Increasing Trust in the Open Source Supply Chain with Reproducible Builds and Functional Package Management

Julien Malka

Increasing Trust in the Open Source Supply Chain with Reproducible Builds and Functional Package Management

Julien Malka
julien.malka@telecom-paris.fr
LTCI, Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France

ABSTRACT

Functional package managers (FPMs) and *reproducible builds (R-B)* are technologies and methodologies that are conceptually very different from the traditional software deployment model, and that have promising properties for software supply chain security. This thesis aims to evaluate the impact of FMPs and R-B on the security of the software supply chain and propose improvements to the FPM model to further improve trust in the open source supply chain.

I INTRODUCTION

A big promise of **Free and Open Source Software (FOSS)** is that it improves the control one have on the code running on their machine. Indeed, one of the strong perks of FOSS is its **auditability**: one can read through the source of a given software component and determine if its security is up to one's standards. Auditability of the source, however, is not enough to achieve **trust** in the software running on a machine. Indeed, to go from source code to an executable binary on the end-machine, code typically has to be *built* and then *distributed* through a package manager, a chain of processes and actors called the **software supply chain**. Once distributed, it is difficult to have any guarantee that the binary has actually been built with the audited code, or that it has not been tampered with during its compilation or distribution.

From a **traceability** perspective, this means that it is difficult to assess what a given system is actually made of: what are the software components it is depending on? Which versions of these dependencies are currently installed? These questions are becoming of greater importance as the European Union and the USA are pushing towards regulations to enforce increased traceability of software [1, 14], including the distribution of a **software bill of material (SBOM)**, a structured list of components needed to build and run a given application.

From a **security** perspective, all the steps taken from the source code to the executable binary correspond to an increase of the attack surface: it may be that the compiler used to create the executable files is malicious [24], or that the machine used to build the software is compromised. It may also be the case that one of the dependency of the software is malicious or vulnerable. The distribution phase also has potential for exposure: the package manager could for example download from untrusted or compromised sources [7]. In the last years, we have witnessed attacks of major impact targeting the software supply chain (see SolarWinds [6] or XCodeGhost [26] for example), raising the need for new methods to increase trustworthiness in the software supply chain.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04
https://doi.org/10.1145/3639478.3639806

II CONTEXT AND DEFINITIONS

Reproducible builds (R-B) is a term used to designate software components for which performing a compilation twice in the same environment results in the same (bit-by-bit identical) generated artifacts. Reproducible builds may be used to reduce the trust one puts in a binary distributor by having several parties reach consensus on the result of a compilation step. Build reproducibility is difficult to achieve in general because compiler outputs may be influenced by the current date and time, the state of the filesystem, random inputs or memory, etc. [16].

Functional package managers (FPMs) are implementations of a software deployment model first introduced by Dolstra [11]. In FPMs, packages are distributed as **pure functions** of their build- and run-time dependencies, and are described by expressions written in a domain-specific functional language. The functional software deployment model has two major implementations: Nix [11] and Guix [8]. While most software distributions are binary-based, **source-based distributions** like FPMs, allow for compilation of software to take place directly on the user's machine, helping to avoid a whole class of software supply chain vulnerabilities relative to distribution. FPMs builds are done in a hermetic and sandboxed environment containing only the build-time dependencies declared in the recipe, allowing for a *reproducible build environment*: any machine executing the same FPM build will perform the compilation in an environment containing exactly the same dependencies and toolchain. Reproducible build environments are a necessary condition to obtain reproducible builds, but not a sufficient one: if the build step itself in not deterministic, the build artifact will still differ between two builds. Because it is not always realistic to build packages on the user machine, FPMs also have a cache containing the build artifacts for the packages in their repository that users commonly rely on to avoid performing a lot of compilation on their local machine. Using binary caches invalidates some of the benefits of FPMs for the security of the software supply chain though; improving the security model of binary caches is one of the research questions of this PhD project.

III PROBLEM STATEMENT

The main objective of my PhD project is to **increase trust in the open source software supply chain using reproducible builds**

and functional package management. For this purpose, I will first evaluate where and how these methodologies and technologies can help, by comparing them to classical software deployment models and establishing which classes of supply chain issues they can eliminate. Then, based on that assessment, I will study how R-B and FPMs can be improved to increase the safety of the open source software supply chain. I'll also assess whether these improvements could benefit traditional (i.e., non FPM) package manager models.

IV RESEARCH QUESTIONS

As part of this thesis, I will study the following research questions:

RQ1: What are the comparative advantages of FPMs over classical package deployment, in terms of software supply chain failures?

- How do FPMs (using, or not, binary caches) compare to classical package managers against supply chain attack vectors documented in the literature [15, 19]? What are the features of FPMs that allow them to perform well against these attacks and could some of them be added to other package managers?
- Do FPMs increase build reproducibility? What proportion of packages in FPMs package repositories build reproducibly and how does that proportion evolve over time?
- Can we identify and classify the patterns that do not allow for a reproducible build, and are there general solutions to reach better determinism in the build systems used?

