A Viewpoint on Human Factors in Software Supply Chain Security: A Research Agenda.

Marcel Fourné, Dominik Wermke, Sascha Fahl, and Yasemin Acar,

Abstract—Securing the software supply chain requires that we recognize the importance of individual developers. 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.[1]

Several steps to address dependency and build chain problems have been proposed—updating and using trusted dependencies, Software Bill of Materials (SBoMs), securing the build process, and more industry participation [5]. 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-backed SSO session, allowing the attacker to execute session cookie theft and impersonate the employee, gaining access to a subset of production systems.[2] 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 them access to the company's cloud storage and resulting in the theft of source code and customer vault data.[3] These incidents highlight that each individual 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 [6]. 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 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 under-investigated area of research. To address the need for more human-centered SCS approaches, we propose multiple aspects for a high-level research agenda below.

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

Digital Object Identifier 10.1109/XXX.0000.0000000

[1] https://www.mandiant.com/resources/blog/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor

[2] https://circleci.com/blog/jan-4-2023-incident-report/

[3] https://blog.lastpass.com/2023/03/security-incident-update-recommended-actions/

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 [7].

While we applaud [8] the work of the reproducible-builds.org project, this should only be the baseline and software build reproducibility - bootstrappability[4] and diversified checking à la David A. Wheelers "Diverse Double-Compiling" [9] 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 CI/CD.
Today's build systems and CI/CD pipelines can interact and chain with other systems and third-party services, allowing for the creation of complex, multi-step 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 these 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 typo-squatting and account-takeover attacks have shown. In response, Python's PyPI and GitHub have begun requiring two-factor authentication (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. E.g., 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 are necessary.

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

[4] https://bootstrappable.org/

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 SSC against impersonation, and 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 (CVE, CVSS, VEX), weaknesses (CWE), or coding practices (OpenSSF Scorecards) play an important role in communicating between 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 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 towards a more secure software ecosystem.

Open Source vs. 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 [10]. This may also be true for dealing with vulnerabilities and 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 trade-offs, due to 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 mid-term costs skyrocket. Costs to interact with OSS—one-time setup, repetitive—may be different on 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 open source software 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.

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

REFERENCES

[1] CISA, *Securing the software supply chain: Recommended practices for developers*, https : / /

www.cisa.gov/sites/default/files/publications/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF, 2022.

[2] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "Sok: Taxonomy of attacks on open-source software supply chains," in *2023 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2023, pp. 1509–1526.

[3] Z. Durumeric, F. Li, J. Kasten, *et al.*, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14, Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 475–488.

[4] D. Everson, L. Cheng, and Z. Zhang, "Log4shell: Redefining the web attack surface," in *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2022*, 2022.

[5] 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.

[6] 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 *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022*, May 2022.

[7] K. Thompson, "Reflections on Trusting Trust," *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984.

[8] 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 *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1527–1544.

[9] D. A. Wheeler, "Countering trusting trust through diverse double-compiling," in *21st Annual Computer Security Applications Conference (ACSAC'05)*, IEEE.

[10] D. Wermke, J. H. Klemmer, N. Wöhler, *et al.*, ""always contribute back": A qualitative study on security challenges of the open source supply chain," in *In Proceedings of the 44th IEEE Symposium on Security and Privacy (IEEE S&P '23)*, IEEE Computer Society, May 2023.