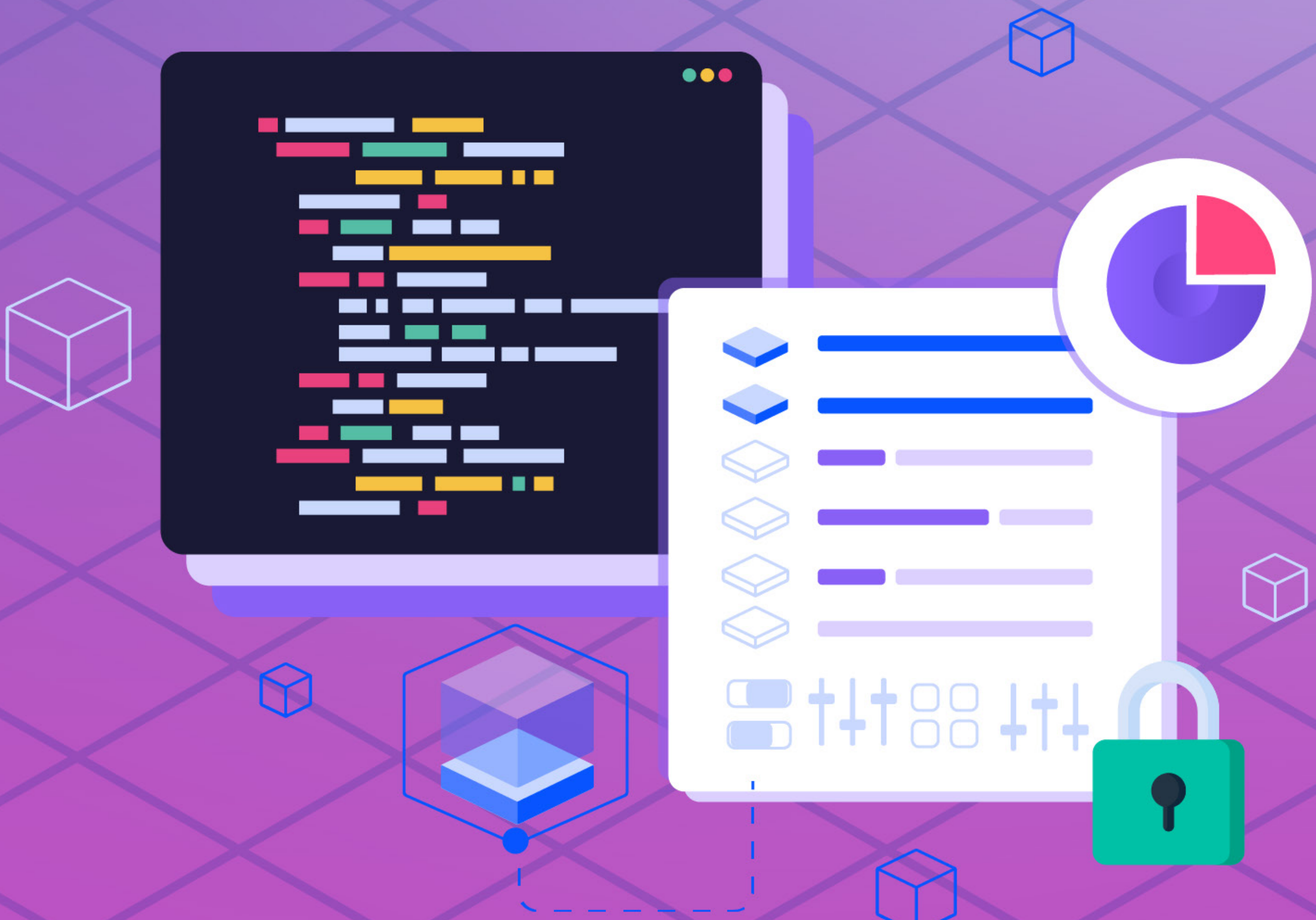
activestate

SOFTWARE SUPPLY CHAIN SECURITY BUYERS GUIDE



Executive Summary

The software supply chain isn't a new concept, but in a post-Solarwinds world, it's become the focal point of efforts to improve cybersecurity in general, and the security of software development in particular. The market result has been a gold rush as traditional software security vendors have quickly positioned themselves as supply chain security vendors. Making head or tail of their claims can be confusing.

The process of importing prebuilt, third-party code into your organization is arguably the weakest link in the software supply chain since it involves trusting thousands of open source developers with whom you have no relationship. But building all open source dependencies from source code (including all transitive dependencies and linked C libraries) is both complex and costly in terms of time and resources.

SOFTWARE SUPPLY CHAIN RISKS INCREASE WITH:

- The number of open source languages/ ecosystems in your development stack
- The number of dependencies you build in a non-reproducible manner
- The fewer codebase updates you perform

A number of traditional and emerging tools discussed in this guide can help mitigate many of the threats inherent in your software supply chain, while reducing cost and complexity.

