S. Wendzel et al. (Hrsg.): SICHERHEIT 2024,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2024 243

Reproducible Builds and Insights from an Independent Verifier for Arch Linux

Joshua Drexel[1] Esther Hänggi[2] Iyán Méndez Veiga[3]

Abstract: Supply chain attacks have emerged as a prominent cybersecurity threat in recent years. Reproducible and bootstrappable builds have the potential to reduce such attacks significantly. In combination with independent, exhaustive and periodic source code audits, these measures can effectively eradicate compromises in the building process. In this paper we introduce both concepts, we analyze the achievements over the last ten years and explain the remaining challenges. We contribute to the reproducible builds effort by setting up a rebuilder and verifier instance to test the reproducibility of Arch Linux packages. Using the results from this instance, we uncover an unnoticed and security-relevant packaging issue affecting 16 packages related to Certbot, the recommended software to install TLS certificates from Let's Encrypt, making them unreproducible. Additionally, we find the root cause of unreproduciblity in the source code of `fwupd`, a critical software used to update device firmware on Linux devices, and submit an upstream patch to fix it.

Keywords: Reproducible Builds; Supply Chain Security; FOSS; Arch Linux

1 Introduction

One of the main advantages of Free[4] [GNUa] and Open Source [Ope06] Software (FOSS) over proprietary software is the possibility for third parties to independently audit the source code in order to find bugs and backdoors. Programs whose source code is publicly available are not necessarily more secure than closed source alternatives. Indeed, some of the worst cyber incidents in recent years have been linked to bugs affecting open source software [The14]; [The21a]. On the other hand, open source projects often attain higher software quality when continuously reviewed by independent developers. This scrutiny not only leads to early detection of bugs and security issues but also fosters a culture of proactive improvement [Ray99]. However, even if the source code is secure, this is not sufficient: Most users and companies using FOSS software do not download the source code and compile it themselves. Instead, they download a binary, a pre-compiled version of the software that can be executed right away. Proving that a certain binary was created from some determined and unmodified source code is a hard task.

A solution to this problem needs to uniquely link source code and binary code, and this is the goal of reproducible builds (R-B) [LZ22]: Providing an independently-verifiable path

[1] School of Computer Science and Information Technology, Hochschule Luzern (HSLU), 6343 Rotkreuz, Switzerland
[2] School of Computer Science and Information Technology, Hochschule Luzern (HSLU), 6343 Rotkreuz, Switzerland
[3] School of Computer Science and Information Technology, Hochschule Luzern (HSLU), 6343 Rotkreuz, Switzerland & Institute for Theoretical Physics, ETH Zurich, 8093 Zurich, Switzerland, iyan.mendezveiga@hslu.ch
[4] Sometimes called "libre" to emphasize that "free" should be understood as in freedom, not as *gratis*.

doi: 10.18420/sicherheit2024_016

from source to binary code. Anyone with access to the source code and detailed building instructions, including dependencies, should be able to transform the thousands or millions of lines of source code into a binary artifact that is bit-by-bit identical every single time the build process is executed. The immediate security benefits of R-B are enormous. For example, independent verifiers can test that distributed binaries have been created from the published and audited source code. Additionally, attacks on the build infrastructure [The11]; [Fre12] can be detected. R-B also bring some benefits beyond security: Identical artifacts lead to higher cache hit ratios, lowering development costs due to more efficient compilations. In the future, R-B could be used in other tasks such as intrusion detection, intelligence gathering using build honeypots, or even become a prerequisite to certify critical software.

R-B still requires a reliable and trusted build toolchain. Even with independent builders and verifiers, if all of them are running the same compromised compiler, the final artifact can include malicious code while being completely reproducible [Skr18]. Obtaining or generating a trusted build toolchain is a hard problem as well. The reason is that it is challenging to start a build process solely from source code. Normally, a specific version of a compiler is built using a pre-compiled binary of another version of the compiler. This leads to a chicken-and-egg problem for trust in the compiler, since we always have to trust at least some initial binary. In the last years, techniques have been developed to increase the trust in pre-compiled compilers such as Diverse Double-Compiling (DDC) [Dav05]; [Dav10]. Bootstrapping, on the other hand, aims to provide ways to generate a working and trusted build toolchain using binary artifacts smaller than a compiler or, ideally, only auditable code. Bootstrappable builds (B-B) is an effort to eliminate the compiler trust loophole in R-B by minimizing, or completely eliminating, the amount of bootstrap binaries required in a build process [The16].

