A Viewpoint on Human Factors in Software Supply Chain Security

A Research Agenda

Marcel Fourné | Max Planck Institute for Security and Privacy
Dominik Wermke | North Carolina State University
Sascha Fahl | CISPA Helmholtz Center for Information Security
Yasemin Acar | Paderborn University

While securing dependencies and build systems is necessary, recent attacks have shown that developers are a commonly successfully attacked link in the chain. Therefore, a comprehensive approach that considers the human factor is crucial for effective software supply chain security.

Human factors, and especially developers, play an important role in securing the software supply chain (SSC).[1]

SSC vulnerabilities pose significant risks to organizations, historically being exploited by threat actors who target unpatched systems with known vulnerabilities.[2] Exploits targeting vulnerabilities like Heartbleed[3] and, more recently, Log4Shell[4] have highlighted weaknesses in both commercial and open source software, affecting both private and government enterprises and billions of end users. More recently, attackers directly exploited the SSC structure by targeting upstream dependencies and build systems to inject malicious code into downstream software, like in the security incident at SolarWinds.[5]

Several steps to address dependency and build chain problems have been proposed—updating and using trusted dependencies, using software bills of materials (SBOMs), securing the build process, and involving more industry participation.[6] However, securing these dependencies and build systems will likely not thwart all attacks: recent, headline-making attacks involved breaches of the SSC through one of its largest attack surfaces—individual developers. In January 2023, CircleCI disclosed that a cybercriminal had used malware on a CircleCI engineer's laptop to steal a valid, two-factor authentication (2FA)-backed single sign-on session, allowing the attacker to execute session cookie theft and impersonate the employee, gaining access to a subset of production systems.[7] In February 2023, password manager LastPass reported that a hacker stole corporate and customer data by infecting an employee's personal computer with key logger malware, giving the hacker access to the company's cloud storage and resulting in the theft of source code and customer vault data.[8] These incidents highlight that each individual

Digital Object Identifier 10.1109/MSEC.2023.3316569
Date of current version: 10 November 2023

developer participating in the SSC may have different technology and knowledge stacks, and humans making decisions about security and trust can put the SSC at risk.[9] We explore both how to empower stakeholders to secure the SSC by considering human factors and reducing developer overwhelm and also how to decrease the attack surface of individual software developers.

Human Factors in Supply Chain Research

Supply chain security (SCS) is a crucial area of concern for businesses operating in today's connected economy. However, as with many aspects of security, we think that the human factors involved in the design, implementation, and use of SCS measures have not received the attention they deserve. We believe that the usability of SCS measures for the people who will use them, especially developers, is a critically underinvestigated area of research. To address the need for more human-centered SCS approaches, we next propose multiple aspects for a high-level research agenda.

The lack of clear hierarchies and trust relationships across different software components can also make it easier for attackers to target and exploit vulnerabilities.

Motivations, Challenges, and Personal Risks

Integrating external software components into software projects presents unique challenges and risks for developers. The development of the external components often involves different developers with a diverse set of technology stacks, motivations, and development as well as communication cultures. There is a responsibility that comes with relying on external build components and processes—in principle transparent, in practice infeasible to completely vet across fragmented ecosystems. The lack of clear hierarchies and trust relationships across different software components can also make it easier for attackers to target and exploit vulnerabilities.

To address these challenges, we think it is necessary to research and promote systematic, usable communication across creators of different components along the software supply chain.

Usable Tooling

Ideally, new functionality is integrated into versatile tooling that developers are already using; it remains unlikely that tools with unique interfaces and functionalities will be widely adopted to check for individual security properties.

To create build artifacts that can be trusted in an informed manner, not through organizational authority, one needs to know everything included in the creation of said artifact; see the "trusting trust attack."[10]

While we applaud[11] the work of the reproducible-builds.org project, this should only be the baseline, and software build reproducibility—bootstrappability[12]—and diversified checking à la David A. Wheeler's "diverse double-compiling"[13] should be further security targets. Their benefit needs to be prevalent in deployed software and not limited to academic study to gain practical advantages for every end user. This prevalence may yet entail publishing the source code and complete build recipes of all software that may have an effect on user security, to have them automatically recompiled and the results checked after variation in the environment, bootstrapping the necessary dependencies cleanly.

