ActiveState

State of Software Supply Chain Security 2021

Lessons for 2022

Executive Summary

Software supply chain security encompasses two main disciplines:

One - Vulnerability Remediation
Two - Security & Integrity of Software Development Processes

Vulnerability Remediation is a well-known process with its own best practices that are well documented elsewhere. **This survey focuses on the security and integrity of key software development processes, including:**

Import | Best practices for securely importing third-party code, such as open source software.

Build | Best practices for securely building software components from source code.

Run | Best practices for securely working with and running software.

The survey garnered more than 1500 responses from coders, security experts and open source advocates worldwide at organizations ranging from small businesses to large enterprises. **The results point to a number of worrisome trends, including:**

- Software supply chain security is currently an immature discipline.
- Build reproducibility is by far the best practice with the least adoption (without reproducibility, software built from source code cannot be deemed secure).
- Implicitly trusting insecure open source repositories is a growing risk. More discipline is required to secure imported code.

Implementing an end-to-end secure software supply chain is a non-trivial undertaking. Integrating multiple point solutions and custom code can be both costly and time consuming. Organizations should look for turnkey solutions that can help bridge the gap quickly to avoid becoming compromised by bad actors.

Table of Contents

ActiveState Survey: Software Supply Chain Security 04
ActiveState and the Software Supply Chain – What's the Connection? 05

Part 1 - Demographics **06**
Q1 - What is the Size of Your Organization? 07
Supply Chain Security Rating by Organization Size 08
Supply Chain Security Rating by Geographic Region 09
Q2 - What Best Describes Your Role/responsibilities? 10
Roles by Organization Size 11

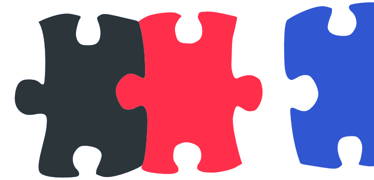
Part 2 - Import Cntrols **12**
Q3 - How Do You Verify Imported Open Source Code? Check All That Apply. 14
Trust by Organization Size 16
Import Controls by Organization Size 17

Part 3 - Build Controls **18**
Q4 - Do You Build the Open Source Packages You Use From Source Code? 20
Building From Source Code By Organization Size 21
Q5 - How Do You Ensure Open Source Builds Are Secure? Check All That Apply. 22
Build Controls By Organization Size 24

Part 4 - Run Controls **25**
Q6 - Do You Work With Signed Packages? 27
Q7 - Does the Signature Include the Following Information Via a Cryptographic Hash? Check All That Apply. 28

Key Takeaways **30**
About ActiveState 32

ActiveState Survey: Software Supply Chain Security



The open source software supply chain has always been susceptible to cyberattack, not least because it's composed of public repositories that feature unsigned software uploaded by anyone that cares to contribute to the ecosystem.

With hundreds of thousands of developers submitting millions of software assets to dozens of repositories that provide little to no guarantee of the security or integrity of those software assets, the message is clear: user beware.

This survey was undertaken to help understand what wary organizations are doing to limit their exposure to the potentially malicious and/or compromised software they import and use within their software development processes. Specifically, the survey examines the implementation of best practices during the import, build and run stages.

☑ **Import** The process of importing third-party tools, libraries, code snippets, packages and other software resources in order to streamline development efforts.

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

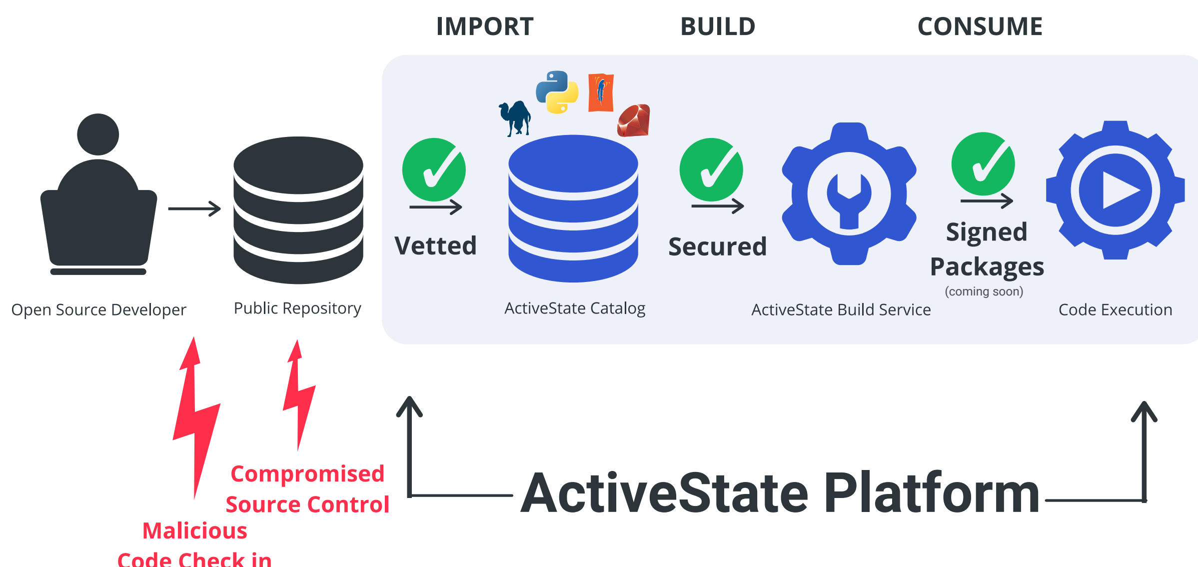
☑ **Run** The process of working with, testing and running built artifacts in development, test and production environments.

By reading these survey results, organizations can get an understanding of what works, what doesn't, and how they can improve their best practices so as to increase the security and integrity of their software supply chain.

The survey results are also instructive in helping ActiveState identify gaps in the software supply chain that our universal package management platform - the ActiveState Platform - can fill.

ActiveState and the Software Supply Chain - What's the Connection?

With a 20+ year history of creating open source language distributions used by organizations both large and small, we've experienced first hand the kinds of supply chain risks enterprises need to wrestle with when importing, building and working with open source components. It's one of the reasons we built the ActiveState Platform, which can help organizations secure their Python, Perl, Tcl and Ruby supply chains by providing a turnkey service that's quick to set up, easy to use and highly automated.



The ActiveState Platform provides:

Secure Import Process
Source code is imported, vetted and flagged for maintainability, security and commercial use.