In this paper, we will describe what reproducible builds are and the progress which has been made in this topic over the last ten years. We will then report on our contributions to the R-B effort. We have set up a rebuilder and verifier instance to test the reproducibility of Arch Linux official packages. Through this, we have identified and reported a packaging issue affecting 16 packages related to Certbot. Fixing these packages to be reproducible is relevant since this software is the recommended tool to install TLS certificates issued by Let's Encrypt. We have also submitted an upstream patch to fix the root cause of unreproducibility in the source code of `fwupd`, enabling distributions to provide reproducible packages. The effectiveness of R-B in preventing supply chain attacks heavily relies on the awareness of the developers about this topic. Increasing visibility of R-B is therefore an additional goal of this paper.

2 Concepts & Definitions

Source code Source code *is the version of software as it is originally written (i.e., typed into a computer) by a human in plain text (i.e., human readable alphanumeric characters)* [The06]. In other words, the source code of a software is the set of all files that contain human-readable instructions written in a programming language that can be understood by programmers.

Build toolchain The build toolchain is the set of tools required to transform the source code of a specific programming language into machine code that can be executed by a central processing unit. Depending on the type of the programming language this may include a compiler, an assembler, a linker, an interpreter, etc.

Artifacts We define artifacts as any output of a build toolchain, with the exception of plaintext logs. This may include machine code executables, libraries, documentation, tarballs, packages, filesystem images, etc.

Reproducible builds (R-B) The build process of a software is reproducible, sometimes also denoted as deterministic or verifiable, *if, after designating a specific version of its source code and all of its build dependencies, every build produces bit-by-bit identical artifacts, no matter the environment in which the build is performed* [LZ22]. A set of independent verifiers can be used to test whether a software is reproducible or not, see Fig. 1.

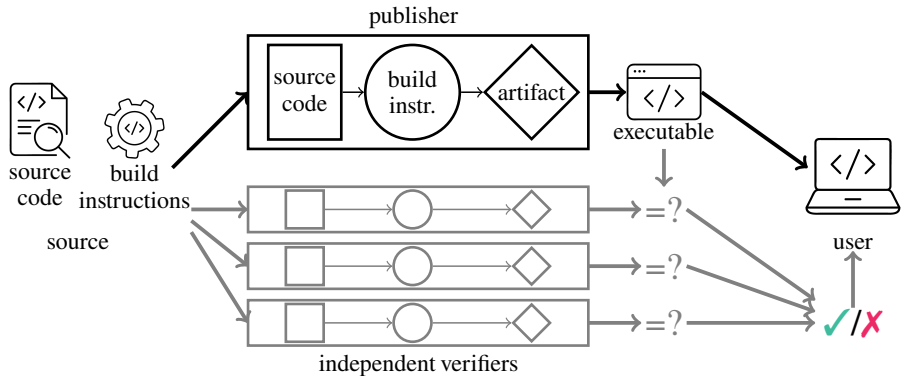

Fig. 1: The publisher of an open source software provides an executable ready to use. Most users would trust and run the executable based on the publisher's reputation. More cautious users may opt to review the source code. However, there is no guarantee that the provided executable comes directly from the published source code. R-B allows testing whether a provided artifact originates from a given source code. A set of independent verifiers can fetch the source code and rebuild the software following the exact same build instructions, as well as using the same dependencies. They publish whether they obtained the same artifacts and the user can now use this information to decide whether or not to trust the distributed executable. This diagram is a simplified version of Fig. 1 from [LZ22].

In this definition, the build dependencies include the details about the build toolchain. The hard requirement of bit-by-bit identical artifacts may look excessive until one realizes that a single bit is what could separate a safe software from an insecure one (see, e.g., [The02]). Note that R-B is not about making software more secure, insecure software will remain insecure.

Bootstrappable builds (B-B) Most compilers are written in the same programming language they are compiling. For example, modern versions of `GNU GCC` are written in `C++` [LWN13]. The Java Development Kit (JDK) is written in Java, so to build JDK a pre-compiled version of JDK is required. The goal of bootstrappable builds is to end these loops where untrusted compilers have to be used to obtain newer versions of the same compiler by providing clear bootstrapping paths, using compilers written in other languages, and minimizing or eliminating the amount of required binary artifacts to start a build process.

Diverse Double-Compiling R-B verifiers are vulnerable to certain compiler attacks, such as "trusting trust" attacks[5]. Diverse Double-Compiling (DDC) is a technique to detect such attacks. It needs three things: The compiler to be tested, its source code, and a second "trusted" compiler. The source code of the compiler is then compiled twice: First with the "trusted" compiler, and a second time with the resulted compiler of the first compilation. If the final result exactly matches the original compiler, the compiler has not been tampered with [Dav05]; [Dav10].

Software Bill of Materials (SBOM) An SBOM is a nested inventory, a list of all the various components used in building a software, including FOSS and proprietary software, third-party libraries and other dependencies. The SBOM is intended to offer transparency and traceability in the software supply chain, and to help securing it [Cyb23].

