Trusting Trust: Humans in the Software Supply Chain Loop

Laurie Williams
Associate Editor in Chief

> To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software. . . . You can't trust code that you did not totally create yourself.
>
> —*Ken Thompson, Turing Award Lecture, 1984*[1]

The modern world relies on digital innovation in almost every human endeavor and for our critical infrastructure. Digital innovation has accelerated substantially as software is increasingly built on top of layers of reusable abstractions, including libraries, frameworks, and cloud infrastructure, which often lie outside an organization's trust boundary. Where previous teams of engineers invested months, today, beginners can write intelligent smartphone apps with a few lines of code. Leveraging these reusable abstractions gives rise to software supply chains, where software products include "upstream" components as well as dependencies, created and modified by others, that, again, often include their own transitive dependencies. Most of these dependencies are open source projects.

Where previous teams of engineers invested months, today, beginners can write intelligent smartphone apps with a few lines of code.

However, with all of the power that software supply chains and open source infrastructure provide also come risks. Software developers did not anticipate how the software supply chain would become a deliberate attack vector. The software industry has moved from passive adversaries finding and exploiting vulnerabilities contributed by honest, well-intentioned developers to a new generation of software supply chain attacks where attackers aggressively implant vulnerabilities directly into infrastructure software (e.g., libraries or tools) and infect build and deployment pipelines.

Sonatype[2] reports a 650% year-over-year increase in detected supply chain attacks (on top of a 430% increase in 2020) targeted toward upstream open source repositories. The U.S. government is so concerned about software supply chain security deficiencies that a whole section of Executive Order 14028[3] (*Improving the Nation's Cybersecurity*), issued on 12 May 2021, is focused on new compliance requirements for government vendors to enhance supply chain security.

Historically, when people thought about the software supply chain attack surface, they thought about the many components that make up a product. More recently, the software supply chain attack surface increasingly encompasses the build infrastructure. In this article, I bring back the progressive thoughts of Ken Thompson and place humans in the software supply chain—as both developers with and without malicious intent and as part of the solution to software supply chain security.

Components and the Software Supply Chain

Attackers exploit vulnerabilities in components. For example, in late 2021, an accidentally injected vulnerability in the popular logging library log4j, used by more than 35,000 Java packages, allowed an attacker to perform

Digital Object Identifier 10.1109/MSEC.2022.3173123
Date of current version: 6 September 2022

remote code execution by exploiting an insecure Java Naming and Directory Interface (JNDI) lookup feature, which is enabled by default in many versions of the library. In 2022, as an instance of protestware, a developer maliciously injected code into the node-ipc package, with more than 700,000 weekly downloads. The initial version of the malicious code attempted to geolocate where the code is running, and, if it discovers it is running within Russia or Belarus, then it attempts to replace the contents of every file on the system with a Unicode heart character.

To manage the component-based supply chain risks, development teams (those humans!) are challenged to update their components when vulnerabilities are found and choose safe components.[4,5] Software composition analysis (SCA) tools aid in identifying vulnerable components. SolarWinds was a wakeup call that reminded security experts that quickly updating to the latest version of a dependency might also introduce malicious code or vulnerable code that may be exploitable. Projects such as Open Source Security Foundation (OpenSSF) Metrics and deps.dev are emerging to provide metrics on open source components to aid teams in making informed choices on components.

Build Infrastructure and the Software Supply Chain

In an emerging attack vector, attackers are infiltrating the build infrastructure. In 2020, the build process for the SolarWinds network management tool, Orion, which is used to manage routers and switches inside corporate networks, was maliciously subverted to distribute malware to create backdoors on victims' networks. This malware enabled spying on at least 100 companies and nine U.S. government agencies, including the Centers for Disease Control and Prevention, U.S. Department of Homeland Security, U.S. Department of Justice, Pentagon, and U.S. Department of State.

To manage the component-based supply chain risks, development teams (those humans!) are challenged to update their components when vulnerabilities are found and choose safe components.

In 2021, attackers used a mistake in how Codecov built docker images to modify a script, which allowed them to send the environment variables from the continuous integration (CI) environment of Codecov customers to a remote server. The attackers accessed private Git repositories from the Git credentials in the CI environment and exploited the secrets and data within.

To manage the build infrastructure-based supply chain risks, development teams (those humans!) are challenged to secure their build infrastructure, considered to be a huge open-ended challenge.[4] The Supply Chain Levels for Software Artifacts [SLSA (pronounced "salsa")] framework provides a checklist of standards for reasoning about the build process. SLSA is based on Google's internal processes and defines four levels, beginning with simply having a scripted build and recording provenance information and ending with using an ephemeral, isolated, parameterless, and hermetic build environment. Bonus points are given if the build is reproducible; i.e., two builds produce bit-for-bit identical output.

Additionally, the industry is increasingly moving toward the use of reproducible builds to verify that the source code was unaltered when the original build was produced. There are a number of efforts on this front. For example, the Debian-initiated https://reproducible-builds.org effort has characterized and classified the many types of nondeterminism that can be introduced during the build process.

Humans and the Software Supply Chain: Attackers

In the supply chain, we can consider attackers as developers who act with malicious intent. Attackers aggressively implant vulnerabilities directly into components, infrastructure, software (e.g., libraries and tools) and infect build and deployment pipelines. Back to Ken Thompson's quote about trusting trust, "Perhaps it is more important to trust the people who wrote the software. . . . You can't trust code that you did not totally create yourself".[1] In reality, innovation would grind to a halt in an organization that decides it can't trust any open source code due to the risk of malicious code injection. That would be like Tesla deciding it can't trust its screw manufacturer and manufacturing its own screws.