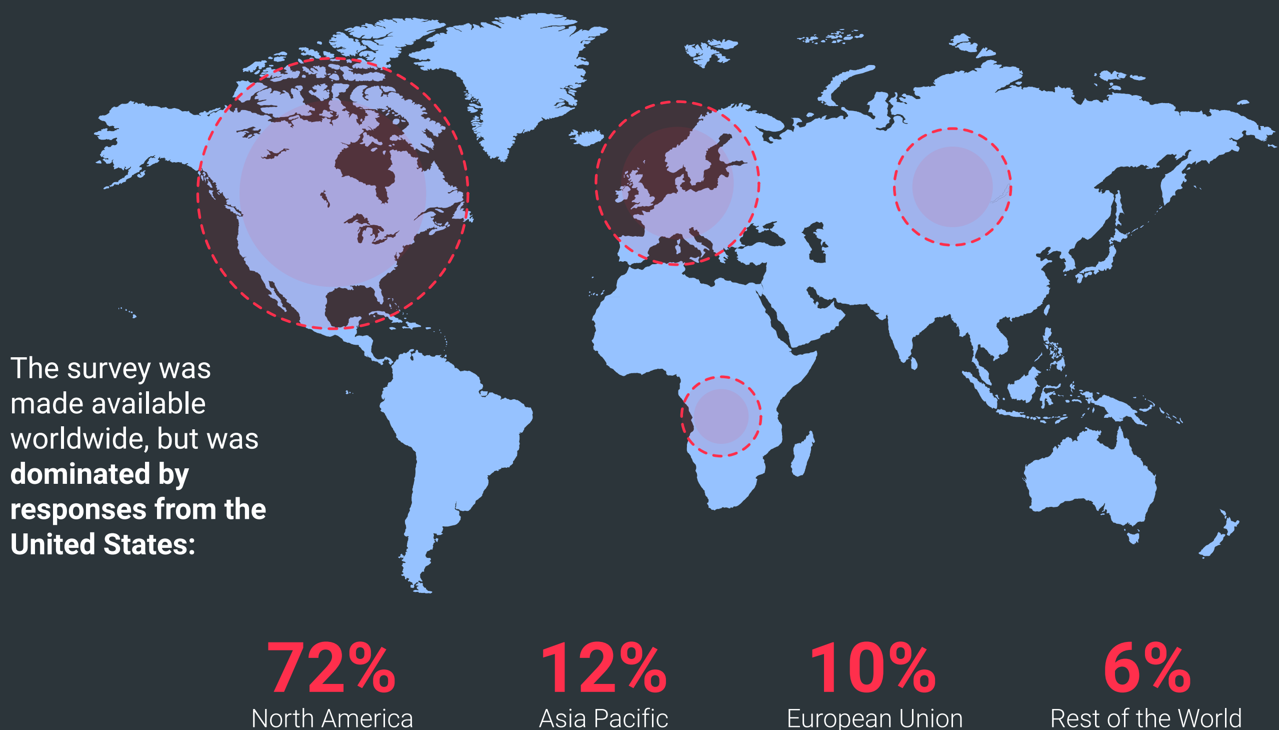
Secure Build Service
Automated, scripted builds that employ ephemeral, isolated and hermetically sealed (i.e., no public network access) environments for each build step.

Securely Built Artifacts
Developers and DevOps gain verifiably reproducible builds that contain artifacts featuring non-falsifiable provenance (ie., each artifact can be traced to its original source).

PART 1

Demographics

The ActiveState Supply Chain Security Survey was taken by more than 1500 respondents who work at organizations of all sizes, but tend to occupy one of three broad roles associated either with working with code, securing code, or providing open source governance.



The survey was made available worldwide, but was **dominated by responses from the United States:**

72% North America

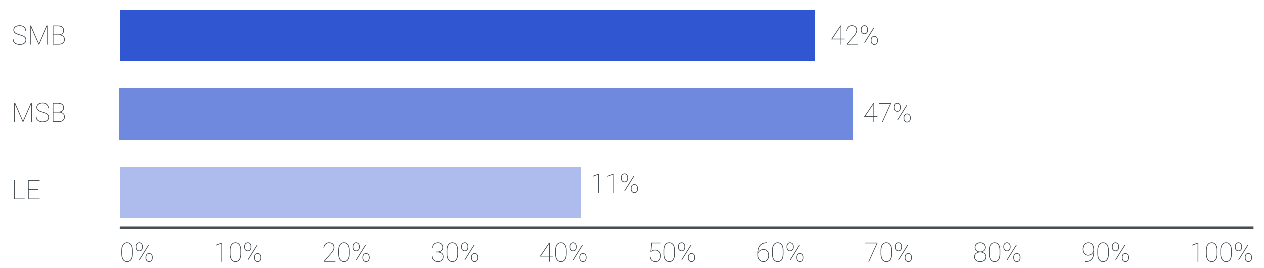
12% Asia Pacific

10% European Union

6% Rest of the World

The fact that the survey was dominated by respondents from the US is likely indicative of an increased regional awareness given the number of local, high profile supply chain attacks like SolarWinds, which prompted US President Biden to issue an Executive Order deploring the current state of supply chain security.

Q1 - What is the size of your organization?



42%
SMB

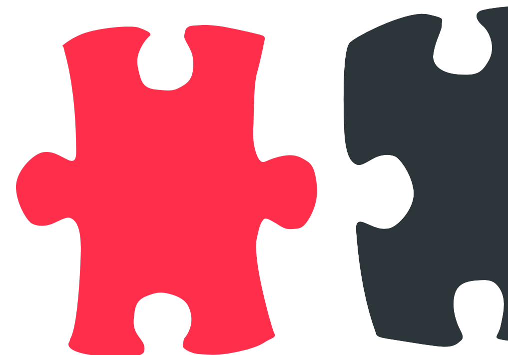
42% of respondents work at Small & Medium Sized Businesses, defined as organizations with less than 200 employees.

47%
MSB

47% of respondents work at Mid-Sized Businesses, defined as organizations with 200 to 2,000 employees.

11%
LE

11% of respondents work at Large Enterprises, defined as organizations with more than 2,000 employees.



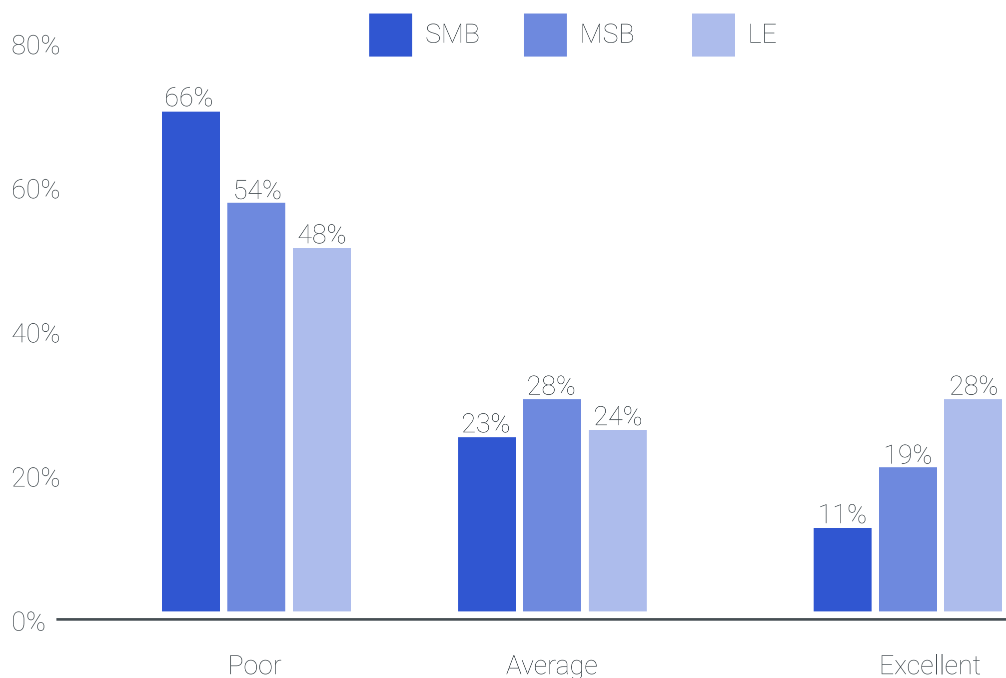
Software Supply Chain Security Rating by Organization Size

The survey provided an overall supply chain security rating for all participants, based on their responses. All questions were weighted evenly since the security of a software supply chain is only as strong as the weakest link.

Poor Assigned to respondents that have implemented a minimum of import, build and run controls.

Average Assigned to respondents that have implemented many of the import, build and run best practice controls.

Excellent Assigned to respondents that have implemented a majority of the import, build and run best practice controls.