3 Origin of reproducible builds

Together with Dennis Ritchie, Ken Thompson won the 1983 Turing Award *"for their development of generic operating systems theory and specifically for the implementation of the UNIX operating system"*. During the award lecture [Tho84] he explained how easy it would be to modify compilers to insert malware in critical software, including the compiler itself. He widely publicized the problem of "trusting trust" attacks that had only been briefly explored by Paul A. Karger and Roger R. Schell before [KS74]. He already pointed out two of the main problems R-B tries to solve: The issue of trusting binary code whose generation is not fully controlled, and the impossibility of securely running untrusted binary artifacts even after a source-level verification. This lecture seeded the interest in reproducible and bootstrappable builds years later.

In the 90s, the company Cygnus provided commercial support for free software. They made the GNU tools they were supporting fully reproducible, including `gcc`, `gdb`, `binutils` and `make`. Interestingly, this effort was only revealed 25 years later in the Reproducible Builds Project email list [rb-17], and remains the first publicly known case of R-B. From the point of view of Cygnus, binary files that did not match bit-by-bit when generated from the same source code were considered a bad software development practice. During the following

[5] In a "trusting trust" attack a malicious compiler produces corrupted executables, including corrupted versions of the same compiler from its source code, making the situation self-perpetuating.

years of source code changes without aiming for R-B, some unreproducibility issues which had been fixed in the 90s, were reintroduced and then had to be fixed again.

Explicit interest in R-B as a desired goal was expressed in the Debian email list in the early 2000s [deb00]; [deb07]. However, no specific actions to make Debian packages reproducible were taken. It was even a common believe that such task was not feasible. What finally sparked interest in R-B was money and privacy concerns from Bitcoin and Tor users. Bitcoin Core was made reproducible in 2011 to ensure that the distributed executable was not modified to steal Bitcoin wallets [Bit11]. Inspired by this work and motivated by the fear of targeted attacks against their users, the Tor Browser started to offer reproducible builds [The13] two years later.

Both Bitcoin and Tor projects showed that achieving R-B was feasible. This motivated Debian developer Jérémy Bobbio (lunar) to organize a public discussion about R-B during DebConf13, the annual Debian Project's developer conference. The session identified three of the most common causes of unreproduciblity while building source code (timestamps, build paths and locales), as well as problems with the Debian toolchain [Deb13]. It was decided to create a web page to keep track of all efforts to make Debian packages reproducible.

Although the origin of R-B is tightly connected to Debian, it is now a widely discussed topic in other communities, and in the software industry in general [Goo23]; [Mic22]. This manifests in a multitude of talks, workshops, and other events focused on R-B (see, e.g., [The23a]; [Cas23]; [Lev23]). Recently, R-B gained further attention after the US government issued an Executive Order on *Improving the Nation's Cybersecurity* [The21b], where SBOM and R-B are explicitly mentioned as a solution to supply chain attacks.

Concerning the toolchain used to ensure reproducibility, early successful attempts by Bitcoin and Tor used sanitized and isolated environments, such as minimal well-defined virtual machines, to remove any uncontrolled inputs to the build system. This is a valid approach but leads to slow building times, excessive technical restrictions on the toolchain used by developers, and it cannot fix unreproducibility caused by non-determinism. A completely different approach is to modify the source code and build tools to ensure that only the relevant inputs (source code, dependencies and build toolchain), and nothing else, can produce changes on the final artifacts. For practical reasons, the modern toolchain to produce and test R-B lies in between these two approaches (see, e.g., [Arc17]).

4 Current state

Many projects have adopted reproducible builds in the last few years. Some, such as Bitcoin Core, Tor Browser, or the WireGuard Android app [Jas23], even test that their new releases are reproducible before making them publicly available.

Some of the largest GNU/Linux distributions, such as Debian, Fedora, OpenSUSE or Arch Linux, have made a noteworthy effort to provide reproducible packages in their official repositories, although none of them enforce their packages to be reproducible at the moment.

The Reproducible Builds Project [LLM] itself was born with the goal of making Debian 100% reproducible. As of November 2023, Debian leads the effort among GNU/Linux distributions to become reproducible with 95.2% of their `Bookworm/amd64` packages being reproducible [Rep23a]. 75.8% of all Arch Linux packages are reproducible [Rep23b]. In addition, Arch Linux has developed an independent verification system for binary packages, `rebuilderd` [Arc23a], which allows independent verifiers to test the reproducibility of Arch's and even Debian's repositories. In the future, `rebuilderd` will also support Alpine Linux.

Security and privacy oriented distributions, Qube OS and Tails, have recently joined the effort. Tails installation images, for example, are fully reproducible [The23b]. Other notable OS such as FreeBSD, NetBSD, Alpine Linux or OpenWrt, have plans to become fully reproducible.

