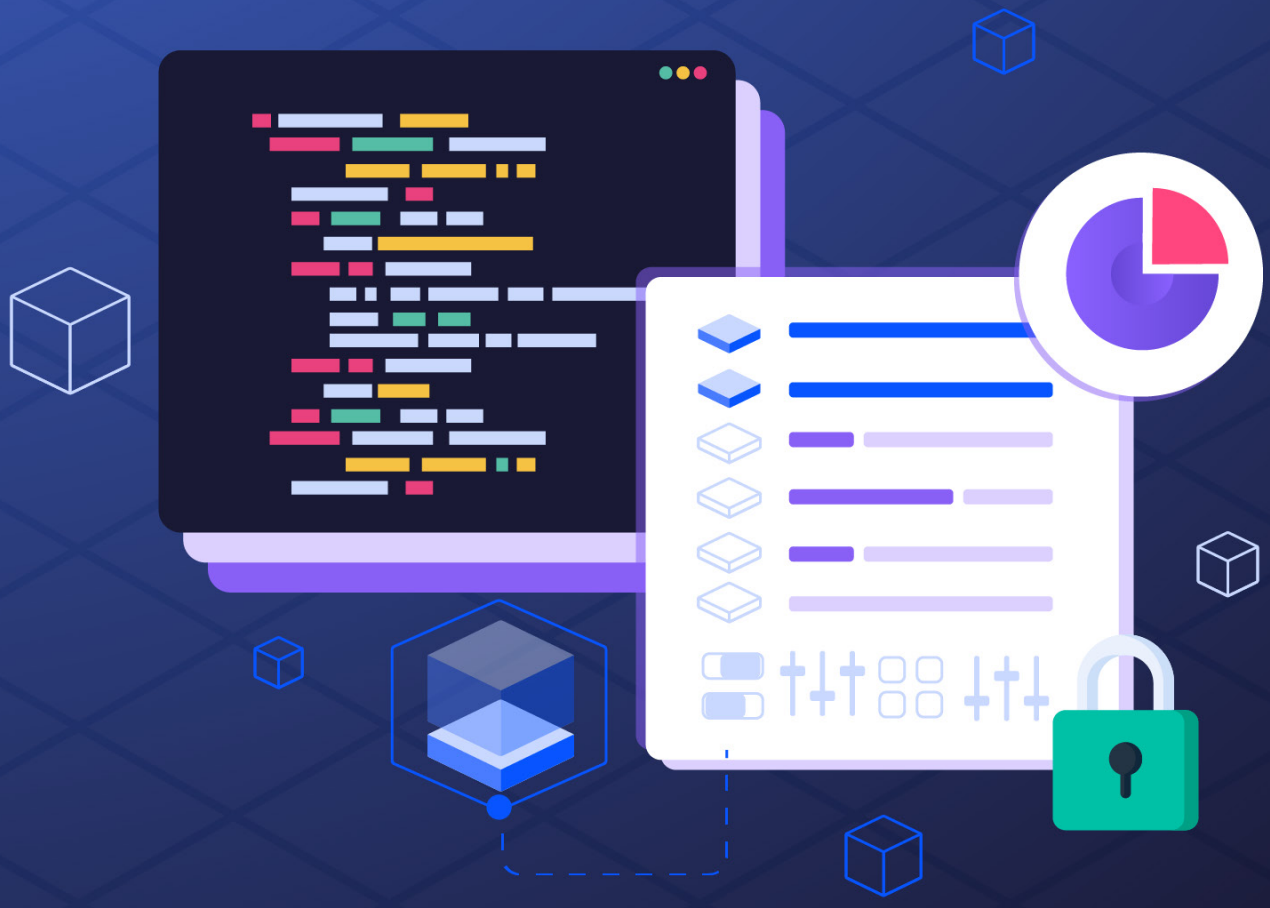
ActiveState

SOFTWARE SUPPLY CHAIN SECURITY BUYERS GUIDE **FOR BUILD TOOLS**



Executive Summary

The software supply chain has been increasingly under attack since the Solarwinds hack of December 2020, which proved the threat to be very real for everyone from the US government to the Fortune 500, and all the SMBs in between. Given the size of the opportunity, it's no wonder that the market has seen an explosion in vendors claiming to offer supply chain security solutions, including those that encompass the software build process.