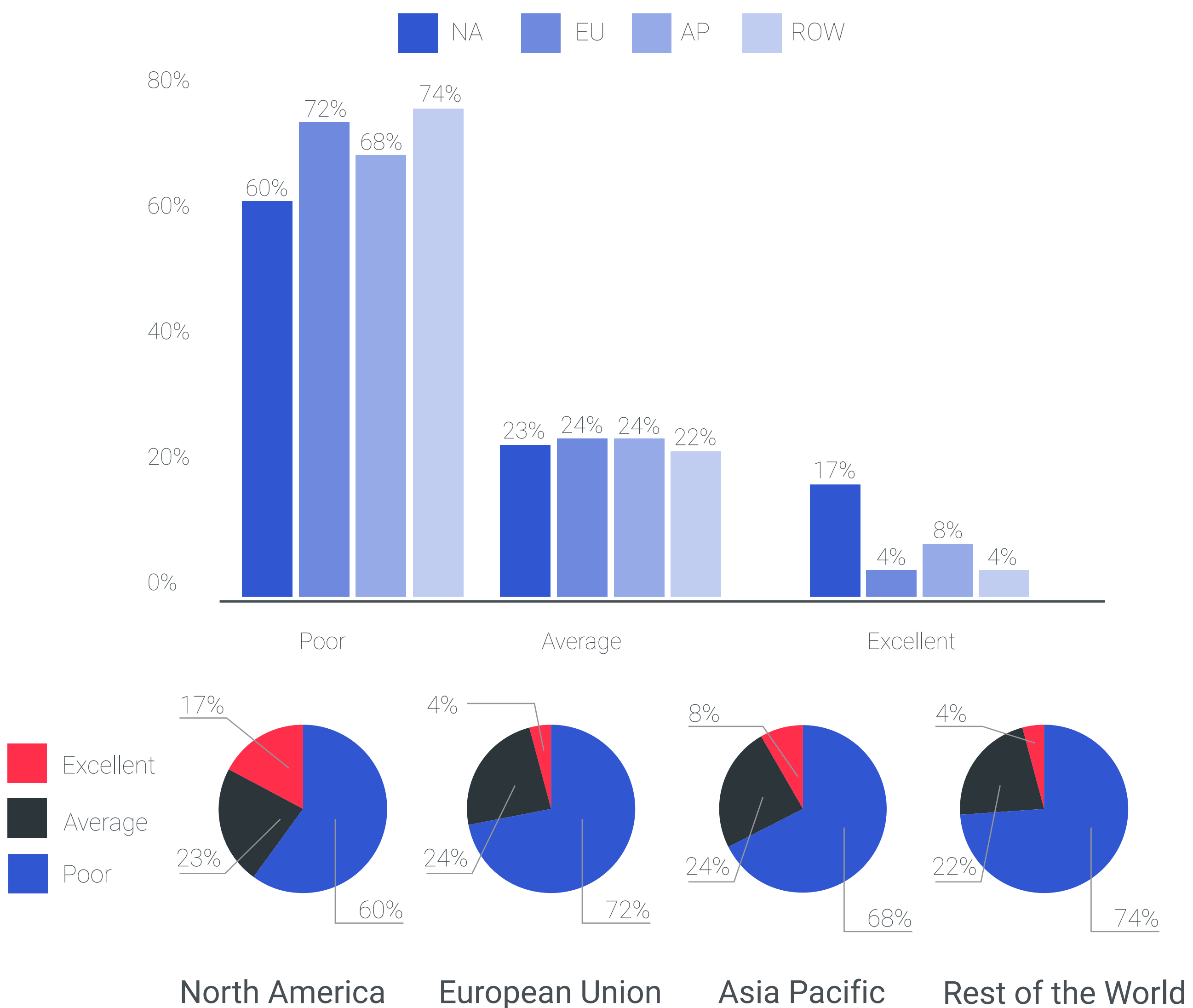


In general, the survey results indicate that supply chain security is not a mature discipline, no matter the size of the company. However, smaller organizations tended to be rated poorer, while larger organizations dominated the “Excellent” rating.

There’s nothing surprising about these results, which are really just an indication of the fact that implementing security and integrity controls across the entire software supply chain is an expensive and resource-intensive undertaking better suited to larger organizations.

Software Supply Chain Security Rating by Geographic Region

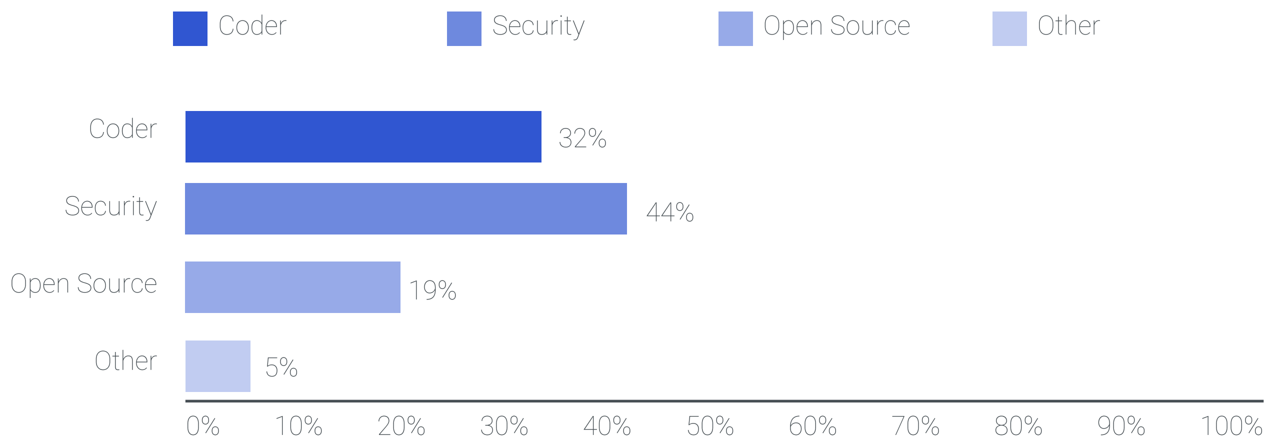
In general, the strength of an organization's supply chain security is not geography dependent. When it comes to implementing a secure supply chain, no region of the world is any further ahead than another.



North America European Union Asia Pacific Rest of the World

The one outlier is the fact that at least twice as many North American companies have already achieved an "Excellent" supply chain security rating compared to other geographies. However, more than 80% of North American organizations still lack a truly robust solution.

Q2 - What best describes your role/responsibilities?



32% **Coder**

32% of respondents indicated that they were coders, which includes titles such as Developer, Engineer, Programmer, IT, QA, DevOps, Ops, etc.

44% **Security**

44% of respondents indicated that they were security personnel, which includes titles like CISO, InfoSec, Cybersecurity and Security Analyst / Engineer / Architect / Consultant, etc.

19% **Open Source Advocate**

19% of respondents indicated that they were open source advocates, which includes titles like Open Source Officer, Program Office, Advocate, Strategist, etc.

