Building Blocks for a Safer Open Source Supply Chain

Reproducible Builds and Software Heritage

Stefano Zacchiroli

Software Heritage
Télécom Paris, Polytechnic Institute of Paris

21 April 2023
KTH Royal Institute of Technology
Stockholm, Sweden

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Outline

About the speaker

- Professor of Computer Science, Télécom Paris, Polytechnic Institute of Paris
- Free/Open Source Software activist (20+ years)
- Debian Developer & Former 3x Debian Project Leader
- Former Open Source Initiative (OSI) director
- Software Heritage co-founder & CTO
- Reproducible Builds board member

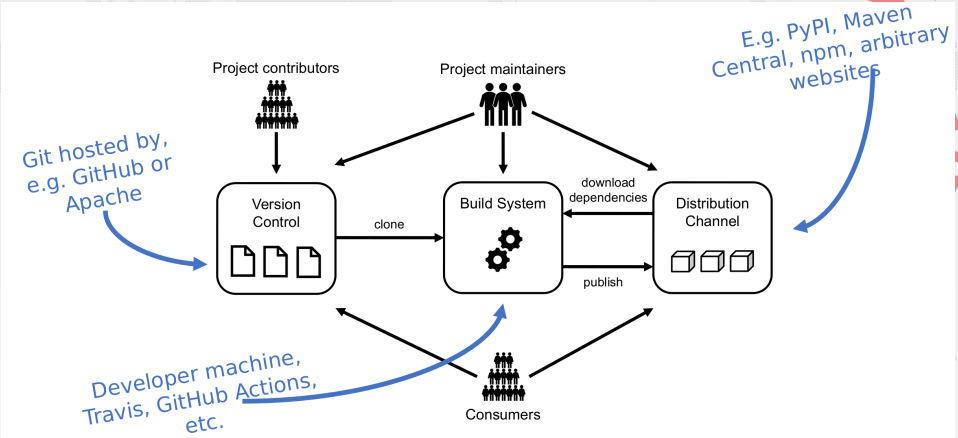
Outline

The software supply chain

- Supply chain: the set of activities required by an organization to deliver goods or services to consumers.
- Software supply chain: the set of software components and software services required to deliver an IT product or service to users.
  - libraries, runtimes, and other software component dependencies
  - base system (operating system, package manager, compiler, …)
  - development tools and platform (e.g., IDEs, build system, GitHub/GitLab, CI/CD, …)
  - etc.

Key artifact for audits: SBOM = Software Bill of Materials

(An) open source development workflow



Supply chain attacks

A software supply chain attack is a particular kind of cyber-attack that aims at injecting malicious code into an otherwise legitimate software product.

Notable examples

- NotPetya (2017): ransomware concelaed in an update of a popular accounting software, hitting Ukranian banks and major corps (B$)
- CCleaner (2017): malicious version of a popular MS Windows maintenance tool, distributed via the vendor website
- SolarWinds (2020): malicious update of the SolarWinds Orion monitoring software, shipping a delayed-activation trojan. Breached into several US Gov. branches as well as Microsoft

Open source supply chain attacks

- Is this specific to Free/Open Source Software (FOSS)? No.
- But modern FOSS package ecosystems are heavily intertwined.
  - Examples: NPM (JavaScript), PyPI (Python), Crates (Rust), Gems (Ruby), etc.
  - 100/10k/1M packages, depending on each other due to code reuse opportunities.
  - Reverse transitive dependencies grow fast. A single package could be required by thousands of others.

left-pad (2016)

(Not an attack, but gives an idea of how entangled package ecosystems could be.)

