Data storage needs are growing, deployment deadlines are getting shorter, and rapid scalability has never been more important. In response, the software industry has become increasingly dependent on cloud technology to power its applications. Part of this cloud-native approach means adopting application deployment methods that support scalability and agility—which containerization and infrastructure-as-code (IaC) do well.

Identifying what software is packaged into containers

Containers make it easy to rapidly deploy, patch, and scale an application or microservice. They also ensure consistent performance across multiple different operating systems and hardware platforms. This makes containerization an ideal solution for a cloud-native approach—but it also introduces additional avenues for threats to enter the supply chain.

The most common approach developers take when containerizing applications is to start with a base image, often open source, and build on top of it. Layers of additional third-party and custom code are added on top of this base image, and then combined into a single filesystem, represented as a binary file, when the final image is being executed. Basic software composition analysis tools assume that files in a layer are added via a package manager such as YUM, which is what the tool will use to determine composition.

However, that is not completely valid. When files are added to layers using certain commands in a Dockerfile, such as ADD, COPY, or RUN, it is done without the use of a package manager. Only a true binary analysis of the container image can inspect component signatures to identify all open source components regardless of their origin, as well as any sensitive data present inside a container image.

Containerization is an ideal solution for a cloud-native approach—but it also introduces additional avenues for threats to enter the supply chain.

Using infrastructure-as-code for deployment securely

Integral to how cloud platforms operate today is the way the supporting infrastructure is configured, managed, and provisioned. To support virtualization and container orchestration, servers now operate in an ephemeral manner, built up by spec, on an as-needed basis. These specs, constructed by teams using software such as Terraform and Ansible, enable operations to take place in an automated manner. However, these specs can also introduce security flaws that can be multiplied at the scale of application deployment, which can range from tens to hundreds of thousands of times per week.

Inadvertent and intentional back doors, created by unauthorized privilege escalations or network exposures in infrastructure configuration, for example, are not a new risk. What's new is who is responsible for building and assembling server configurations. This used to be the job of an IT operations engineer—someone who was trained and experienced in securely configuring servers and anticipating threats. The responsibility has since shifted to development teams, which can use infrastructure-as-code (IaC) to easily specify deployment configurations for the applications they're building. Inexperience in this area can lead to mistakes and oversights that make it easier for attackers to infiltrate the infrastructure—putting attackers one step closer to an application's users, operators, and data.

Part of what makes IaC so valuable is that it is composed of code. As such, it's written and read like code, it's saved and versioned as code, and it can be analyzed and reviewed as code. Just as a static analysis tool should be used to analyze application source code for security weaknesses, it should also be used to scan IaC files to uncover costly configuration mistakes that developers typically aren't trained or experienced enough to look out for.

Question 5

Do you create or consume software within a regulated industry?

Regulators are implementing new standards to help manage the complexities of the software supply chain. SBOMs are becoming compulsory in various industries as a critical component of regulatory compliance to address such risks as

- Outdated components—SBOMs can highlight components that are no longer supported or have known security vulnerabilities
- Unauthorized components—SBOMS can identify components the organization did not approve, potentially posing licensing or security risks
- Noncompliance with regulations—SBOMs can verify that the software complies with relevant security regulations and standards

The U.S. Food and Drug Administration (FDA) is mandating that all medical device manufacturers create an SBOM for all software in a medical device and submit that SBOM for review by the FDA. This rule went into effect in March 2023. Open source software is widely used in medical devices, with Linux being one of the most popular choices for medical device systems. If you're building on Linux, your software likely incorporates other open source components such as middleware and front-end frameworks, all of which must be regularly reported to the FDA through an SBOM.

The U.S. Food and Drug Administration is mandating that all medical device manufacturers create an SBOM for all software in a medical device.

Likewise, while there are no SBOM requirements within the automotive industry, there are many working groups trying to understand what an SBOM would be used for. Several original equipment manufacturers (OEMs) have established SBOM requirements to communicate open source software license requirements within their specific supply chains. According to one study, between 50% and 70% of the automotive software stack is based on open source, all of which must be inventoried in an SBOM.

Integrating security into the software development life cycle with an SSDF assessment

The National Institute of Standards and Technology developed the Secure Software Development Framework (SSDF) to guide development organizations on how to best produce secure software. While not a requirement, the activities contained in the SSDF are becoming key elements informing how legislation is evolving. As an example, the recent CISA Secure Software Development Attestation Form is based on a subset of SSDF activities.

In short, the SSDF is a set of best practices and guidelines aimed at integrating security into the software development life cycle. It provides a structured approach to developing secure software by incorporating security considerations at each stage of the development process.

The SSDF focuses on proactive security measures, promoting the idea of "building security in" (a concept introduced in the BSIMM project) rather than addressing security issues as an afterthought. It emphasizes the importance of considering security requirements, conducting threat modeling, performing secure coding practices, implementing appropriate security testing, and maintaining security throughout the software's life cycle.

What's next

Examining your existing security activities, policies, and procedures in conjunction with the questions posed here will help you identify areas of your supply chain security that are weak and need attention. While supply chain security can seem daunting, breaking it into clear and concise considerations and then tackling identified areas of weakness with the right tools and practices can help you achieve a secure supply chain.

For more information about securing your software supply chain,
see how Black Duck can help you identify and manage risks end-to-end.

About Black Duck

Black Duck® offers the most comprehensive, powerful, and trusted portfolio of application security solutions in the industry. We have an unmatched track record of helping organizations around the world secure their software quickly, integrate security efficiently in their development environments, and safely innovate with new technologies. As the recognized leaders, experts, and innovators in software security, Black Duck has everything you need to build trust in your software. Learn more at www.blackduck.com.