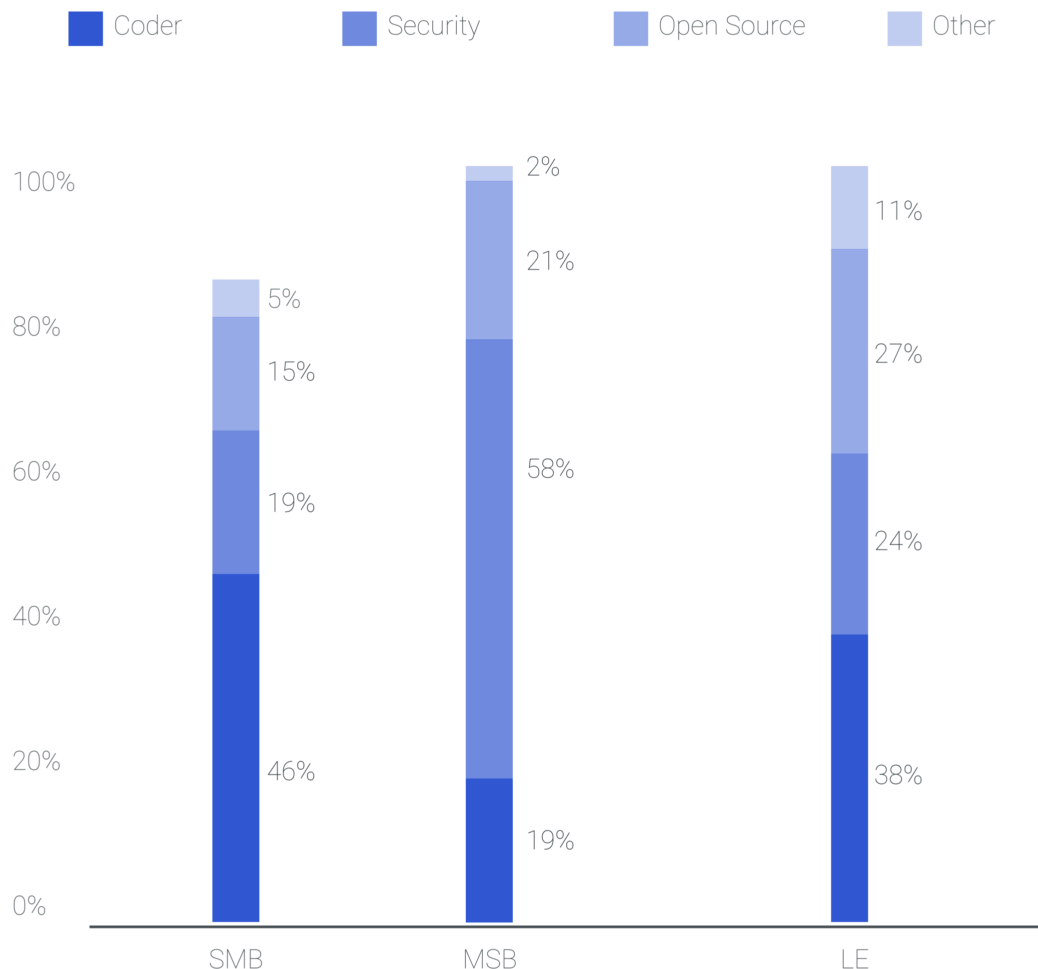
5% **Other**

5% of respondents indicated that they held a different role, including Product Manager, R&D/Engineering Manager, Support Manager, Project Manager, SysAdmin, Technician, CEO/CIO/CTO and Student.

It seems appropriate that the largest block of respondents were security personnel who are primarily concerned with the integrity and security of the software their organizations create and use. In ActiveState's experience, coders generally view security as secondary to their primary role of completing their coding deliverables. However, their input here is key since they are also the ones responsible for resolving issues found by their security teams.

In contrast, open source advocates are generally responsible for establishing policies and governance around the use of open source in their organization, and would likely have input on security measures, as well.

Roles by Organization Size



Given that in most organizations (regardless of size), there tend to be more coders than security personnel, we expected to see more responses from those responsible for writing code than evaluating the security of that code. While this held true for SMBs and LEs, more than half of MSB respondents characterized their role as "Security." This may be an anomaly, or it may be an indication that, for MSBs at least, security is becoming everyone's responsibility. More followup is required.

Open source advocates, by comparison, participated at a rate that reflected the size of their company: the larger the organization, the greater the proportion of open source advocates that participated in the survey. This is likely a reflection of the fact that larger companies tend to put more of an emphasis on hiring for open source roles than smaller ones.

PART 2

Import Controls

Respondents were asked what kinds of controls they have in place to ensure against importing compromised software. For instance, importing open source components from public repositories poses a number of risks.

Typosquatting

Also known as brandjacking or cybersquatting, this is the practice of attackers submitting a compromised package to an open source repository that is named similar to a popular, existing package.

Author Impersonation

While many public repositories have implemented 2-factor authentication to help mitigate author account hijacking, packages with no reviewers, or with fewer than two reviewers should be treated as suspect. As should packages that have new authors all of a sudden.

Dependency Confusion

Dependency Confusion can occur when a build system mistakenly pulls in a similarly named dependency from a public repository rather than your private repository.

Public repositories are just that: public, which means anyone can upload whatever code they want. While most public repositories have implemented 2-factor authentication to limit author impersonation, they have yet to verify and sign the code they offer. Thus, no guarantees are offered as to whether prebuilt packages are malware-free.

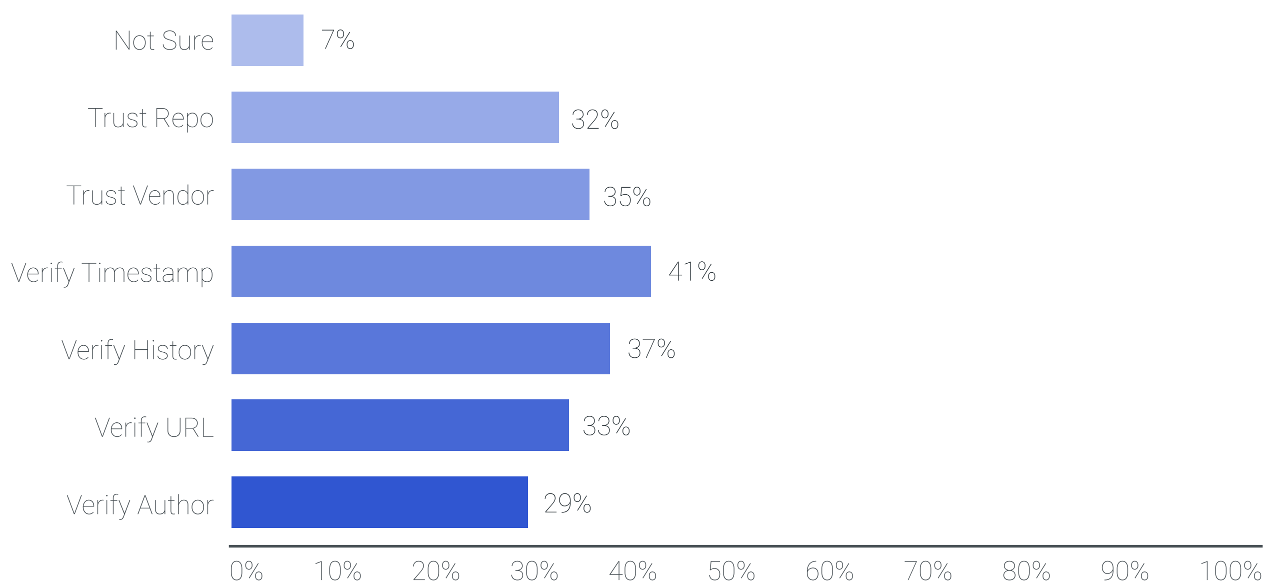
To limit the risk of using public repositories, security-conscious organizations typically implement a number of controls that might include verifying the author, maintainers, and reviewers, as well as checking timestamps, and possibly even implementing a quarantine zone for code that fails to pass.

The ActiveState Platform imports only source code from public repositories like Python Package Index (PyPI), CPAN, RubyGems, GitHub, among others. The code is vetted for maintainability, security, and commercial use, and then loaded into the ActiveState Platform catalog, ready to be automatically built on demand. Developers can therefore be assured that the code they use from the ActiveState Platform is far more secure than working with prebuilt packages from public repositories.

Read more about the ActiveState Platform's import controls >

Q3 - How do you verify imported open source code? Check all that apply.

Participants were asked about the import process they use to bring code from external sources into their organization.



Of note is the fact that almost one-third of respondents continue to implicitly trust public repositories, despite the growing number of supply chain attacks targeting them. Surprisingly, almost as many respondents trust public repositories as trust their vendor, despite the disparity in the security of the components they offer. However, the survey did not distinguish between those that import prebuilt components from public repositories versus those that import only source code. The risk posed by importing source code is less compared to the risk of importing prebuilt packages and/or precompiled binaries, which can obfuscate malicious code.

32%
Trust Repo

32% of respondents indicated they implicitly trust public repositories such as npm, PyPI, GitHub, etc.

Implicitly trusting public repositories is risky, given that they provide no guarantees as to the security and integrity of the components they offer.

35%
Trust Vendor

35% of respondents indicated they implicitly trust their vendor's ecosystem, such as Redhat, Anaconda, etc

Trusting a vendor is far less risky than trusting a public repository, but security-conscious organizations should still be prepared to trust but verify.