RQ2: How can the FPM model be leveraged to further improve the safety of the software supply chain?

- *Binary caches greatly weaken the trust model of FPMs, forcing users to trust the distributor of the binary.*
  What is the trust model for binary substitution using binary caches and how could it be improved, in particular with decentralized binary distribution models [5]? Is it possible to include traceable metadata in a build output that will increase the trustworthiness of the distributed binary?
- *Given that packages in Nix/Guix are pure functions from their inputs, it is possible to build a static graph of all the build-time and run-time dependencies of a given package.*
  Can we leverage the dependency graph created by FPMs to increase software traceability? How precise are SBOMs generated using these dependency graphs?

V EXPECTED CONTRIBUTIONS

The expected contributions of this thesis are the following:

(1) A formal analysis of the security benefits of the functional software deployment model including an analysis of the distincts features of FPMs that have the most important impact on software supply chain security;
(2) Propositions for improvements of the FPM model that have positive impact of the software supply chain security;
(3) Derived from (1), propositions of amelioration of classical package managers that can be cherry-picked from FPMs.

VI RESEARCH APPROACH

To address *RQ1*, I'll first devise a model of package managers and their features which I'll compare with a selection of mainstream package managers to validate that I've been able to capture their main caracteristics. With my model, I'll then perform an analysis of the classical software supply chain attack vectors identified in the literature [15, 19]. To analyse comparative performance on these attack vectors of FPMs against classical package managers, I'll assert how features of my model relate to the identified attack vectors and draw conclusions on the performance of each package manager against them. In order to evaluate whether FPMs properties allow them to have a large proportion of their packages building reproducibly, I'll perform empirical studies of the reproducibility over time of Nix evaluation and builds on the Nix package repository.

The first part of the PhD thesis will allow for identification of strengths and weaknesses of the FPM model when it comes to software supply chain security. In a second part, I will use that assessment to propose improvements to the FPM model to improve trust in the software supply chain security. *RQ2* identifies potential ideas and work areas that could be explored around the binary cache trust model or automatic SBOM generation, but outcomes from the work described earlier may point to other directions.

VII RELATED WORK

Most research on the topic of *software supply chain security* is focused on *a posteriori* analysis of software supply chain incidents and mitigation proposals [12, 20, 21, 23, 27]. Some metastudies [15, 19] have leveraged these works to give an overview and categorization of real-world incidents that were used to create a database of unique attack vectors and mitigating safeguards. I will use these datasets as the basis for the performance analysis of FPMs against classical attacks of the software supply chain (*RQ1*).

Substantial work is done by the *reproducible builds* group [4] to track down non-reproducibility issues in compilers and fix them, allowing for more reproducible packages in downstream software distributions. Interest has also been picked up by some software distributions, that increase monitoring for their packages reproducibility: for example, Debian now claims that over 95% of its 35 000 packages are reproducibly built [3], while NixOS claims that their minimal installation image is perfectly reproducible [2]. Yet, a study [13] shows that there is no consensus among experts on the reachability of the goal to make a mainstream software distribution fully reproducible.

Functional package managers have their origin in academic research on software deployment models [8, 11]. Since the original works describing the functional software deployment model, research in that area has mainly focused on the use of FPMs for reproducible science [22, 25] and more particularly applications in high performance computing [10], but recent work explores the security of the Guix software supply chain [9] and introduces a method for authenticating Guix commits when users perform an upgrade. Some non-academic proof of concepts have also been introduced in the Nix ecosystem around automatic SBOM generations [18] and binary cache distribution [5]. Additionally, functional package distributions have been the stage for experiments around full-source bootstrap [17], progressing towards a solution to the trusting trust problem [24].

REFERENCES

[1] 2022. Cyber Resilience Act | Shaping Europe's digital future. https://web.archive.org/web/20231109015038/https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act

[2] 2023. NixOS Reproducible Builds: minimal installation ISO successfully independently rebuilt - Announcements - NixOS Discourse. https://web.archive.org/web/20231030141336/https://discourse.nixos.org/t/nixos-reproducible-builds-minimal-installation-iso-successfully-independently-rebuilt/34756

[3] 2023. Overview of various statistics about reproducible builds. https://web.archive.org/web/20231029223006/https://tests.reproducible-builds.org/debian/reproducible.html

[4] 2023. Reproducible Builds — a set of software development practices that create an independently-verifiable path from source to binary code. https://web.archive.org/web/20231113151826/https://reproducible-builds.org/

[5] 2023. Trustix - A new model for Nix binary substitutions. https://github.com/nix-community/trustix original-date: 2020-12-03T15:30:40Z.

[6] Rahaf Alkhadra, Joud Abuzaid, Mariam AlShammari, and Nazeeruddin Mohammad. 2021. Solar Winds Hack: In-Depth Analysis and Countermeasures. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 1–7. https://doi.org/10.1109/ICCCNT51525.2021.9579611

[7] Justin Cappos, Justin Samuel, Scott M. Baker, and John H. Hartman. 2008. A look in the mirror: attacks on package managers. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, Peng Ning, Paul F. Syverson, and Somesh Jha (Eds.). ACM, 565–574. https://doi.org/10.1145/1455770.1455841

[8] Ludovic Courtès. 2013. Functional Package Management with Guix. https://doi.org/10.48550/arXiv.1305.4584 arXiv:1305.4584 [cs].