The efforts to achieve R-B have led to thousands of patches involving hundreds of FOSS projects. For example, the GNU GCC compiler introduced additional options to support embedding relative paths in the final artifacts instead of absolute paths. Many projects were modified to support the `SOURCE_DATE_EPOCH` specification [Rep15], and improve their build automation files (e.g., `Makefile`) to remove non-deterministic behaviours.

Actively testing for reproducibility has helped improving the overall quality of software by making the build setup more robust under all kind of environments, as well as uncovering issues that would have otherwise remained unnoticed, sometimes even with security implications (see, e.g., [Deb16]). Fixes and improvements on specific FOSS projects, as well as on the build toolchain, benefit all users independently of the operating system they use.

Nowadays, the most common causes of unreproducibility are well understood, and suggested patches or workarounds to fix them are documented (see, e.g., [Rep14a]; [Deb23]). Some of the most frequent ones are mentioned in Table 1.

Category	Count (%)	Identified & documented issues
Timestamps	127 (30%)	C++ macros, Gzip headers, man pages, PDFs generated with LaTeX, docs generated by Doxygen, Maven version files, . . .
Randomness	71 (17%)	random order in tarball files, random hashes in Cython, random order of static libraries, randomness in fat lto objects, . . .
Paths	68 (16%)	build path captured by gcc, build path captured by Rust, absolute paths in CMake files generated by Meson, . . .

Tab. 1: Summary table containing some of the well-known causes of unreproducibility, from a total of 422 documented and categorized issues in [Rep14a], as of November 2023. For actual source code examples also check "The Unreproducible Package" repository [Wie17]. This classification also applies to other GNU/Linux distributions, although some of the technical solutions or recommended practices may differ.

Tools are available to assist with the root cause analysis of unreproducible artifacts. The most widely used is `diffoscope` [Rep14b]. It transforms multiple binary formats into more human-readable forms and presents the results as a `diff` text or html, allowing a better comparison. Embedded times and paths are highlighted, while it can also help debugging more complex non-deterministic issues such as embedded randomness, uninitialized memory

or address space layout randomization. Despite some recent progress in automating these tasks [Ren+22], achieving R-B remains a manual and highly time-consuming process.

5 Challenges

Awareness Despite the efforts of both the Reproducible Builds Project [LLM] and the Bootstrappable Build Project [The16], these topics still remain unknown to many developers. Raising awareness about software supply chain attacks and the benefits of R-B and B-B will improve the situation and further accelerate the goal of being able to run a fully reproducible operating system. Aware programmers are less likely to introduce new sources of non-deterministic behaviour in their programs that could cause regressions in the reproducibility status.

Policy changes Even though most GNU/Linux distributions working on R-B have achieved success ratios above 90% in their official repositories, some of the remaining unreproducible packages will be challenging to fix since they involve multiple changes upstream. Unmaintained unreproducible software will never be fixed, and a small fraction of the remaining software will not be reproducible by decision of their developers [Gho]. Achieving a 100% reproducible ratio for GNU/Linux distributions with large repositories, such as Debian, will only be possible together with a clear decision of not packaging unreproducible software. Such decision will be controversial, and may lead to new forks like it happened in the past [Dev].

Archive infrastructure Continuously testing for package reproducibility involves downloading old versions of dependencies that may no longer exist in the stable repositories. Debian and Arch Linux maintain their own archives servers [Deba]; [Arca] that allow such testing. Debian Snapshot, for example, includes all packages since 2005. Arch Linux Archive only maintains packages for a few years[6], but older packages are moved to the Internet Archive [The] before being deleted. Maintaining this kind of infrastructure requires time and effort, and comes with a huge cost that smaller projects cannot afford.

Independent rebuilders R-B are only useful with independent verifiers continously rebuilding the software, allowing users to query whether a distributed software is reproducible or not (see Fig. 1). This is lacking at the moment as only a few testing platforms exist, most of them maintained by the Reproducible Builds project. Ideally, in the future, easy to setup rebuilders will allow users and companies to run their own reproducible tests and report their results. In addition, protocols have to be developed to allow users to quickly decide if a binary is to be trusted.

[6] As of November 2023, the oldest Linux kernel available in the Arch Linux Archive is version 4.20.10 packaged in February 2019.

6 Our contribution to the reproducible builds effort

6.1 Hochschule Luzern verifier for Arch Linux reproducibility

One of the key open issues to fully benefit from R-B is the lack of independent verifiers that continuously test packages, or other artifacts, to check if they are reproducible (see Sec. 5). Only with enough verifiers confirming reproducibility, we can trust that the distributed artifacts originate from a given source code.