41%
Verify Timestamp

41% of respondents indicated they verify the timestamp of the code commit.

Verifying the timestamp is a simple way to check for anomalous uploads.

37%
Verify History

37% of respondents indicated that they verify the change history of the code they import.

Checking the commit history of a component raises awareness around change descriptions, contents of the change, and/or parent revisions.

33%
Verify URL

33% of respondents indicated that they verify the URL/immutable reference to the original source of the code they import.

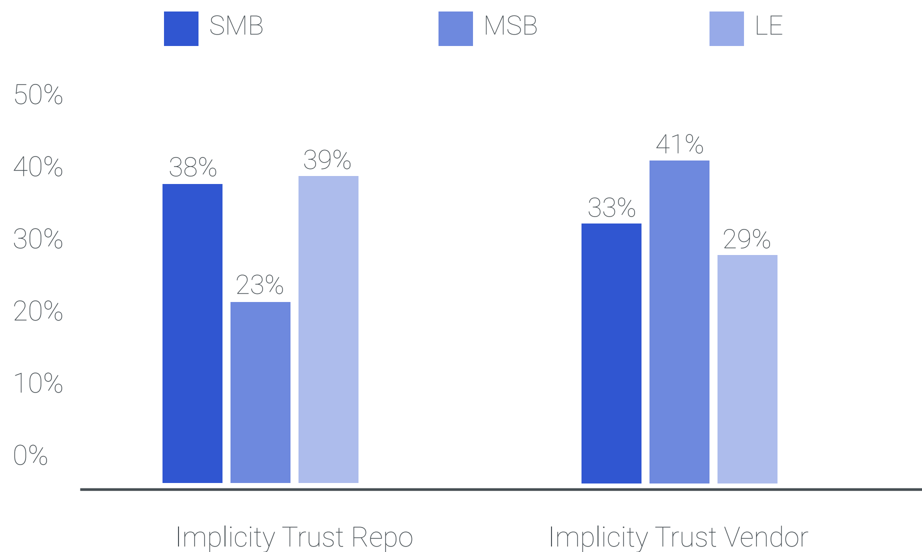
Recording the URL allows organizations to trace built artifacts back to their original source. It can also help identify whether you have inadvertently imported a compromised component subsequently discovered by the community.

29%
Verify Autor

29% of respondents indicated they verify the identity of uploaders and reviewers.

You should always check whether the code has been vetted by at least two reviewers, and that the uploader and reviewer are two different trusted persons.

Trust by Organization Size

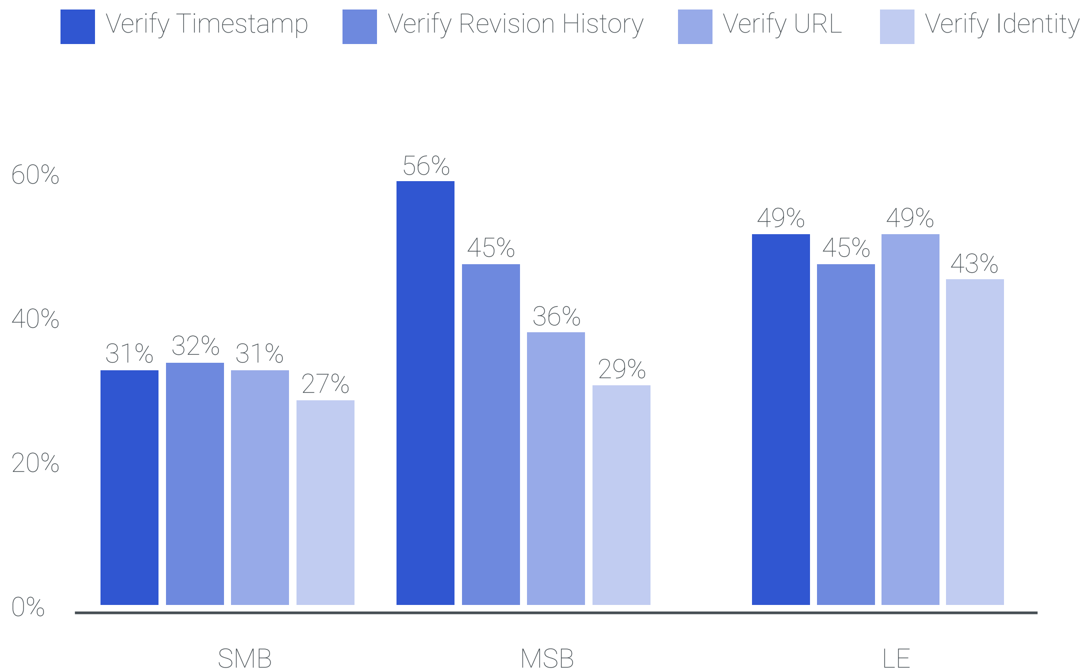


Here, it's interesting to note that while SMBs and LEs tend to make more use of public repositories, MSBs are at least 30% more likely than SMBs or LEs to rely on code from their vendor's ecosystem, such as ActiveState's Perl or Anaconda's Python, for example.

This result seems to correlate strongly with the fact that the majority of MSB respondents are security professionals, who are more likely to put their trust in a proven vendor over a public repository.

The fact the LEs are more likely to put their trust in public repositories than vendors is indicative of the fact that they are the most likely group to import source code and build it themselves, as we'll see in the next section on Build Controls.

Import Controls by Organization Size



Interestingly, both SMBs and LEs were fairly consistent in implementing the entire range of code import best practices, whereas MSBs placed far more emphasis on verifying timestamps than any other control.

In fact, a simple timestamp check was almost twice as likely to be implemented than the more complex verification of uploader/reviewer identities.

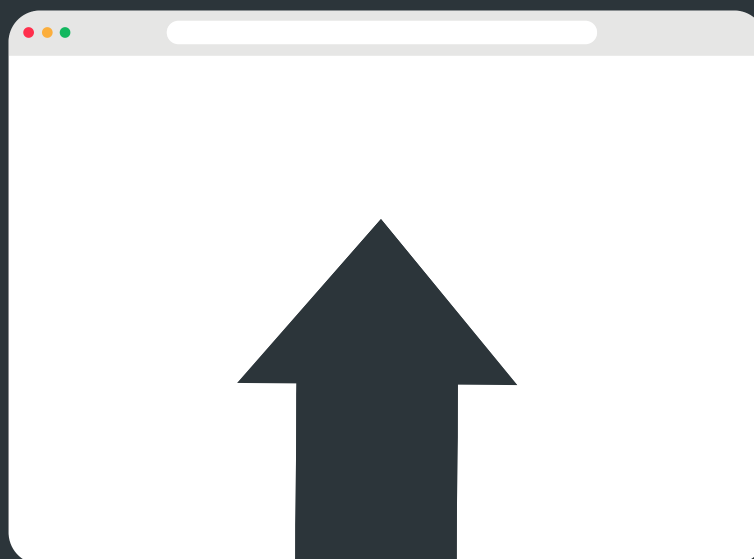
PART 3

Build Controls

Respondents were asked what kinds of controls they have in place to ensure the build process for their software is secure. Supply chain attacks are on the rise, and the build process is a key target. For example, infamous hacks like SolarWinds and Codecov were attacks against their build environments.

Without a secure build service, organizations can be exposed to a number of vectors of attack, including:

- Malicious install scripts that pull in packages you don't expect.
- Unconstrained packages that do more than you expect.
- Dynamic packages that include remote resources.



To counter these kinds of attacks, organizations should implement a number of best practices:

Secure Build Service
A dedicated service that runs on a minimal set of predefined, locked down resources rather than a developer's desktop or other arbitrary system that can offer a larger attack surface to hackers.

Scripted Builds
Predefined build scripts that cannot be accessed and modified within the build service, preventing exploits.