Introduction

Organizations concerned with the security of their software supply chain often have trouble navigating the ever-expanding labyrinth of open source and commercial software solutions that claim to help. While solutions exist across the entire software supply chain, it can be difficult to determine which vendors are merely "supply-chain-washing" their traditional security tools, and which vendors have created purpose-built tools to solve key software supply chain issues. This guide is designed to provide clarity.

THE SOFTWARE SUPPLY CHAIN EXTENDS FROM:

Imported Code - any open source packages, code snippets, tools, or other third-party software brought into the organization in order to streamline the software development process.

Build Process – the process of compiling, building and/or packaging code, usually via an automated system that also executes tests on built artifacts.

Use/Deployment – the process of working with, testing and running built artifacts in dev, test and production.

Hackers are increasingly targeting software development systems, open source components and software build pipelines in order to gain economies of scale when it comes to cyberattacks. A single compromised software artifact in a popular application can seed hundreds or even thousands of downstream customers with a potential point of attack. It's one of the key reasons that software supply chains are increasingly under assault, with 2023 featuring twice as many attacks as the past 3 years combined, despite the fact that 2019-2022 saw a742% increase in software supply chain attacks, year on year.

As a result, Gartner is predicting that "By 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021[2]". Online marketplace vendor Capterra confirms the trajectory, citing the fact that "three-fifths (61%) of US businesses have been directly impacted by a software supply chain threat" in 2022, and as a result, more than 50% of organizations have lost trust in legacy vendors due to supply chain attacks

[1] Sonatype 9th Annual State of the Software Supply Chain
[2] Gartner report: How Software Engineering Leaders Can Mitigate Software Supply Chain Security Risks

> “74% of IT pros believe technologies like static and dynamic application security testing [SAST & DAST] are important, but feel that those technologies aren’t enough to protect them from supply chain threats”
>
> – ReversingLabs

With IBM reporting that the average cost of a supply chain attack has hit $4.45M, requiring an average of 26 days to identify and contain, it’s no wonder that software supply chain security tools budgets have become an increasingly large part of the budget for security-conscious organizations.

Securing the Weakest Link in the Supply Chain

The import process is often seen as the weakest link in the software supply chain since it requires trusting hundreds or even thousands of open source authors with whom the organization has no relationship. After all, the open nature of open source ecosystems allows anyone to publish anything without a rigorous process to eliminate threats prior to publication.

And because there are no industry-wide standards in place today, each language and repository often requires its own solution. As a result, every time you add a new open source ecosystem to your development stack, you may require a different sets of tools to deal with:

- **Vulnerabilities** - while Software Composition Analysis (SCA) tools have some shortcomings, they remain the best way to decrease Mean Time To Awareness (MTTA) for open source security vulnerabilities.
- **Malware** - while code scanners can have difficulties with binary packages, they provide one of the best ways to identify typical malware threats.

As the depth of the software supply chain increases, new best practices and tools must be implemented to counter emerging threat vectors. For example:

Verifying Provenance - aka, verifying the origin of imported code. Software Attestations can be used here to ensure third-party code has followed best practices when it was developed and built for distribution. TestifySec Witness provides a framework for automating, normalizing and verifying software attestations.

- **Quarantining** - artifact repositories like Sonatype Nexus (and their Lifecycle offering), JFrog Artifactory, or similar solution can be used to quarantine suspicious prebuilt artifacts for investigation.

The rate of change in the software supply chain presents multiple threats, particularly since most organizations are reluctant[3] to update their codebase for a number of reasons, including:

- **Drag on Development** - automated dependency management tools can help you decrease the time and resources required to remediate vulnerabilities, update outdated components, and even automatically build/rebuild dependencies from source code.

Breaking Changes - upgrading even a single dependency can end up breaking the build. Dependency vendoring is one way to avoid dependency conflicts, ensure build consistency and even automate updates/upgrades of your runtime environment.

While numerous open source and commercial solutions exist to help deal with the breadth, depth and rate of change of your software supply chain, keep in mind that:

- Ecosystem-specific tools tend to be more robust, but will significantly increase costs to implement and maintain if you work with more than one open source language.
- Commercial solutions tend to be more comprehensive and expensive, but may be more cost-effective in the long run than trying to integrate and maintain multiple, best-in-class open source solutions.

[3] Veracode's State of Software Security v11: Open Source Edition

SECURING THE BUILD PROCESS

Software build systems take the form of a Continuous Integration / Continuous Delivery (CI/CD) system that can automate the build, test and delivery of software artifacts. While CI/CD has been widely adopted by software vendors, there are a number of security issues the industry continues to wrestle with, including:

Transparency – understanding the original source for all artifacts entering the CI/CD pipeline can improve both security and integrity of built artifacts.

Contamination – longer lived processes and/or containers can become polluted, especially if they're not sealed off from the internet.

Reproducibility – consistent output is key to ensuring artifacts are built securely from the same set of inputs – every time.

Supply chain Levels for Software Artifacts (SLSA) is an emerging software security framework that can help resolve the transparency and contamination issues, and can put you on the road to achieving reproducibility.