We have set up a server running `rebuilderd` [kpc] to test the reproducibility of Arch Linux packages in the official repositories. The reason we have chosen to use this distribution is twofold: Arch Linux provides packages with minimal distribution-specific changes with respect to upstream versions, and the package build description files are `bash` scripts, easy to read and understand even for people without prior packaging experience.

The verifier works in the following way: A sync daemon checks and downloads the latest packages from the official Arch Linux repositories every 5 minutes. The `rebuilderd` daemon splits the work among the active workers, that can be dynamically deployed based on the workload, and exposes an API to query the status of the packages: GOOD are reproducible, BAD failed to build or are not reproducible, and UNKWN have not been checked yet. A website available at `https://reproducible.crypto-lab.ch` shows the status of the packages, build logs, `diffoscope` logs for unreproducible builds and attestation logs for reproducible packages. A simplified diagram of our setup, as well as the specs of the hardware used can be seen in Fig. 2. A list of all independent `rebuilderd` instances can be found in the Arch Linux wiki [Arcb].

Packages marked as reproducible are not tested again until a new version becomes available in the official repositories. Unreproducible packages, however, are requeued endlessly by `rebuilderd` after some delay. This delay is determined by the number of attempts times a configurable base amount. We increased the value of the base delay from 24 hours to 1 week in order to avoid delaying the test of new versions of packages over unreproducible ones.

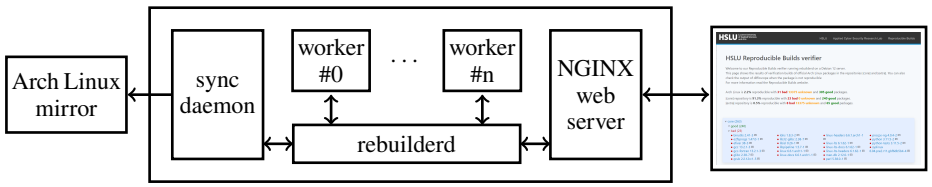

Fig. 2: The `rebuilderd` daemon requires few resources and can be run on a small cloud instance (e.g., 1 vCPU and 1 GB of RAM). Our independent verifier (sync daemon and a small worker) runs on a modest refurbish PC with an Intel i7-4790K CPU and 16 GB of RAM. Workers need at least 32 GB of RAM to be able to build all the official packages. A significant speedup on large builds (e.g. linux kernel, web browsers, etc.) can be achieved with at least 64 GB of RAM and mounting the worker's directory as `tmpfs`. For the initial setup, two VMs with 16 vCPU, 64 GB of RAM and 200 GB of SSD storage were used as workers.

6.2 Fixing unreproducible Arch Linux packages

While exploring some of the unreproducible packages detected by our instance, we uncovered and reported a packaging bug in how all the `certbot` packages are created in Arch Linux [Arcc]. Certbot [Ele] is the official tool to obtain TLS certificates from Let's Encrypt [Int]. It can be considered critical software since it is currently used by millions of users to manage certificates and malicious versions could potentially leak the private keys. All certbot-related packages are considered reproducible in Debian's testing infrastructure [Debb], so there are no technical reasons why they should not be reproducible in Arch Linux as well. The underlying problem affecting the 16 packages was the workflow followed to update the package to a new version.

Arch Linux uses a `bash`-alike script called `PKGBUILD` [Arc23b] to describe the instructions to create a package from source code, and stores an SBOM in a key-value formattted file called `.BUILDINFO` [Arc23c] inside the generated package. Dependencies required to run, build and test a package are specified in the `PKGBUILD` variables `depends`, `makedepends`, and `checkdepends`, respectively. They are all fetched and installed in the building machine before any function is executed. A minimal `PKGBUILD` must contain at least the `package()` function, which is used to install the relevant files into the packaging directory. Optionally, the functions `prepare()`, `pkgver()`, `build()` and `check()` can also be used. Relevant to understand the issue we uncovered is the `pkgver()` function, which is used to update the `pkgver` variable where the version of the package is stored. In this particular case, a fixed and manually defined git commit is used by this function, which determines the associated tag version and uses it to update `pkgver`. The issue was that `pkgver` had been previously used to determine the version of one of the dependencies, `python-acme`. This is accurately recorded in the SBOM of the package, causing a mismatch between the required version by the `PKGBUILD` script and the actual version used to produce the official package.

To avoid similar problems in the future, Arch Linux build toolchain could be patched to abort in situations where a function can modify the building description (`PKGBUILD`) after the dependencies have been fetched and installed.

6.3 Fixing unreproducible source code

Fixing unreproducible packages in a specific distribution is a useful contribution to the R-B efforts. Even better is fixing the causes of unreproducibility directly in the source code of the upstream projects, since this has the potential to fix unreproducible packages for all distributions working towards that goal. Using the insights from our rebuilder instance, we have further discovered an issue in the building scripts of `fwupd` [Hug15]. Our motivation to make `fwupd` reproducible is that it is closely related to securing software security chains, as it is the standard software to update firmware in Linux devices.

