Beyond SBOMs: providing strong security guarantees for future software supply chains using in-toto

Standards and Guidelines to Enhance Software Supply Chain Security

(Position statement for the "Guidelines for software integrity chains and provenance" Area)

Santiago Torres-Arias
Department of Electrical and Computer Engineering, Purdue University
santiagotorres@purdue.edu

May 27, 2021

1 Introduction

Modern software is built through a complex series of steps called a *software supply chain*. These steps are performed as the software is written, tested, built, packaged, localized, obfuscated, optimized, and distributed. In a typical software supply chain, these steps are "chained" together to transform (e.g., compilation) or verify the state (e.g., the code quality) of the project in order to drive it into a *delivered product*, i.e., the finished software that will be installed on a device. Usually, the software supply chain starts with the inclusion of code and other assets (icons, documentation, etc.) in a version control system. The software supply chain ends with the creation, testing and distribution of a delivered product.

Securing the supply chain is crucial to the overall security of a software product. An attacker who is able to control any step in this chain may be able to modify its output for malicious reasons that can range from introducing backdoors in the source code to including vulnerable libraries in the delivered product. Hence, attacks on the software supply chain are an impactful mechanism for an attacker to affect many users at once. Moreover, attacks against steps of the software supply chain are difficult to identify, as they misuse processes that are normally trusted.

Unfortunately, such attacks are common occurrences, have high impact, and have experienced a spike in recent years. Attackers have been able to infiltrate version control systems, including getting commit access to the Linux kernel, stealing Google's search engine code, and putting a backdoor in Juniper routers. Furthermore, attackers have used software updaters to launch attacks, with Microsoft, Adobe, Google, and Linux distributions all showing history of compromise. Perhaps most troubling are several attacks in which nation states, such as Iran, North Korea, China and Ukraine, have used software supply chain compromises to target their own citizens and political enemies. This troubling trend of course culminated with the SolarWinds compromise, that has affected critical organizations within the US government.

Currently, supply chain security strategies are limited to securing each individual step within it. For example, Git commit signing controls which developers can modify a repository, reproducible builds enables multiple parties to build software from source and verify they received the same result, and there are a myriad of security systems that protect software delivery. These building blocks help to secure an individual step in the process.

Although the security of each individual step is critical, such efforts can be undone if attackers can modify the output of a step before it is fed to the next one in the chain. These piecemeal measures by themselves can not stop malicious actors because there is no mechanism to verify that: 1) the correct steps were followed and 2) that tampering did not occur in between steps. For example a web server compromise was enough to allow hackers to redirect user downloads to a modified Linux Mint disk image, even though every single package in the image was signed and the image checksums on the site did not match. Though this was a trivial compromise, it allowed attackers to build a hundred-host botnet in a couple of hours due to the lack of verification on the tampered image. While Software Bills of Materials (SBOMs) provide a unique property of software transparency, they fall short without adequate supply chain integrity mechanisms in place.

As such, further mechanisms are required in order to ensure the correct application of software supply chain policies at both , the individual step level (i.e., an operation within the supply chain) and at the inter-step level (i.e., as software artifacts flow throughout the chain, from one step to another), is crucial to prevent further software supply chain compromises. This is why we designed in-toto to provide these properties throughout various software supply chains. in-toto is currently used throughout various ecosystems, and various software vendors to protect thousands of companies and millions of users.

2 Guidelines for software integrity chains and provenance

As outlined in the Executive Order Sections 4(e)(ii, and vi), a need of evidence of conformance to processes as well as the ability to evaluate security policies regarding software development are crucial to improve software supply chain security. To build a secure software supply chain that can combat the aforementioned threats, we envision that the following security goals would need to be achieved.

Property 1: Supply Chain Layout Integrity This property requires that all of the steps defined in a supply chain are performed in the specified order. This means that no steps can be added or removed, and no steps can be reordered. This tightly relates to the necessity of *communicating software supply chain processes* information in a trustworthy fashion to consumers of software. In in-toto, this is achieved throughout a software supply chain policy file (also called an in-toto layout). This way, an authoritative source (such as a government agency) can stipulate the expected processes to be carried out. Further, policies are cryptographically signed to ensure authenticity.

Property 2: Step Authentication To provide Step Authentication, we require that steps can only be performed by the intended parties. No party can perform a step unless it is given explicit permission to do so. Further, no delivered products can be released unless all steps have been performed by the right party (e.g., no releases can be made without a signoff by a release engineer, which would stop accidental development releases). This requirement is crucial and goes beyond a regular Software Bill of Materials, as it dis-aggregates trust to different software information providers. In other words, this allows for actors within a supply chain to provide information about parts of the process they are trusted to carry out.

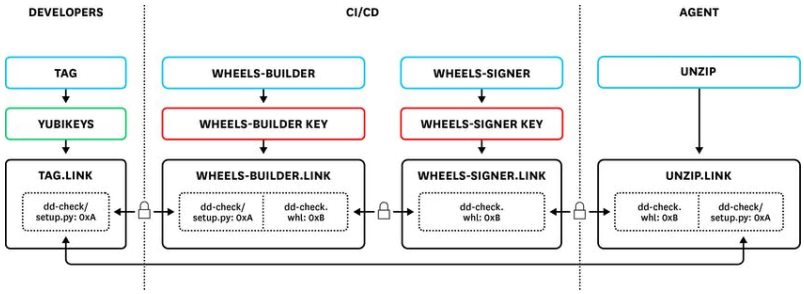

Figure 1: The in-toto deployment at Datadog to provide a their "tamper evident CI/CD system"

Property 3: Artifact-Flow Integrity All of the software artifacts (e.g., binaries, source code, OS images) created, transformed, and used by steps must not be altered in-between steps. This means that, as shown in Figure 1 if step *develop* creates a file `setup.py` and step CI/CD uses it, step CI/CD must use the exact file `setup.py` created by step *develop*. It must not use, for example, an earlier version of the file created in a prior run. By providing Artifact-Flow Integrity, we are better able to provenance information that goes beyond the source of a software artifact, but also its contents, which version, and more.

Artifact flow integrity is preserved by collecting evidence in in-toto in the shape of *link metadata*, which are signed attestations created by actors in the software supply chain. These attestations collect information as the step in the supply chain is carried out so as to show that it was carried out properly. Finally, the attestation is also signed to allow for non-repudiation and authentication of the information provided (the latter also being necessary to ensure step authentication).

2.1 Providing Mechanisms to Ensure These Properties

We designed in-toto to provide these properties and accommodate current software development practices. To use in-toto, you can enable features on multiple off-the-shelf tools (e.g., via a Jenkins plugin or Tekton) to generate and verify evidence of software supply chain compliance. In principle, in-toto is a series of tools, and a document formats to enable actors in the supply chain to exchange software supply chain information (e.g., SBOMs) and to validate that the internal properties of the evidence provided.

3 Conclusion

We presented a series of security properties of paramount importance to improve the security of software supply chains. Although we believe that any tool that provides the previously-mentioned properties will fit the bill, in-toto is the first tool to do so. Further, as a thriving open source software community around this topic area, it serves as a hub for multiple existing players to exchange ideas and extensions to the original design. We invite interested parties to read more about in-toto on `https://in-toto.io`.