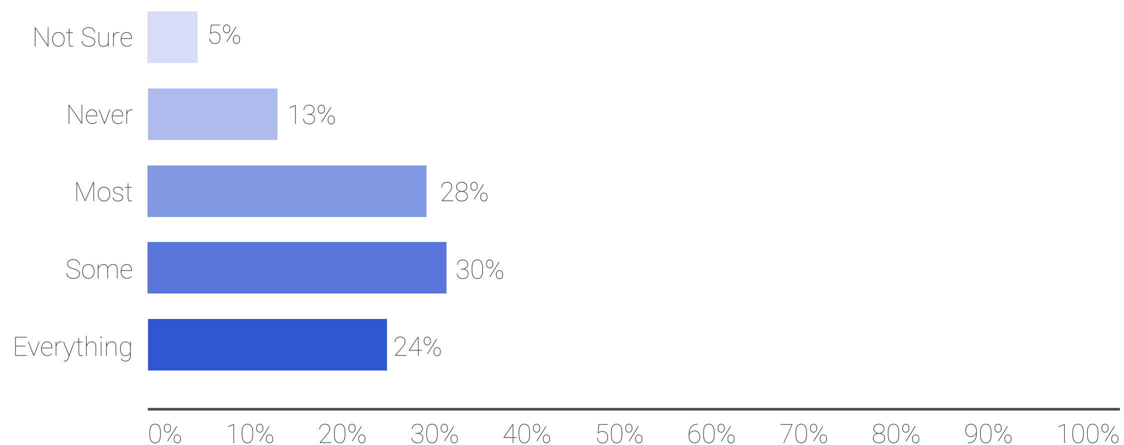
Ephemeral, Isolated Build Steps
Every step in a build process should execute in it's own container/VM, which is discarded at the completion of each step. In other words, containers/VMs are purpose-built to perform a single function, reducing the potential for compromise.

Hermetic Environments
Containers/VMs have no internet access, preventing (for example) dynamic packages from including remote resources.

The ActiveState Platform's secure build service implements all of the above best practices to ensure the integrity of artifacts built on demand from source code haven't been compromised. The output is a verifiably reproducible build, where not only do the same inputs produce the same outputs every time, but whose provenance can also be verified by tracing each component back to its original source.

Read how the ActiveState Platform can secure your builds >

Q4 - Do you build the open source packages you use from source code?



5%
Not sure

5% of respondents were unsure whether they build open source packages from source code.

13%
Never

13% of respondents indicated that they only work with prebuilt binaries, obtained from either public repositories or else vendors.

While working with prebuilt components is faster and easier than building from source, it can potentially expose organizations to undue risk.

28%
Most

28% of respondents indicated that they build most of the open source packages they use from source code.

30%
Some

30% of respondents indicated that they build at least some packages from source code.

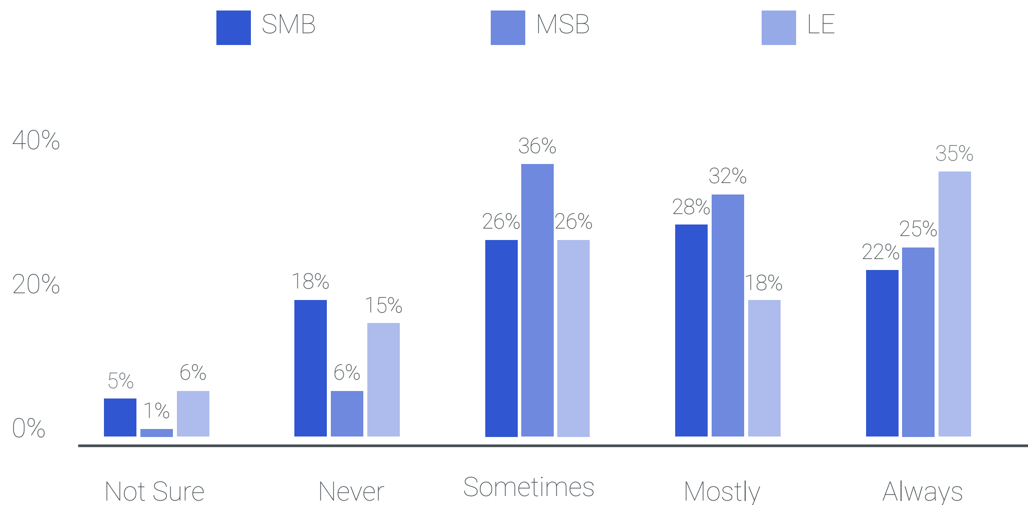
In ActiveState's experience, when organizations report building "some components from source" they are generally referring to OpenSSL, which frequently requires patching/updating to ensure the network communications of the software you build remain secure.

24%
Everything

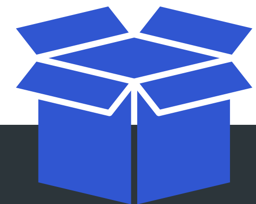
24% of respondents indicated that they build everything from source code.

The best practice of building all software components used in the development process from source code is not as widespread as it should be. As supply chain attacks increase, whether you build some or most components from source code, you are still introducing undue risk into your organization by working with prebuilt components.

Building From Source Code By Organization Size

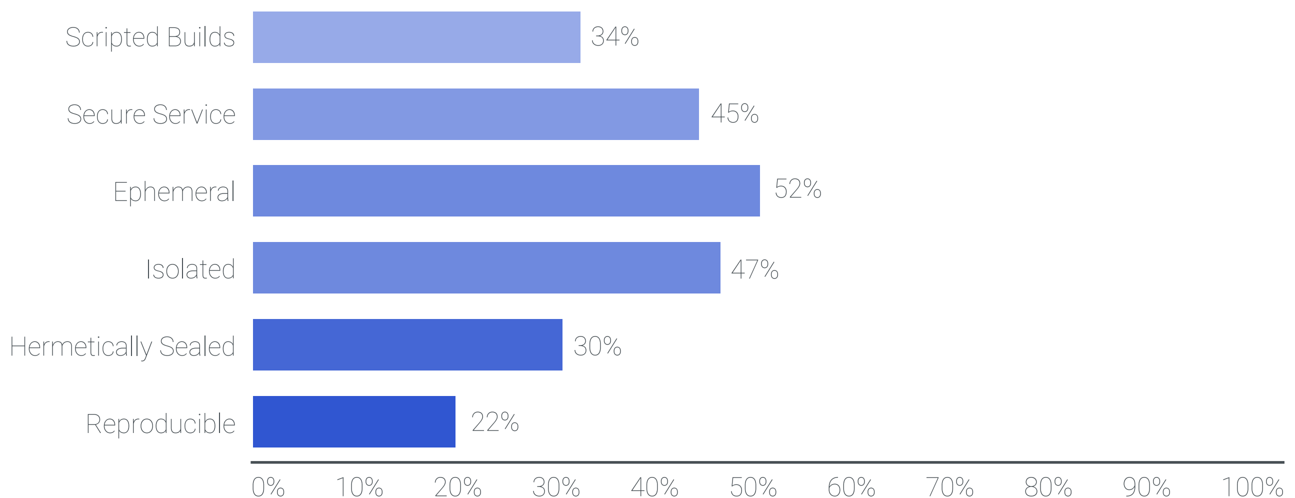


Unsurprisingly, SMBs are the least likely to spend their limited resources to build the open source packages they use from source code, while LEs are most likely to always build everything from source.



The general trend is in the right direction:
80% or more of respondents, no matter the size of their organization, build at least some components from source.

Q5 - How do you ensure open source builds are secure? Check all that apply.



34%
Scripted Builds

34% of respondents indicated that their builds are defined by a build script (i.e., builds do not require manual inputs).

The less need for manual input, the fewer points of potential compromise.

45%
Secure Service

45% of respondents indicated that their builds are run using a dedicated build service that is not on a developer's workstation.

Build services executed on a locked down system that runs only those services required to fulfill the build reduces the potential attack surface.