In this case, the very well-known issue of an embedded timestamp in the header of a compressed file (see Table 1) was causing the unreproducibility. The reason why this was

not caught earlier is probably because the file was generated using a Python script and `gzip` from the Python Standard Library [Pyt23] instead of the standard GNU tool `gzip` [GNUb]. We submitted an upstream patch to fix the issue, which was quickly accepted [fwu]. This will make the package in Arch Linux reproducible [Arcd], as well as in any other distribution packaging the latest release of `fwupd` once it becomes available.

7 Conclusions

In recent years, a clear trend has emerged aimed at enhancing the security of software supply chains. Some countries are actively advocating, and in some cases mandating, the implementation of software bill of materials (SBOM). Reproducible builds (R-B) is the only solution to validate these SBOMs through independent verification. Bootstrappable builds (B-B) serve as a definitive protective measure against "trusting trust" attacks during the build process. The combination of both B-B and R-B ensures a clear and unequivocal linkage between source code and binary code. When complemented with routine source code audits, these measures effectively mitigate the risk of supply chain attacks.

Despite the great number of achievements of R-B during the last ten years, there are still many open challenges to achieve the goal of running 100% reproducible operating systems. First and foremost, an independent network of rebuilders and verifiers is needed. Our instance of an independent verifier supports this cause. In addition, all the software running these verifiers should be itself reproducible, which is still an open issue. Second, many specific issues in the build process and upstream source code still need to be fixed to make all packages reproducible. We have reported on both types of issues, uncovering a packaging issue affecting the Arch Linux build process, and fixing directly the source code of a crucial software to make it reproducible.

We finally hope with this paper to encourage others to do the same and join the reproducible builds effort.

8 Acknowledgments

We thank the core team of the Reproducible Builds Project, and the Arch Linux developers and package maintainers active in the #archlinux-reproducible IRC channel for valuable insights, especially kpcyrd, the author of `rebuilderd`, and Jelle van der Waa, the maintainer of the official Arch Linux reproducible status website.

This work was supported by SNSF Practice-to-Science grant no. 199084.

References

[GNUa] GNU Operating System. *What is Free Software?* Accessed: November 14, 2023. URL: https://www.gnu.org/philosophy/free-sw.html.

[Ope06] Open Source Initative. *The Open Source Definition*. Accessed: November 14, 2023. 2006. URL: https://opensource.org/osd/.

[The14] The MITRE Corporation. *CVE-2014-0160: OpenSSL Heartbeat Extension Vulnerability*. Accessed: November 9, 2023. 2014.

[The21a] The MITRE Corporation. *CVE-2021-44228: Apache Log4j Remote Code Execution Vulnerability*. Accessed: November 9, 2023. 2021.

[Ray99] Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 1999, p. 30. ISBN: 1-56592-724-9.

[LZ22] Chris Lamb and Stefano Zacchiroli. "Reproducible Builds: Increasing the Integrity of Software Supply Chains". In: *IEEE Software* 39.2 (2022), pp. 62–70. URL: https://doi.org/10.1109/MS.2021.3073045.

[The11] The Register. *Kernel.org Linux repository rooted in hack attack*. Accessed: November 17, 2023. 2011. URL: https://www.theregister.com/2011/08/31/linux_kernel_security_breach/.

[Fre12] FreeBSD. *Security Incident on FreeBSD Infrastructure*. Accessed: November 17, 2023. 2012. URL: https://www.freebsd.org/news/2012-compromise/.

[Skr18] Yrjan Skrimstad. "Improving Trust in Software through Diverse Double-Compiling and Reproducible Builds". MA thesis. University of Oslo, 2018. URL: https://www.duo.uio.no/handle/10852/65737.

[Dav05] David A. Wheeler. "Countering trusting trust through diverse double-compiling". In: *21st Annual Computer Security Applications Conference (ACSAC'05)*. 2005, 13 pp.–48. URL: https://doi.org/10.1109/CSAC.2005.17.

[Dav10] David A. Wheeler. "Fully Countering Trusting Trust through Diverse Double-Compiling". PhD thesis. George Mason University, 2010. URL: https://doi.org/10.48550/arXiv.1004.5534.

[The16] The Bootstrappable Builds Project. *Bootstrappable Builds*. Accessed: November 14, 2023. 2016. URL: https://bootstrappable.org/.

[The06] The Linux Information Project. *Source Code Definition*. Accessed: November 13, 2023. 2006. URL: https://www.linfo.org/source_code.html.

[The02] The MITRE Corporation. *CVE-2002-0083: OpenSSH 2.x/3.0.1/3.0.2 - Channel Code Off-by-One*. Accessed: November 14, 2023. 2002. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0083.