Security-conscious organizations know that the only way to ensure the security and integrity of software dependencies is to build them from scratch by:

- Importing all open source dependency source code into their code repository
- Creating a declarative CI/CD pipeline consisting of isolated, ephemeral environments
- Generating an SBOM and attestation to prove the security & integrity for downstream consumers

However, most of the software industry builds very few third-party dependencies from source code, preferring to import prebuilt software due to the cost and complexity of vendoring dependencies in any organization with multiple development teams and diverse technology stacks.

A number of traditional and emerging tools discussed in this guide can help reduce the cost and complexity of vendoring your dependencies.

Introduction

Organizations are increasingly concerned with the security of their software supply chain, but often have trouble navigating the ever-expanding labyrinth of open source and proprietary software solutions that claim to help. While solutions exist across the entire software supply chain, this section of ActiveState's Buyers Guide to Securing the Software Supply Chain focuses specifically on the tools used in the software build process.

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

Solarwinds has become the poster child for supply chain security within the build process, having delivered signed software to thousands of customers who assumed it was secure despite the fact that their Orion product had been compromised after the signing process within their CI/CD pipeline.

Code signing has been used for decades to ensure that software has not been altered or corrupted between the time it was built and the time it gets deployed. But the real value of signed code for most customers is the establishment of trust, which is why the SolarWinds hack was so pernicious: it effectively undermined trust in signed software.

Despite this fact, very few enterprises are taking proactive efforts to identify, assess and mitigate software supply chain risks. In fact, according to Sonatype[1], only 7% of surveyed organizations are prioritizing supply chain security.

This result is echoed by results from the "Application Security Posture Management (ASPM) 2024" report that surveyed 500 US CISOs:

> "77% of CISOs perceive software supply chain security as a substantial blind spot for AppSec"
>
> ASPM 2024 Report

1. Sonatype 9th Annual State of the Software Supply Chain

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. CI/CD is an agile software development best practice designed to enable more frequent and reliable code updates.

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 security framework that can help ensure that all the code you import into your organization and/or build within your organization is done in a secure manner. As such, SLSA is key to helping resolve the transparency and contamination issues, and can put you on the road to achieving reproducibility.

SLSA Secures the Build Process

SLSA defines a number of "Build Levels" that can provide you with key stages on your road to securing your build service:

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:

- Who built the package (person or system)
- What process/command was used
- What the input artifacts (e.g., dependencies) were

A downstream system should be implemented to automatically verify packages were built as expected. For example, 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.

Declarative Pipelines

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). Hardening your build service will require a CI/CD solution that allows you to implement a declarative pipeline that breaks down each stage of the pipeline into multiple discrete steps.

Open Source Declarative Pipeline Tools

The following open source CI/CD solutions can support declarative pipelines:

Jenkins is perhaps the world's most popular automation server, and offers hundreds of plugins (including a declarative pipeline plugin) to support building, deploying and automating any project on Windows, Linux, and macOS.

Key Capabilities:

- Highly extensible with a large plugin ecosystem (1500+ plugins)
- Integrates with most popular cloud platforms
- Supports performing work in parallel

Advantage: Best-in-class community support.

Spinnaker (originally from Netflix) is a multi-cloud continuous delivery platform that provides a flexible pipeline management system with integrations to most major cloud providers.

Key Capabilities:

- Create pipelines that launch and stop server groups, system tests and track rollouts
- Create immutable images to accelerate rollouts, simplify rollbacks and eliminate configuration drift problems
- Integrate with monitoring services like Datadog, Prometheus, Stackdriver, or SignalFx for canary analysis

Advantage: Offers a CLI administration tool (Halyard) you can use to install, configure, and upgrade your instances.

Drone offers an Apache 2.0 licensed open source CI platform in which each pipeline step is executed inside an isolated Docker container to ensure security.

Key Capabilities:

- Integrate with most popular code repos like GitHub, Bitbucket and GitLab.
- Supports parallel builds and tests
- Integrates with LambdaTest for cross-browser testing
- Offers plugins for cloud integration, reporting, testing, notifications, etc

Advantage: By default, isolated Docker containers ensure build steps do not conflict.