All of this needs to be low effort for a wide range of developers with different technology stacks, so research is needed to explore which problems can be solved with tooling, and how tooling can communicate with its users in a way that allows for wide adoption.

Build Processes and Continuous Integration and Delivery

Today's build systems and continuous integration and delivery (CI/CD) pipelines can interact and chain with other systems and third-party services, allowing for the creation of complex, multistep build and distribution processes for software. But this complexity also increases the risk of misuse, misconfiguration, or leakage of secrets, and as software is increasingly being built and deployed using third-party services, these services are becoming high-value targets for attackers seeking to infect all customers and compromise their SSCs. This is highlighted by the recent CircleCI security incident, where an attacker used malware on an engineer's laptop to gain unauthorized access to CircleCI's production systems.

As build systems and CI/CD pipelines are becoming increasingly complex, it is essential to not overlook the

human factor in setting up, maintaining, and using them. Usability plays an important role in ensuring that developers can effectively and securely utilize and manage these complex systems: by establishing what makes them usable for stakeholders, such as clear documentation, user-friendly interfaces, and effective training materials, we as researchers can reduce the risk of misconfigurations and other vulnerabilities while allowing developers to maintain productivity and workflow efficiency.

Dependencies

Allowing developers to leverage external dependencies as building blocks for their software, e.g., from package repositories like npm or PyPI, is an important advantage of the SSC. However, the reliance on external repositories also introduces new attack surfaces as recent typosquatting and account-takeover attacks have shown. In response, Python's PyPI and GitHub have begun requiring 2FA for developer accounts with critical projects. While such approaches may increase the overall security of package repositories and dependencies, it is crucial to also consider their usability. For example, if 2FA is required, but the authentication process is too complicated or time-consuming, developers may find ways to bypass it, which would undermine the intended security benefits. Developers need to keep track of changes in dependencies, so good communication practices about those changes are necessary.

> **We believe that the usability of SCS measures for the people who will use them, especially developers, is a critically underinvestigated area of research.**

By examining the common challenges and usage patterns involved in using and providing external dependencies, researchers can identify ways to improve the adoption of security processes, ultimately enhancing the security of the SSC.

Usable and Acceptable Authentication for Developers

Developers create our digital tools, but how do we know that the tools we have on our computer are actually created by developers we trust? This is a problem that has been well studied and can be solved with cryptographic signatures. With the most common form of digital signatures in open source projects being the venerable OpenPGP signature and Web of Trust, both for authenticating developers as well as the artifacts of their labor, some projects have sprung up thinking about how to solve this use case in a more user-friendly way without bringing in the possibility of impersonation by some (trusted) third party. One of those projects favored by the industry is Sigstore, with ideas from certificate infrastructures, while more security-minded software distributions have opted for simpler tools, like OpenBSD's `signify`.

This divide seems to be irreconcilable without investigating this area to find a universally acceptable solution to the authentication problem that includes all potential user concerns while being more user friendly than currently (at least partially) established practices. Having a universally accepted standard for authenticating developers and their work would heighten the level of security in the SSC against impersonation, and this can only be established by research that centers the users of this mechanism.

Metrics and Frameworks

In the context of a secure SSC, metrics and frameworks that classify security vulnerabilities (Common Vulnerabilities and Exposures, Common Vulnerability Scoring System, and Vulnerability Exploitability eXchange), weaknesses (Common Weakness Enumeration), or coding practices (OpenSSF Scorecards) play an important role in communicating among stakeholders. Adoption is a critical factor in the effectiveness of any metric or framework. If these tools become widely adopted, they create a network effect, whereby stakeholders become familiar with the metric and share a common understanding of what constitutes secure software development practices. As more stakeholders adopt and utilize a particular metric or framework, they build a collective understanding of what it takes to develop secure software (according to the metric), leading to a higher level of standardization and consistency in secure software development practices.

Designing these tools around the metrics that stakeholders actually care about and centering usability are key in making these tools effective and widely accepted. Without proper consideration of the human factor, these tools may not be utilized to their fullest potential or even adopted at all. Additionally, when more stakeholders utilize these tools, they can provide feedback on how to further improve their usability, resulting in

a continuous improvement cycle. By investigating and improving their usability, researchers can increase the likelihood that stakeholders will utilize these tools and ultimately establish a common understanding toward a more secure software ecosystem.

