The Open Source Supply Chain Can Be Fixed

Best Practices for Mitigating the Risk of Software Supply Chain Threats

ActiveState

Executive Summary

For most software vendors, the way they import, build and consume open source code is at odds with their software development security and integrity goals. Simply put, too many organizations:

> Have immature supply chain security processes and policies in place.
>
> Implicitly trust the open source components they obtain from public repositories.
>
> Do not build open source artifacts from source code in a reproducible way.
>
> Lack key, deterministic controls to ensure the open source components they use can be trusted.

The problem lies in the fact that the software supply chain for most vendors is extremely complex, being both wide and deep. As a result, software vendors may be unduly exposing their customers to compromise when (not if) they suffer an open source supply chain attack.

This white paper is a companion piece to our Software Supply Chain Security survey, which found that supply chain security across the software industry as a whole is far more immature than expected. This paper examines the threats that exist at each level of the supply chain, and suggests best practices that can be implemented to mitigate the risk associated with working with open source software.

The State of Open Source Supply Chain Security

After more than thirty years, open source software has become a vital resource at virtually all enterprises, from famously strict organizations like NASA to innovation-focused software vendors to security-conscious banking and finance companies. But open source software continues to be a dual edged sword, fostering both innovation and security issues. Each and every year, open source security issues continue to expose enterprises to cyberattack, some with catastrophic consequences. But essentially the model has remained unchanged: bad actors search for needle-sized exploits in the production environments of a worldwide haystack of corporations.

All that changed in December 2020 with the SolarWinds hack, which showed that one compromised development environment at a key software vendor can result in trojanized patches and software updates being propagated downstream to tens of thousands of customers, potentially exposing them all to attack. In other words, bad actors have discovered economies of scale: a single cyberattack against a popular software vendor can grant global corporate access in today's internet-connected world.

And since most software vendors have poor controls (if any) around internally auditing their software build infrastructure and development processes, these kinds of supply chain exploits can go undetected for months or even years.

All of which led to President Biden's Executive Order in 2021, which states:

> "The private sector must adapt to the continuously changing threat environment, [and] ensure its products are built and operate securely."

More specifically, organizations need to be:

> "Ensuring and attesting...to the integrity and provenance of open source software used within any portion of a product."

Unfortunately, software supply chains tend to be very large and complex, extending across multiple software development processes, including:

> **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.
>
> **Consume** – the process of working with, testing and running built artifacts in development, test and production environments.

The breadth and depth of the open source supply chain affords multiple points of entry for malicious actors who will always look for the weakest link in the chain to exploit. This often means that "ensuring the integrity and provenance of open source" is too complicated and costly a problem for most organizations to solve on their own since no out-of-the-box solution exists. As our Software Supply Chain Security survey shows, a number of anti-patterns have taken root:

More than

32%

of organizations implicitly trust public repositories that offer no guarantees as to the security and integrity of the open source components they provide.

Only

22%

of organizations create reproducible builds. Without reproducibility, no package built from source code should be deemed secure.

Only

11%

of organizations achieved a grade of “Excellent” for their existing supply chain security practices.

This white paper can help Industry leaders responsible for the security and integrity of the software they produce to investigate the current state of their development processes, understand the threats at each stage, and create a plan to improve their open source supply chain security.

Importing Open Source Code Securely

WIth the creation of increasingly sophisticated software applications comes even greater reliance on open source components that shorten software delivery times and reduce developer workload. Unfortunately, the software industry’s growing reliance on open source has not been matched by an increase in protections to ensure packages installed from public repositories are malware free. On the contrary, that work has been left up to each individual organization to tackle on their own.

But according to ActiveState’s Supply Chain Security Survey, failing to validate open source code before it’s imported into the organization is a common occurrence, no matter the size of the organization. The problem is a difficult one to solve since it involves:

> **Breadth** - most organizations work with more than one open source language, and import their code from more than one public repository. Because there are no industry-wide standards in place today, each language and repository must be treated uniquely.
>
> **Depth** - There is a large set of best-practice security and integrity controls you could implement in order to scrutinize the open source components you import. How far you go down this rabbit hole is largely dependent on your appetite for risk, but also your time and resources, as well.
>
> **Change** - no supply chain is ever set in stone. Open source authors change, and the packages they produce are constantly updated, become vulnerable, and get patched. Languages go EOL, repositories move, trusted vendors change, etc. Keeping up with it all will, once again, demand time and resources.
>
> **Iterative Reuse** - one codebase’s supply chain is n number of components. If that codebase is then incorporated in another codebase with its own supply chain, that’s n+n+1 supply chains. In other words, a seemingly endless number of supply chains to secure all the way down.

Given this level of complexity, it’s no wonder that one-third of software vendors choose to just implicitly trust the public repository. After all, most public repositories are quick to remove typosquatted and brandjacked packages, as well as those that contain malware/trojans once brought to their attention. Unfortunately, repository updates can often be too late for 80% of software projects that never update third-party libraries after including them in their codebase.

Software professionals concerned with the security and integrity of the open source they import need to balance risk and control in order to create a pragmatic, maintainable solution. Recommended best practices include creating an automated system that can (at a minimum):

- Verify the identity of uploader(s)/ author(s) and reviewers of an open source component
- Verify the timestamp and revision history
- Verify the URL/ immutable reference
- Flag known vulnerabilities
- Quarantine suspect components for further investigation

While implementing these kind of security controls is a non-trivial exercise, it can help address some of the more common threats, including:

- **Typosquatting** – the practice of attackers submitting a package to an open source repository that is named similar to a popular, existing package.
- **Dependency Confusion** – occurs when a build system pulls in a similarly named dependency from a public repository rather than your private repository.
- **Author Impersonation** – can occur when author or repository credentials are compromised.