[LWN13] LWN. *GCC's move to C++*. Accessed: November 17, 2023. 2013. URL: https://lwn.net/Articles/542457/.

[Cyb23] Cybersecurity and Infrastructure Security Agency (CISA). *Securing the Software Supply Chain: Recommended Practices for Software Bill of Materials Consumption*. 2023. URL: https://www.cisa.gov/news-events/alerts/2023/11/09/cisa-nsa-and-partners-release-new-guidance-securing-software-supply-chain.

[Tho84] Ken Thompson. "Reflections on Trusting Trust". In: *Commun. ACM* 27.8 (1984), pp. 761–763. ISSN: 0001-0782. DOI: 10.1145/358198.358210. URL: https://doi.org/10.1145/358198.358210.

[KS74] Paul A. Karger and Roger R. Schell. *Multics Security Evaluation: Vulnerability Analysis*. Tech. rep. ESD-TR-74-193, Vol. II. Electronic Systems Division, Hanscom AFB, 1974. URL: http://csrc.nist.gov/publications/history/karg74.pdf.

[rb-17] rb-general. *SOURCE_PREFIX_MAP and Occam's Razor*. Accessed: November 14, 2023. 2017. URL: https://lists.reproducible-builds.org/pipermail/rb-general/2017-January/000309.html.

[deb00] debian-devel email list. *Autobuilding and embedded timestamps*. Accessed: November 14, 2023. 2000. URL: https://lists.debian.org/debian-devel/2000/11/msg01758.html.

[deb07] debian-devel email list. *Re: Building packages three times in a row*. Accessed: November 14, 2023. 2007. URL: https://lists.debian.org/debian-devel/2007/09/msg00746.html.

[Bit11] Bitcoin. *Bitcoin: Gitian building*. Accessed: November 17, 2023. 2011. URL: https://github.com/bitcoin/bitcoin/blob/v0.10.0/doc/gitian-building.md.

[The13] The Tor Blog. *Deterministic Builds Part One: Cyberwar and Global Compromise*. Accessed: November 14, 2023. 2013. URL: https://blog.torproject.org/deterministic-builds-part-one-cyberwar-and-global-compromise/.

[Deb13] DebConf13. *Meeting Minutes: BoF on reproducible builds*. Accessed: November 14, 2023. 2013. URL: https://wiki.debian.org/ReproducibleBuilds/History?action=AttachFile&do=view&target=dc13-bof-reproducible-builds.txt.

[Goo23] Google Cloud. *Software supply chain security*. Accessed: November 17, 2023. 2023. URL: https://cloud.google.com/software-supply-chain-security/docs/overview.

[Mic22] Microsoft. *Best practices for a secure software supply chain*. Accessed: November 17, 2023. 2022. URL: https://learn.microsoft.com/en-us/nuget/concepts/security-best-practices.

[The23a] The Reproducible Builds Projects. *Events*. Accessed: November 17, 2023. 2023. URL: https://reproducible-builds.org/events/.

[Cas23] Vagrant Cascadian. *Breaking the Chains of Trusting Trust: Reproducible Builds and More!* Accessed: November 17, 2023. 2023. URL: https://2023.fossy.us/schedule/presentation/118/.

[Lev23] Holger Levsen. *Reproducible Builds, the first ten years*. Accessed: November 17, 2023. 2023. URL: https://bornhack.dk/bornhack-2023/program/reproducible-builds-the-first-ten-years/.

[The21b] The White House. *Executive Order on Improving the Nation's Cybersecurity*. Accessed: November 10, 2023. 2021. URL: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/.

[Arc17] Archlinux. *repro: Tools to reproduce Arch Linux packages*. Accessed: November 17, 2023. 2017. URL: https://github.com/archlinux/archlinux-repro.

[Jas23] Jason A. Donenfeld. *Android Reproducible Builds & Signing Key Changes*. Accessed: November 17, 2023. 2023. URL: https://lists.zx2c4.com/pipermail/wireguard/2023-April/008045.html.

[LLM] Chris Lamb, Holger Levsen, and Vagrant Cascadian Mattia Rizzolo. *The Reproducible Builds Project*. Accessed: November 10, 2023. URL: https://reproducible-builds.org.

[Rep23a] Reproducible Builds. *Debian Continuous Integration Tests*. Accessed: November 13, 2023. 2023. URL: https://tests.reproducible-builds.org/debian/bookworm/index_suite_amd64_stats.html.

[Rep23b] Reproducible Builds. *Arch Linux Continuous Integration Tests*. Accessed: November 13, 2023. 2023. URL: https://tests.reproducible-builds.org/archlinux/archlinux.html.