Open Source Versus Closed Source Conflicts

Large companies sometimes adapt open/libre source software (OSS) projects and may have internal changes to them that they maintain, update, develop, and never contribute back into the OSS space.[14] This may also be true for dealing with vulnerabilities, and it incurs costs for companies as well as the OSS ecosystem where the software originated.

However, there are legitimate reasons for forking and maintaining in a closed environment: oftentimes, the OSS community may not want to develop in the direction that a large company requires (e.g., Google, BoringSSL). A company may look at its plate first instead of the whole upstream bowl.

There are tradeoffs because of the work required for maintaining each internal fork a company may keep internally—each one has to be kept up to date with security patches, inventoried, and kept on watch for vulnerabilities and plain errors being fixed upstream. While having an internal cache of external dependencies brings benefits like keeping each dependency available even facing upstream disaster, they are certainly not for free and tend to break if left without care. That care may even be a full internal fork with in-house patches, but they too need to be maintained, making midterm costs skyrocket. Costs to interact with OSS—one-time setup, repetitiveness—may be different for each project.

A deeper understanding of better cooperation possibilities may be more economical, provide more prompt reaction to security incidents, and be in the best interest of all parties overall. Centering the needs, challenges, and decisions of stakeholders in human factors research can help improve cooperation in this space.

"One Guy in Kansas"

Some OSS is so ubiquitous in the development and operation of IT systems that its existence is hardly noticed. Its absence, or even just unfixed bugs, could wreak large costs and other damages for big organizations. Surprisingly, some of that software was developed or is maintained by very few developers, colloquially known as "that one guy in Kansas." This is specifically true for many open source projects, which are often done by default as hobbies, not contributing significantly to the income of their main developers.

Research into how to better support these small projects may have a significant impact on the security of the SSC. In conclusion, we need to consider human factors to secure the SSC, dependencies, and build systems. Recent attacks have demonstrated that developers are working on every link in the chain, making approaches that consider the human factor an important step for effective SSC security. ■

Acknowledgment

We would like to thank Henrik Plate and Laurie Williams for lively discussions and valuable feedback on this article. This work is funded in parts by NSF Grant CNS-2206865 and a Google Research Scholar Award.

References

1. "Securing the software supply chain: Recommended practices for developers," Cybersecurity and Infrastructure Security Agency, Washington, DC, USA, 2022. [Online]. Available: https://www.cisa.gov/sites/default/files/publications/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF
2. P. Ladisa, H. Plate, M. Martinez, and O. Barais, "SoK: Taxonomy of attacks on open-source software supply chains," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2023, pp. 1509–1526, doi: 10.1109/SP46215.2023.10179304.
3. Z. Durumeric et al., "The matter of heartbleed," in *Proc. Conf. Internet Meas. Conf.*, Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 475–488, doi: 10.1145/2663716.2663755.
4. D. Everson, L. Cheng, and Z. Zhang, "Log4shell: Redefining the web attack surface," in *Proc. Workshop Meas., Attacks, Defenses Web (MADWeb)*, 2022, pp. 1–8, doi: 10.14722/madweb.2022.23010.
5. "Highly evasive attacker leverages SolarWinds supply chain to compromise multiple global victims with SUNBURST backdoor." Mandiant. Accessed: Jan. 4, 2023. [Online]. Available: https://www.mandiant.com/resources/blog/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor
6. 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, Mar./Apr. 2022, doi: 10.1109/MSEC.2022.3142338.
7. "CircleCI incident report for January 4, 2023 security incident." CircleCI. Accessed: Jan. 4, 2023. [Online]. Available: https://circleci.com/blog/jan-4-2023-incident-report/
8. "Security incident update and recommended actions." LastPass. Accessed: Jan. 4, 2023. [Online]. Available: https://blog.lastpass.com/2023/03/security-incident-update-recommended-actions/
9. D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, "Committed to trust: A qualitative study