Commercial Declarative Pipeline Tools

In addition to the popular open source tools listed above, there are a number of commercially available CI/CD solutions that also support declarative pipelines, including:

TeamCity – created by JetBrains, TeamCity can be installed on Windows and Linux servers. It provides integration with Docker, Visual Studio Team Services, Maven, NuGet, Azure DevOps, Jira Software Cloud, etc, and also supports launching build agents in Kubernetes clusters.

Key Capabilities:

- Highly extensible/ customizable
- Runs parallel builds
- Pipelines are defined using Kotlin-based Domain Specific Language (DSL)

Advantage: Provides for viewing test progress (and history) reports on-the-fly.

CircleCI is available as both a cloud-based and on-premise (self-hosted) solution that supports Windows, Linux, and macOS. It uses a proprietary YAML syntax for its pipelines. Its cloud native CI/CD pipelines are easy to set up as workflows that can be integrated with code repos like GitHub, Bitbucket, etc.

Key Capabilities:

- Builds can be split and balanced across multiple containers
- Supports parallel testing

Advantage: CircleCI Orbs are reusable snippets of code that help automate repetitive tasks and streamline integration with third-party tools.

Travis CI is an early entrant in the CI/CD Security market that initially supported only open-source projects, but subsequently added support for closed-source projects, as well. Travis provides both a cloud-hosted (SaaS) and self-hosted version of their offering that supports more than 30 programming languages on all major OSs, and integrates with common code repos like GitHub and Bitbucket.

Key Capabilities:

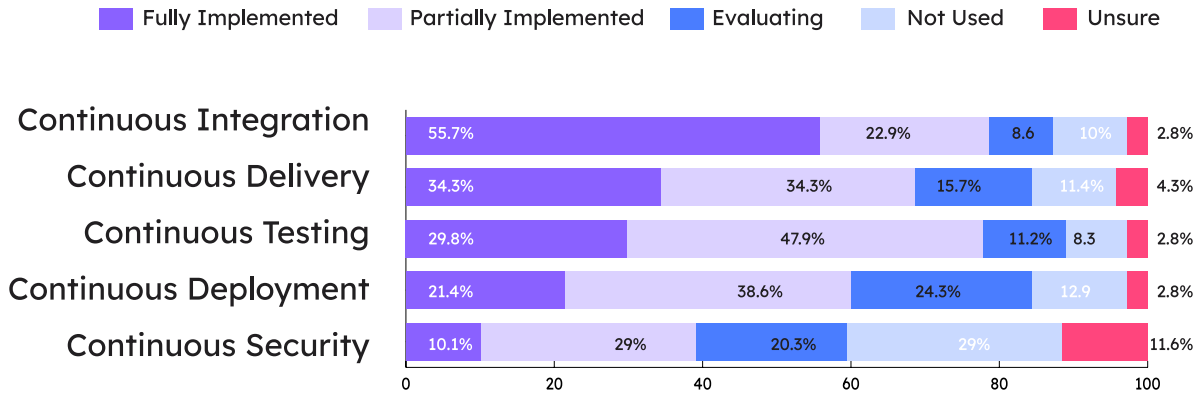
- Build matrix provides support for parallel builds
- Offers integration with cloud based testing platform LambdaTest for testing across different browsers, platforms, and devices (emulators)

Advantage: Uses a proprietary YAML syntax that integrates with GitHub Enterprise tools.

There are literally dozens of players in the CI/CD Security market, which is far too many to list here. For an overview of how many of the most popular CI/CD tools compare, CloudZero offers a comprehensive overview.

Conclusions

ActiveState's State of CI/CD Survey found that while most software vendors are at least in the process of implementing a CI/CD system, very few would consider themselves experts.



Tellingly, the practice of Continuous Security is the least mature. One of the key reasons for this is due to the fact that the only way to ensure the security and integrity of any built software requires that all its components be built from source code. When you consider that >80% of all components in any modern software application are open source libraries, building them all from source (along with their complete set of dependencies and any linked C libraries) becomes an extremely complex task. Especially if your organization has multiple development teams and a diverse technology stack.

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 features and functionality.**

ActiveState

Secure open source integration

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