52%
Ephemeral

52% of respondents indicated that each of their build steps are executed in ephemeral environments (i.e., once the build step is complete, the environment is discarded).

Extracting the output of the step and discarding the container/VM in which it was built ensures environments don't become polluted through reuse.

47%
Isolated

47% of respondents indicated that each of their build steps are executed in isolated environments (i.e., each build step executes independently).

By isolating each build-step container/VM, you can ensure that each build is free from influence by other build instances.

Continued on next page...

...continued from previous page

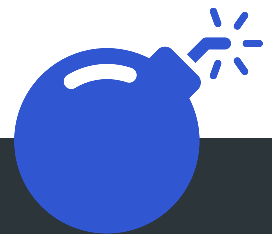
30%
Hermetically Sealed

30% of respondents indicated that their build steps are executed in hermetically sealed environments (i.e., environments that have no public network access).
All dependencies should be available locally via an immutable reference. Eliminating public network access to a container/VM eliminates the possibility of including remote resources, as may happen in cases of dependency confusion.

22%
Reproducible

22% of respondents indicated that their builds are reproducible.
Simply put, the same "bits" input should result in the same "bits" output. If they don't, there is no guarantee the artifacts you're working with haven't changed from build to build.

While most respondents agree on the use of ephemeral environments, such as containers or VMs in which to run their builds, there is less consensus around how isolated or sealed off those environments need to be, let alone where or even how much manual intervention is required to run a build.

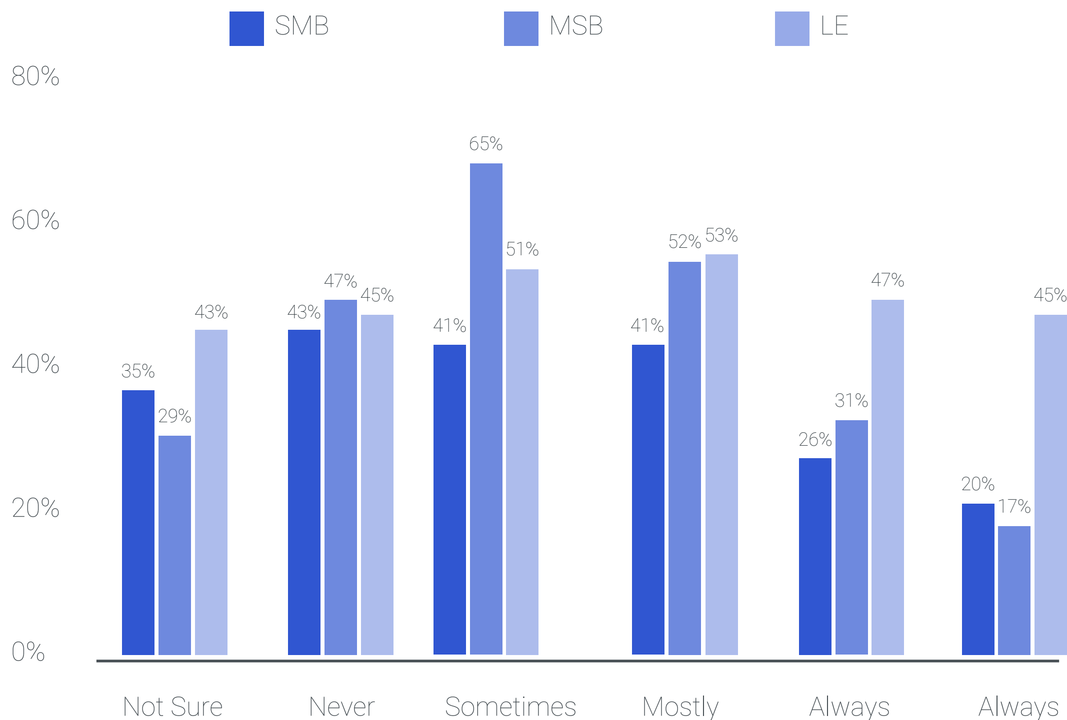


Only 22% of respondents are capable of creating reproducible builds.

The implication is that their organizations are unable to verify that the source code was unaltered when the original build was produced.

As a result, these organizations could be using compromised code and never know it until they (or their customers) get hacked.

Build Controls By Organization Size



Compared to SMBs and MSBs, LEs are far more consistent about implementing a wide range of best practices when it comes to building securely from source code.

And when it comes to reproducible builds, LEs are more than twice as likely to implement them.

In ActiveState's experience, security-conscious LEs place a strong emphasis on reproducible builds, often creating dedicated, experienced build teams that centrally supply all the runtime environments required by the enterprise's numerous projects. However, with less than half of LE respondents supporting reproducible builds, even the largest organizations still have a long way to go to secure their supply chain.

PART 4

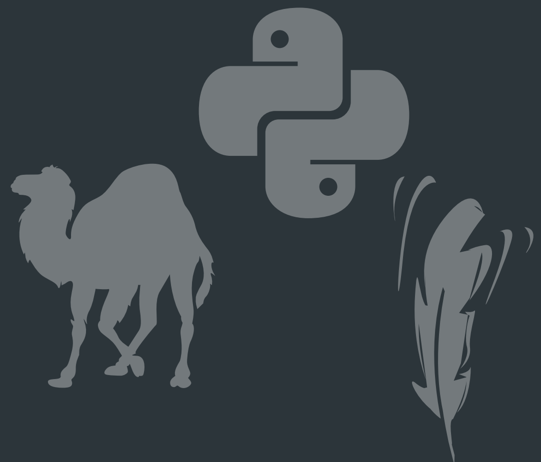
Run Controls

Respondents were asked what kinds of controls they have in place to ensure the components they develop their software with are secure. Traditionally, cyberattacks have focused on software being run in production environments, which is where most organizations place their greatest security emphasis. But bad actors have become wise to this strategy, and are now targeting less secure environments where software is run, such as CI/CD pipelines or even dev environments.

But if you've done your homework to ensure the security and integrity of your import and build processes, running code in development, test and even production environments should also be secure.

The best way to ensure the security of the components you run is to ensure they're signed either by your own organization or a trusted third party. **Code signing has been around for decades, and is widely considered a best practice to ensure that code:**

- **Was created by the signing entity (typically, the author of the software).**
- **Has not been altered or corrupted since the code was signed.**



But the real value of signed code is the establishment of trust. Trusted software vendors are an essential ingredient in creating effective cybersecurity at any security conscious organization.

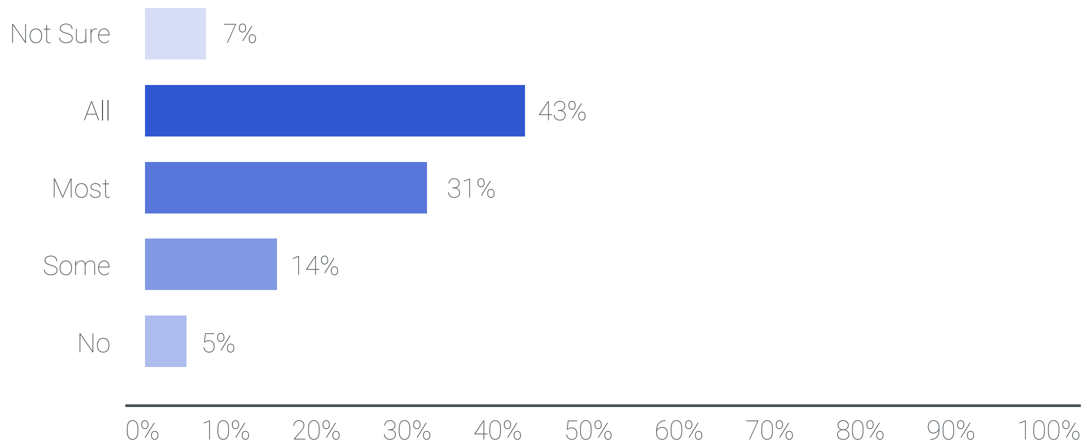
The technique of digital signing is a best practice that lets downstream consumers have confidence that the signed software originated with a trusted vendor, and that it hasn't been tampered with.

