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

Julien Malka

To cite this version:

Julien Malka. Increasing Trust in the Open Source Supply Chain with Reproducible Builds and Functional Package Management. 46th International Conference on Software Engineering (ICSE 2024) - Doctoral Symposium (DS) Track, Apr 2024, Lisbonne, Portugal. 10.1145/3639478.3639806 . hal-04482192

HAL Id: hal-04482192
https://hal.science/hal-04482192
Submitted on 28 Feb 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 FPMs 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 audibility: 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.

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 is 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.
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?

IV RESEARCH QUESTIONS

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

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 distinct 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 characteristics. 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.

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].