SLSA Build Level 1: Provenance – any code, library or open source package imported into the organization must have a type of software attestation known as a "provenance attestation" that shows where the code was sourced from and how the package was built.

A downstream system should be implemented to automatically verify packages were built as expected. The aforementioned TestifySec Witness provides a framework for automating, normalizing and verifying software attestations.

Build Level 2: Build Service + Signing – introduces a build service that includes signing of the provenance attestation. An isolated signing service ensures against bad actors accessing secrets used to sign the provenance, as well as providing downstream systems/users with signed artifacts to indicate they were not tampered with after being built.

A downstream service should be implemented to verify the authenticity of the signature. For example, SigStore offers software signing using Cosign to generate the key pairs needed to sign and verify artifacts, along with a transparent ledger so anyone can find and verify signatures.

Build Level 3: Hardened Builds – specifies a number of controls that harden the organization's build service, including:

- **Build Steps** – each build step should have a single responsibility. When fulfilled, the output should be checked and, if required, passed to the next step.
- **Infrastructure** – all build steps should have dedicated resources that are discarded at step completion, preventing contamination of subsequent steps.
- **Environments** – care must be taken to ensure the runtime environment is minimized, and the container only includes components that are absolutely required.
- **Provenance** – each build step must be dependency complete, and each dependency traceable to the originating source.
- **Service/Network** – run on a segmented network with no internet or manual access in order to limit local exploits and remote tampering/intrusion. This means employing:
  - **Pre-Scripted Parameterless Builds** – build scripts cannot be accessed and modified within the build service, preventing exploits.
  - **Ephemeral, Isolated Build Steps** – every step in a build process must execute in its own container, which is discarded at the completion of each step.
  - **Hermetically Sealed Environments** – containers have no internet access, preventing (for example) dynamic packages from including remote resources.

Bonus:

- **Reproducibility** – if the same "bits" input don't always result in the same "bits" output, there's no guarantee the artifacts you're working with haven't changed from build to build.

In general, CI/CD systems support either a declarative programming model (supports SLSA Build Level 3), or an imperative programming model based on scripts (supports SLSA Build Level 2). Look for an open source or commercial CI/CD solution that allows you to implement a declarative pipeline that breaks down each stage of the pipeline into multiple discrete steps.

SECURING THE DEPLOYMENT PROCESS

Providing a centralized set of dependencies approved for use ensures:

- Developers always know which dependencies/ versions to use.
- Compliance, security and development managers can vet dependencies for licensing, vulnerabilities and usability/ maintainability.

Most importantly, it institutes a process that discourages developers from downloading unapproved dependencies directly. Artifact repositories provide the best way to manage binary artifacts used in and generated by the software development process.

Conclusions

It seems like every vendor is now a software supply chain security vendor, causing confusion in the marketplace. But avoiding this market confusion is not an option since it only makes businesses increasingly susceptible to the threat of ransomware, malware, and other security risks. Organizations need tangible solutions that can help them address supply chain security threats during their code import, build and deployment processes.

Following Biden's Executive Order[4] on improving cybersecurity, many are still trying to figure out what that looks like for their own organization. As software supply chain attacks hit unprecedented levels, it may be time to start rethinking whether downloading prebuilt open source software makes sense anymore.

Organizations that don't vendor their dependencies and build them from source code can only play catch up with supply chain security via traditional AppSec tools – tools that are often working against an incomplete set of dependencies since their dependency graph wasn't generated at build time.

But not only do you need to vendor all your dependencies in order to build ***everything*** from source code, you also need to set up and maintain a secure, hardened build system. The time and resources required for such a task are beyond the means of all but the largest of software vendors.

This is why we built the ActiveState Platform: to vendor your project's dependencies on your behalf, and automatically build them securely from source code for you.

The ActiveState Platform is a SLSA Build Level 3-compliant, hardened build service that builds all of your open source dependencies (including linked C and Fortran libraries) from source code, and then packages them into a secure runtime environment.

ActiveState goes a step further by ensuring all builds are reproducible, as well as generating an SBOM and signed software attestations (both Provenance and Verification Summary Attestations).

By integrating your existing software development process with the ActiveState Platform you can gain SLSA compliance in a matter of days (not months), freeing up your developers to work on what matters most: creating new features and functionality.

[4] Executive Order on Improving the Nation's Cybersecurity, MAY 12, 2021

activestate

About ActiveState

ActiveState enables DevOps, InfoSec, and Development teams to improve their security posture while simultaneously increasing productivity and innovation to deliver secure applications faster.

With a single platform that tames open source complexity, teams get a continuously secure software supply chain, unparalleled observability, robust vulnerability management, continuous upgrades, and governance support that enhance collaboration across the organization.

All from the trusted partner that pioneered and continues to lead enterprise adoption and use of open source software.

Start An Enterprise Trial

www.activestate.com • Toll-free in NA: 1-866.631.4581 • solutions@activestate.com