While the ActiveState Platform doesn't yet sign the packages it automatically builds from source code (coming soon!), it does verify all checksums internally so you can be confident your developers:

- Are always working with verified packages that have been built by ActiveState from source code, rather than installing pre-built, public binaries.
- Are working with a Python, Perl or Tcl-based development environment whose vulnerability status is always known, and who are empowered to simply point-and-click to automatically rebuild a secure version of their environment.

See how easy it is to shift security left with the ActiveState Platform >

Q6 - Do you work with signed packages?



7% **Not Sure** 7% of respondents were unsure whether they were working with signed packages.

43% **All** 43% of respondents indicated that they only work with signed packages. *Working with packages signed by a secure, internal build service, or else a trusted vendor is the best way to ensure components haven't been tampered with.*

31% **Most** 31% of respondents indicated that the majority of packages they work with are signed by a trusted entity.

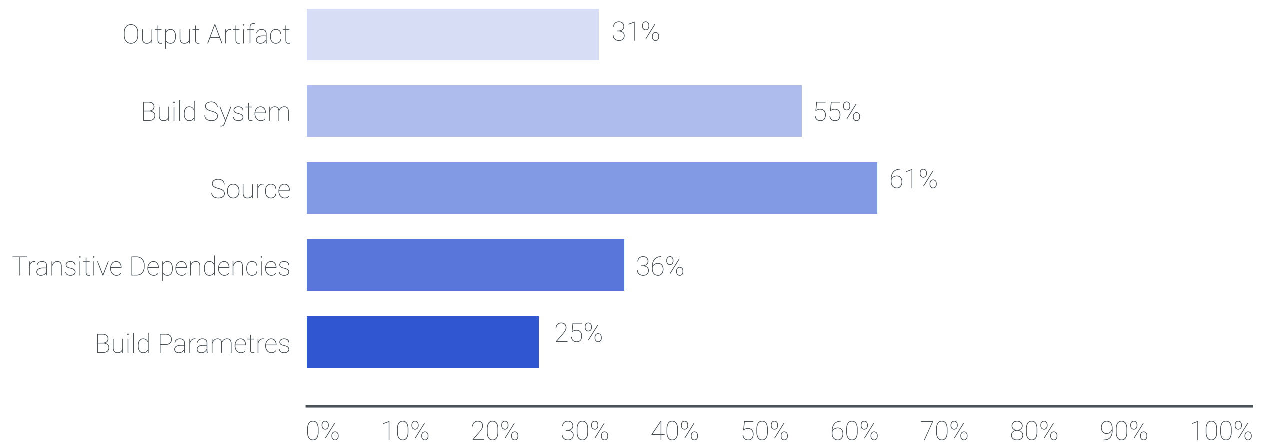
14% **Some** 14% of respondents indicated that they work with at least some signed packages.

5% **No** 5% of respondents indicated that they never use signed packages.

More than 80% of respondents work with at least some signed packages.

However, as the SolarWinds hack proved, signing is no guarantee that the software hasn't been compromised prior to the signing service.

Q7 - Does the signature include the following information via a cryptographic hash? Check all that apply.



Digital signatures should be generated from a private key accessible only to the build service. Digital signatures can be used to attest to the authenticity and integrity of the signed component in a number of ways, ensuring that it has not been compromised.

31%
Output Artifact

31% of respondents indicated that the signature identifies the built component.

Like the other information in a signature, the artifact is typically identified by a SHA-256 cryptographic hash.

55%
Build System

55% of respondents indicated that the signature identifies the build system used to create the component.

This is the entity that performed the build, and may be a CI/CD system or just a user's machine.

61%
Source

61% of respondents indicated that the signature contained an immutable reference to at least the build script.

This is typically a link to the build script in a version control system.

Continued on next page...

...continued from previous page

36%
Transitive Dependencies

36% of respondents indicated that the signature provided provenance (i.e., the source) for all transitive dependencies.

Transitive dependencies are dependencies of dependencies, and can shift over time as top-level dependencies evolve.

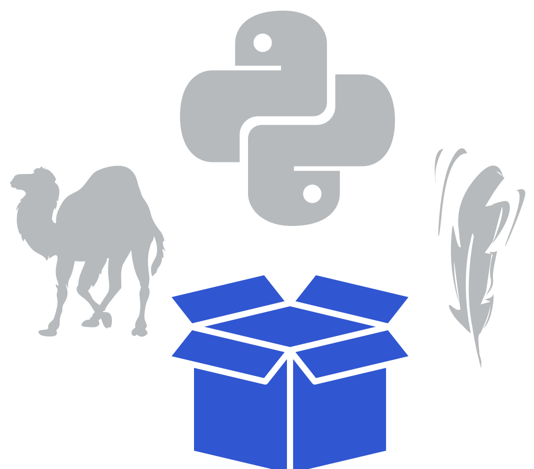
25%
Build Parameters

25% of respondents indicated that the signature identifies the build parameters (if any) with which the artifact was created.

Some components offer a number of user-controlled parameters or switches that can dramatically affect the functionality and/or performance of the component.

Surprisingly few participants indicated that the digital signature identified the output artifact. This is a "table stakes" item that should be included to ensure built components are properly identifiable.

On the other hand, the lower results for build parameters and transitive dependencies are expected, as they are typically seen as advanced requirements.



Key Takeaways

The security of the software supply chain has largely been ignored to date. Organizations have been distracted by ransomware and primarily focused on vulnerabilities. As a result, they have largely ignored the security and integrity of their software development processes.

1. **Supply Chain Security Immaturity**
Implementation of best practice controls to ensure the security and integrity of software development processes does not match the growing supply chain threat. Much more work needs to be done in 2022 to ensure software development organizations and their downstream customers can credibly avoid being compromised by bad actors.

2. **Build Reproducibility**
For those organizations that build components from source, by far and away the most worrying result from our 2021 survey is the lack of reproducible builds. Without reproducibility, no built artifact can be deemed secure. This should be a top priority for software development organizations in 2022.

3. **Public Repository Trust**
Open source organizations are making great strides to improve the security of their public repositories, but the reality is that they are still the wild west where anything goes. Unfortunately, survey results indicate that a worryingly high proportion of organizations continue to implicitly trust open source repositories. Organizations that work with public repositories should focus on implementing robust import controls in 2022.

The ActiveState Platform can help organizations secure their software supply chain by providing a turnkey secure supply chain for open source languages like Python, Perl, Ruby, and Tcl. It implements many of the key import, build and run controls discussed in this Survey, as well as features isted below.

- A comprehensive **Software Bill Of Materials** (SBOM) for each of your projects.
- A **Secure Build Service** that not only creates reproducible builds, but also provides provenance for all built components.
- **Software Integrity** that ensures your existing import, build and run processes haven't been compromised.
- **Automated Vulnerability Remediation** that not only identifies vulnerabilities, but also allows developers to resolve them in minutes, not days.

For more survey-related updates and the latest softwares supply chain security resources, head to:

activestate.com/software-supply-chain-security/

The Activestate Platform can help ensure the integrity and security of the open source software organizations use to develop their digital products and services.

Try the ActiveState Platform yourself by getting started for free. Or let us show you just how quick and easy it can be to secure your software supply chain.

Get a Demo

About ActiveState

ActiveState helps enterprises manage the complexity and risk of using open source languages at scale, while giving developers the kinds of tools they love to use. More than 2 million developers and 97% of Fortune 1000 enterprises use ActiveState to support mission-critical systems and speed up software development while enhancing oversight and increasing quality.

ActiveState

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

©2021 ActiveState Software Inc. All rights reserved. ActiveState®, ActivePerl®, ActiveTcl®, ActivePython®, Komodo®, ActiveGo™, ActiveRuby™, ActiveNode™, ActiveLua™, and The Open Source Languages Company™ are all trademarks of ActiveState.