on security & trust in open source software projects," in *Proc. 43rd IEEE Symp. Secur. Privacy (S&P)*, May 2022, pp. 1880–1896, doi: 10.1109/SP46214.2022.9833686.
10. K. Thompson, "Reflections on trusting trust," *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984, doi: 10.1145/358198.358210.
11. M. Fourné, D. Wermke, W. Enck, S. Fahl, and Y. Acar, "It's like flossing your teeth: On the importance and challenges of reproducible builds for software supply chain security," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2023, pp. 1527–1544, doi: 10.1109/SP46215.2023.10179320.
12. Bootstrappable Builds. [Online]. Available: https://bootstrappable.org/
13. D. A. Wheeler, "Countering trusting trust through diverse double-compiling," in *Proc. 21st IEEE Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 2005, pp. 13–48, doi: 10.1109/CSAC.2005.17.
14. D. Wermke et al., "'Always contribute back': A qualitative study on security challenges of the open source supply chain," in *Proc. 44th IEEE Symp. Secur. Privacy (S&P)*, San Francisco, CA, USA: IEEE Comput. Soc., May 2023, pp. 1545–1560, doi: 10.1109/SP46215.2023.10179378.

Marcel Fourné (they/them) is a research engineer at the Max Planck Institute for Security and Privacy, 44799 Bochum, Germany. Their research focuses on cryptography, verification and (binary) program analysis, and programming languages, all through a lens of security as a part of correct program behavior. Fourné received an M.Sc. (and German Ingenieur) in IT security from Ruhr-University Bochum and is working toward a Dr. rer.nat. (Ph.D. equivalent) in computer science at the University of Paderborn. Recently, their focus shifted to security in open source development and its impacts on the computing ecosystem. They do this by analyzing the developers' work and their individual and shared environments, not only as producers but also as users of the things they build, maintain, and run. They sometimes solve problems in the Debian Haskell Group, maintain different software necessary for a full software supply chain, and publish software themselves. Their research won a distinguished paper award at the IEEE Symposium on Security and Privacy. They are a Student Member of IEEE. Contact them at email@marcelfourne.de and https://marcelfourne.de.

Dominik Wermke (he/him) is an assistant professor in the Department of Computer Science at North Carolina State University (NCSU), Raleigh, NC 27695-8206, USA. His research interests center on empowering software experts to design, develop, and deploy secure, privacy-respecting, and trustworthy software. His general areas of research are in usable security and privacy, software supply chain security and transparency, the open source ecosystem, and supporting software experts in designing secure and user-friendly systems. Wermke received his Dr. rer. nat. (Ph.D. equivalent) in computer science from Leibniz University Hannover. Prior to joining NCSU, he was a researcher at the CISPA Helmholtz Center for Information Security and part of the TeamUSEC research group for human-centered security. Contact him at dwermke@ncsu.edu and https://dwermke.com.

Sascha Fahl (he/him) is tenured faculty at the CISPA Helmholtz-Center for Information Security and professor of empirical information security at Leibniz University Hannover, 30159 Hannover, Germany. Previously, he was an associate professor at Ruhr University Bochum, Germany, chair of Information Security at Leibniz University Hannover, and an independent research group leader at CISPA, Saarland University. Fahl received a Ph.D. in computer science from Philipps University Marburg. He worked with the Chrome Security team and was a researcher at Fraunhofer FKIE in Bonn. His research won the Distinguished Paper award at the IEEE Symposium on Security and Privacy and the Best Student Paper award at the IEEE Symposium on Security and Privacy, and the National Security Agency's Best Scientific Cybersecurity Paper Competition. He received the Google Faculty Research Award. He is a recipient of the Heinz Maier-Leibnitz Prize and *Manager Magazin's* "Curious Mind" award. He is a Member of IEEE. Contact him at sascha.fahl@gmail.com and https://saschafahl.de.

Yasemin Acar (she/her) is a professor of computer science at Paderborn University, 33102 Paderborn, Germany, and a research assistant professor at the George Washington University. Her research focuses on human factors in computer security, centering on humans, their comprehension, behaviors, wishes, and needs. She aims to better understand how software can enhance users' lives without putting their data at risk. Her recent focus has been on human factors in secure development, investigating how to help software developers implement secure software development practices. Her research has shown that working with developers on these issues can resolve problems before they ever affect end users. Acar received a Dr. rer.nat. (Ph.D. equivalent) in computer science from Philipps University Marburg. Her research has won distinguished paper awards at the IEEE Symposium on Security and Privacy and USENIX Security, and a National Security Agency Best Cybersecurity Paper competition. She is a Member of IEEE. Contact her at acar@sec.uni-hannover.de and https://yaseminacar.de.