As an industry, we need to develop models for identifying malicious actors and malicious code injection. Because the attackers act in ways that well-meaning developers do, we are challenged to identify their actions. Models are beginning to emerge to identify weak leaks signals that arouse suspicion, such as the identification that a component maintainer's domain is expired and does not have two-factor authentication (2FA) authentication set up on the account. An attacker can

relatively easily hijack that component or a component that has an install script.[6]

More signals that indicate malicious activity need to be developed and verified. We can't stop the attackers, but we can make it harder for them. For example, typosquatting was a very popular attack vector. As ecosystems automated the identification and takedown of rogue typosquatted packages, attackers have moved away from this attack vector. However, we play "cat and mouse"—with the plethora of weaknesses in most applications and infrastructures, moving to a different spot on the attack surface is not a big deal for the attacker but can be a big deal for the defender.

Humans and the Software Supply Chain: Software Developers

> Some might argue that it's almost too easy to introduce a new dependency into your software systems. I'm definitely guilty of this in my previous life as an engineer. I remember pulling in random Python packages when building my own websites and not putting any thought into security. It should be fine if so many other people are using the same package, right?
>
> —*Kim Lewandowski, Google Product Manager*[7]

In the supply chain, we can consider software developers as well-intentioned actors in the supply chain who are just trying to deliver functionality but sometimes make mistakes that enable security breaches. The quote from Lewandowski epitomizes a common but now naive belief held by developers. While developers may feel a popular package must be secure, attackers intentionally leverage their efforts by injecting malicious code in packages with many dependents and a high download frequency. A popular package may, in fact, be more risky.

While developers may feel a popular package must be secure, attackers intentionally leverage their efforts by injecting malicious code in packages with many dependents and a high download frequency.

Predominantly measured by his or her ability to deliver functionality, a developer can be overwhelmed and overloaded by the additional compliance restrictions and the notifications from supply chain security tools. For example, SCA tools, such as Dependabot, send email and pull requests for every dependency and transitive dependency in a package that has a discovered vulnerability. The vulnerability may be in a part of a component not used by the package, and an automatic acceptance of the pull request may break functionality and/or pose additional security risk—increasing, not lowering, the overall risk.

Additionally, package maintainers may be overloaded, which may lead to hasty and possibly dangerous decisions around accepting new maintainers and pull requests. (They are humans, after all.) For example, a study on the npm ecosystem revealed that the top 1% of maintainers own an average number of 180.3 packages, with an average of 4,010 direct dependents.[6] That's a lot!

The Humans as First-Class Players in the Secure Software Supply Chain Solution

For humans to be the solution to supply chain security, developers need education, guidance, and risk-based tools. Part of this education is just the awareness that not all open source software can be trusted. Major players in the industry are already coming together via a number of projects. Both SLSA (mentioned earlier) and the Open Web Application Security Project (OWASP) Software Component Verification Standard provide frameworks for identifying activities, controls, and best practices that can help in identifying and reducing risk in a software supply chain. Additional projects include OpenSSF (mentioned earlier); sigstore; and in-toto,[8] a joint industry–academia project that helps shed light on code-to-binary provenance. Package managers and researchers are exploring logic-based and machine learning-based mechanisms for identifying malicious code and malicious contributors. Currently, this machine learning-based sorting to identify bad hygiene has a high signal-to-noise ratio and presents technical challenges, so more work is needed.

Is it possible to trust trust? Can we develop mechanisms for software developers to trust code that we did not totally create ourselves in an informed manner? ■

Acknowledgments

My thinking on the role of people in the software supply chain was highly influenced by conversations with 30 practitioners during three Software Supply Chain Summits. These were coheld by William Enck and me,[4]

with extensive collaboration with Yasemin Acar, Michel Cuckier, William Enck, Alex Kapravelos, and Christian Kästner—especially Yasemin, who is one of the few researchers at the intersection of human factors and secure software development.

References

1. K. Thompson, "Reflections on trusting trust," *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984, doi: 10.1145/358198.358210.
2. "2021 State of the software supply chain," Sonatype, Fulton, MD, USA, Jul. 2021. https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021
3. "Executive order 14028: Improving the nation's cybersecurity," Federal Register, May 12, 2021. https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity
4. W. Enck and L. Williams, "Top five challenges in software supply chain security: Observations from 30 industry and government organizations," *IEEE Security Privacy*, vol. 20, no. 2, pp. 96–100, 2022, doi: 10.1109/MSEC.2022.3142338.
5. D. Drusinsky and J. Michael, "Obtaining trust in executable derivatives using crowdsourced critiques with blind signatures," *Computer*, vol. 53, no. 4, pp. 51–56, Apr. 2020, doi: 10.1109/MC.2020.2970819.
6. N. Zahan, L. A. Williams, T. Zimmermann, P. Godefroid, B. Murphy, and C. S. Maddila, "What are weak links in the NPM supply chain?" in *Proc. Int. Conf. Softw. Eng. Softw. Eng. Pract.*, 2022, to be published.
7. K. Lewandowski, "Security scorecards for open source projects," Open Source Security Foundation Nov. 6, 2020. [Online]. Available: https://openssf.lfprojects.linuxfoundation.org/blog/2020/11/06/security-scorecards-for-open-source-projects/
8. S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *Proc. USENIX Security Symp.*, Aug. 2019, pp. 1393–1410.

Digital Object Identifier 10.1109/MSEC.2022.3199547