[9] Ludovic Courtès. 2022. Building a Secure Software Supply Chain with GNU Guix. *The Art, Science, and Engineering of Programming* 7, 1 (June 2022), 1. https://doi.org/10.22152/programming-journal.org/2023/7/1 arXiv:2206.14606 [cs].

[10] Ludovic Courtès and Ricardo Wurmus. 2015. Reproducible and User-Controlled Software Environments in HPC with Guix. https://inria.hal.science/hal-01161771

[11] Eelco Dolstra. 2006. *The purely functional software deployment model.* Ph. D. Dissertation. s.n., S.l. OCLC: 71702886.

[12] Gabriel Ferreira, Limin Jia, Joshua Sunshine, and Christian Kästner. 2021. Containing Malicious Package Updates in npm with a Lightweight Permission System. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1334–1346. https://doi.org/10.1109/ICSE43902.2021.00121 ISSN: 1558-1225.

[13] Marcel Fourné, Dominik Wermke, William Enck, Sascha Fahl, and Yasemin Acar. 2023. It's like flossing your teeth: On the Importance and Challenges of Reproducible Builds for Software Supply Chain Security. https://teamusec.de/publications/conf-oakland-fourne23/

[14] The White House. 2021. Executive Order on Improving the Nation's Cybersecurity. https://web.archive.org/web/20231114135442/https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[15] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. 2022. Taxonomy of Attacks on Open-Source Software Supply Chains. https://doi.org/10.48550/arXiv.2204.04008 arXiv:2204.04008 [cs].

[16] Chris Lamb and Stefano Zacchiroli. 2022. Reproducible Builds: Increasing the Integrity of Software Supply Chains. *IEEE Software* 39, 2 (March 2022), 62–70. https://doi.org/10.1109/MS.2021.3073045 Conference Name: IEEE Software.

[17] Janneke Nieuwenhuizen and Ludovic Courtès. 2023. The Full-Source Bootstrap: Building from source all the way down. https://web.archive.org/web/20231112105303/https://guix.gnu.org/en/blog/2023/the-full-source-bootstrap-building-from-source-all-the-way-down/

[18] nikstur. 2023. Bombon. https://github.com/nikstur/bombon original-date: 2022-08-18T23:06:53Z.

[19] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment (Lecture Notes in Computer Science)*, Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves (Eds.). Springer International Publishing, Cham, 23–43. https://doi.org/10.1007/978-3-030-52683-2_2

[20] Marc Ohm, Timo Pohl, and Felix Boes. 2023. You Can Run But You Can't Hide: Runtime Protection Against Malicious Package Updates For Node.js. https://doi.org/10.48550/arXiv.2305.19760 arXiv:2305.19760 [cs].

[21] Adriana Sejfia and Max Schäfer. 2022. Practical automated detection of malicious npm packages. In *Proceedings of the 44th International Conference on Software Engineering (ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1681–1692. https://doi.org/10.1145/3510003.3510104

[22] Francesco Strozzi, Roel Janssen, Ricardo Wurmus, Michael R. Crusoe, George Githinji, Paolo Di Tommaso, Dominique Belhachemi, Steffen Möller, Geert Smant, Joep de Ligt, and Pjotr Prins. 2019. Scalable Workflows and Reproducible Data Analysis for Genomics. In *Evolutionary Genomics: Statistical and Computational Methods*, Maria Anisimova (Ed.). Springer, New York, NY, 723–745. https://doi.org/10.1007/978-1-4939-9074-0_24

[23] Matthew Taylor, Ruturaj K. Vaidya, Drew Davidson, Lorenzo De Carli, and Vaibhav Rastogi. 2020. SpellBound: Defending Against Package Typosquatting. https://doi.org/10.48550/arXiv.2003.03471 arXiv:2003.03471 [cs].

[24] Ken Thompson. 1984. Reflections on Trusting Trust. *Commun. ACM* 27, 8 (1984), 761–763. https://doi.org/10.1145/358198.358210

[25] Ricardo Wurmus, Bora Uyar, Brendan Osberg, Vedran Franke, Alexander Gosdschan, Katarzyna Wreczycka, Jonathan Ronen, and Altuna Akalin. 2018. PiGx: reproducible genomics analysis pipelines with GNU Guix. *GigaScience* 7, 12 (Dec. 2018), giy123. https://doi.org/10.1093/gigascience/giy123

[26] Claud Xiao. 2015. Novel Malware XcodeGhost Modifies Xcode, Infects Apple iOS Apps and Hits App Store. https://web.archive.org/web/20230920153656/https://unit42.paloaltonetworks.com/novel-malware-xcodeghost-modifies-xcode-infects-apple-ios-apps-and-hits-app-store/

[27] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. 995–1010. https://www.usenix.org/conference/usenixsecurity19/presentation/zimmerman