[Arc23a] Arch Linux. *Arch Linux Reproducible status*. Accessed: November 13, 2023. 2023. URL: https://reproducible.archlinux.org/.

[The23b] The Tails Project. *Verifying a Tails image for reproducibility*. Accessed: November 13, 2023. 2023. URL: https://tails.net/contribute/build/reproducible/.

[Rep15] Reproducible Builds Project. *SOURCE_DATE_EPOCH specification*. Accessed: November 17, 2023. 2015. URL: https://reproducible-builds.org/specs/source-date-epoch/.

[Deb16] Debian. *Bug report: gbrowse: ships a deterministic/predictable OpenID consumer secret*. Accessed: November 17, 2023. 2016. URL: https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=833885.

[Rep14a] Reproducible Builds. *Notes on build reproducibility of Debian packages*. Accessed: November 17, 2023. 2014. URL: https://salsa.debian.org/reproducible-builds/reproducible-notes.

[Deb23] Debian. *Known issues related to reproducible builds*. Accessed: November 13, 2023. 2023. URL: https://tests.reproducible-builds.org/debian/index_issues.html.

[Wie17] Bernhard M. Wiedemann. *The Unreproducible Package*. Accessed: November 17, 2023. 2017. URL: https://github.com/bmwiedemann/theunreproduciblepackage.

[Rep14b] Reproducible Builds Project. *diffoscope: In-depth comparison of files, archives, and directories*. Accessed: November 17, 2023. 2014. URL: https://diffoscope.org/.

[Ren+22] Zhilei Ren et al. "Automated Patching for Unreproducible Builds". In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 200–211. ISBN: 9781450392211. URL: https://doi.org/10.1145/3510003.3510102.

[Gho] GhostScript Bugzilla. *Support SOURCE_DATE_EPOCH for reproducible builds*. Accessed: November 15, 2023. URL: https://bugs.ghostscript.com/show_bug.cgi?id=696765.

[Dev] Devuan. *Forking Debian*. Accessed: November 16, 2023. URL: https://www.devuan.org/os/announce/.

[Deba] Debian. *The Snapshot Archive*. Accessed: November 16, 2023. URL: https://snapshot.debian.org/.

[Arca] Arch Linux. *The Arch Linux Archive*. Accessed: November 16, 2023. URL: https://archive.archlinux.org/.

[The] The Internet Archive. *The Internet Archive*. Accessed: November 16, 2023. URL: https://archive.org.

[kpc] kpcyrd. *rebuilderd: Independent verification of binary packages - reproducible builds*. Accessed: November 16, 2023. URL: https://github.com/kpcyrd/rebuilderd.

[Arcb] Arch Linux Wiki. *Rebuilderd: Package rebuilders*. Accessed: November 16, 2023. URL: https://wiki.archlinux.org/title/Rebuilderd#Package_rebuilders.

[Arcc] Arch Linux Bugtracker. *[certbot] Issue with .BUILDINFO in recent versions*. Accessed: November 16, 2023. URL: https://bugs.archlinux.org/task/80266.

[Ele] Electronic Frontier Foundation. *Certbot*. Accessed: November 16, 2023. URL: https://certbot.eff.org/.

[Int] Internet Security Research Group. *Let's Encrypt*. Accessed: November 16, 2023. URL: https://letsencrypt.org/.

[Debb] Debian. *Reproducible Builds of Debian: python-certbot*. Accessed: November 16, 2023. URL: https://tests.reproducible-builds.org/debian/rb-pkg/bookworm/amd64/python-certbot.html.

[Arc23b] Arch Linux. *PKGBUILD(5)*. Accessed: November 17, 2023. 2023. URL: https://man.archlinux.org/man/PKGBUILD.5.en.

[Arc23c] Arch Linux. *BUILDINFO(5)*. Accessed: November 17, 2023. 2023. URL: https://man.archlinux.org/man/BUILDINFO.5.

[Hug15] Richard Hughes. *fwupd: A system daemon to allow session software to update firmware*. Accessed: November 17, 2023. 2015. URL: https://github.com/fwupd/fwupd/.

[Pyt23] Python Software Foundation. *gzip — Support for gzip files*. Accessed: November 17, 2023. 2023. URL: https://docs.python.org/3/library/gzip.html.

[GNUb] GNU Operating System. *GNU Gzip*. Accessed: November 17, 2023. URL: https://www.gnu.org/software/gzip/.

[fwu] fwupd. *GitHub PR: Remove timestamp from gzip to achieve reproducible build*. Accessed: November 16, 2023. URL: https://github.com/fwupd/fwupd/pull/6388.

[Arcd] Arch Linux Bugtracker. *[fwupd] Remove timestamp to make package reproducible*. Accessed: November 16, 2023. URL: https://bugs.archlinux.org/task/80271.