A more cost effective alternative can be sourcing prebuilt packages from a trusted vendor, which is an approach that 36% of organizations (on average) who responded to our supply chain survey have already adopted.

Building Open Source Code Securely

Open source software is typically distributed in either a prebuilt form or else as source code, but prebuilt components (even those sourced from trusted vendors) may contain things other than just compiled source code. For example, they may contain debug capabilities. This lack of a one-to-one correlation between source code and built artifact increases the chance that malicious code may be included in the final component. As a result, it's often safer and more reliable to build open source packages yourself.

However, according to ActiveState's Supply Chain Security Survey, only one-quarter of organizations, on average, build all the open source packages they require from source code. Worse, only 22% of organizations implement a reproducible build process, making it very difficult to trust that the packages were built securely in the first place.

The problem lies in the fact that most organizations typically build the open source components they need only once per project (unless a critical vulnerability is discovered), and usually perform those builds on a developer's machine rather than a dedicated build service. This "one-off" methodology has led to a reproducibility crisis in many scientific disciplines, and can potentially expose software vendors to such threats as:

Tampering with build scripts, which are typically editable by any user on the developer's system.

Build steps that overstep their bounds and attempt to exploit common OS services present on a developer's machine.

Dependencies that dynamically include remote resources, since developer systems are typically connected to the internet.

And so on. Instead of treating source code builds like an artisan's workshop, they should really be treated like assembly lines in a modern day factory. Automation is key to connecting all the steps in the process in a deterministic way. In addition, the same defense-in-depth paradigm currently used for production environments should also be applied here. For example:

Build Steps - each build step should have a single responsibility, such as retrieving source code, building an artifact, etc. When that responsibility is fulfilled, the output is checked and, if required, passed to the next step.

Infrastructure - all build steps should have dedicated resources, such as a hardened container in which to execute. Once the step is complete, all those resources should be discarded, preventing contamination of subsequent steps.

Environments - each build step requires an environment in which to run. Care must be taken to ensure the runtime environment is minimized, and the container only includes components that are absolutely required (such as compilers) in order to shrink the attack surface.

Provenance - each build must be dependency complete, and each dependency traceable to the originating source. An out-of-band Software Bill Of Materials (SBOM) should be produced for independent verification.

Service/Network - creating a dedicated build service that runs on a segmented network with no internet access will limit local exploits and remote tampering/intrusion.

Unfortunately, implementing these kinds of best practices is far too costly for those organizations that commonly build open source packages only once per project. As an alternative, you might want to consider employing a commercially-available secure build service that features many of the best practices listed here for a fraction of the cost to build and maintain them yourself.

Consuming Open Source Code Securely

Typically, built artifacts like open source packages are stored and distributed via some kind of binary or artifact repository. The packages must be verified and secured against tampering to ensure they remain secure between the time they're imported and the time your developers incorporate them into their codebase. As such, the repository you employ should feature:

Trust - to help establish trust, ensure your repository is populated only by signed components. Signing is the most common trust mechanism for open source packages, allowing you to verify that they haven't been tampered with between the time they were signed and the time they are used by your development team.

Security - in order to ensure the security of the open source packages you use they must be continuously monitored for vulnerabilities. Vulnerability scanning is typically provided by Software Composition Analysis (SCA), which may be incorporated into the repository's functionality, or else invoked as a separate tool.

Integrity - verifying the integrity of an artifact typically takes the form of a checksum verification. This step is usually skipped for signed components, which assumes that the checksums have already been verified prior to signing.

> **Note:** even signed components can be compromised, as proven by the SolarWinds hack. Verification at runtime (i.e., prior to a package being loaded into the interpreter) may be a desirable feature for more security conscious organizations.

Freshness - in order to understand when a component becomes outdated, you'll need to be able to determine not only how old the version of a package is relative to more recent versions, but also when it was last imported/updated.

According to ActiveState's Supply Chain Security Survey, 87% of organizations work with at least some signed packages from a trusted source, but only 43% (or about half of them) work exclusively with signed packages. As a result, most organizations potentially expose themselves to:

Compromised Prebuilt Artifacts - as previously noted, prebuilt components may be compromised by malware at build time. But prebuilt packages can also be compromised at any point after creation if they haven't been signed. The message is clear with unsigned packages: user beware.

To counter the threat posed by prebuilt components, organizations should either:

Work only with signed packages from a trusted provider that can help you populate your repository with secure, trusted artifacts, or

Build and sign all packages from vetted source code using a secure build service, as introduced in the previous section.

Without the proper controls in place, organizations open themselves to attack at multiple points across the import, build, and consume processes.

Conclusions

Like it or not, software vendors are now the frontline of security for their customers. Security and software professionals alike must take steps to ensure their existing software development processes have not been compromised, and to secure their software supply chain going forward.

As software supply chain complexity increases, verifying the security and integrity of the software development lifecycle must rely more and more on automated validation of key software development processes when importing, building and consuming open source components. To date, this has been accomplished by cobbling together point solutions from multiple vendors, as well as implementing best practices at a cost of significant time and resources.

While no vendor currently provides a comprehensive, end-to-end supply chain solution, some like ActiveState have begun to offer a turnkey solution for use cases involving open source languages like Python, Perl, Tcl and Ruby. Such an out-of-the-box solution can save enterprises significant time, resources and money when compared to a multi-vendor approach. You can learn more about our approach at www.activestate.com.

Next Steps

Get the complete report for ActiveState's Supply Chain Security Survey:
https://www.activestate.com/resources/state-of-software-supply-chain-security/

Book a demo and learn how ActiveState secures you open source:
https://www.activestate.com/solutions/enterprise-security/

Talk to us

ActiveState

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