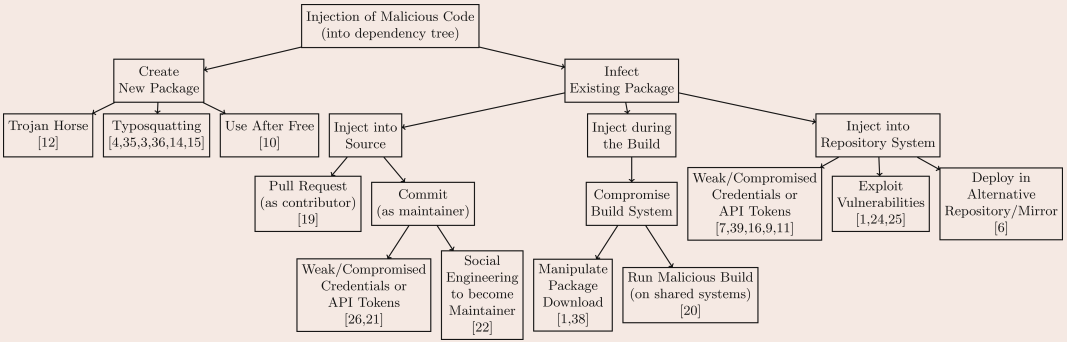
```
function leftpad (str, len, ch) {
    str = String(str);
    var i = -1;
    if (!ch && ch !== 0) ch = ' ';
    len = len - str.length;
    while (++i < len) { str = ch + str; }
    return str;
}
```

- Maintainer: *"I think I have the right of deleting all my stuff"*. "Unpublish" package.
- Impact: "many thousands of projects"—including major ones like babel and atom—no longer installable.
- NPM operators forcibly "un-unpublish" package.

Open source supply chain attacks (cont.)

- For an attacker, code injection into (transitively) popular leaf packages has a low opportunity cost.
- Also, entirely open FOSS package ecosystems (!= Linux distros) can be easy to infiltrate.

Attack tree — Injection



(image from [Ohm20])

Attacker's goal: package P containing malicious code is available from download from a distribution platform and P is a reverse transitive dependency of a legitimate package.

Attack vector — Compromise build system

Injection of Malicious Code → Infect Existing Package → Inject during the Build → Compromise Build System

- Often, code run by users is written but not built by maintainers
- Rather, it is built by 3rd-party vendors
  - e.g., GNU/Linux distros, app store operators, arch "porters"
- It hence becomes attractive to break into vendor build systems, compromising binaries "downstream", without anybody auditing source code noticing

Related attack vectors: Inject into [Package] Repository System (!= VCS)

Outline

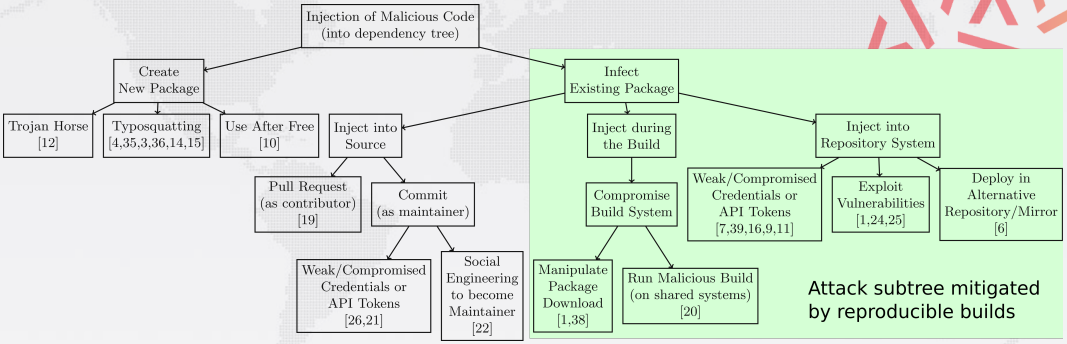
On untrusted code

> *You can't trust code that you did not totally create yourself. [...] No amount of source-level verification or scrutiny will protect you from using untrusted code.*
> *— Ken Thompson, Reflections on Trusting Trust, Turing Lecture 1984*

- 40 years later nobody "totally creates" code they run
- Reuse of open source software (FOSS) is everywhere in IT
  - "99% of audited code bases contain FOSS components" (Synopsis, 2020)
- Also, the FOSS we run is often not built by its developers

Problem statement

How can we increase users' trust when running (trusted) FOSS code built by (untrusted) 3rd-party vendors?



A reproducible build (r-b) process

Precondition/hypothesis: we can "reproducibly build" all relevant (FOSS) products, i.e.:

> *The build process of a software product is reproducible if, after designating a specific version of its source code and all of its build dependencies, every build produces bit-for-bit identical artifacts, no matter the environment in which the build is performed. — [Lamb22]*

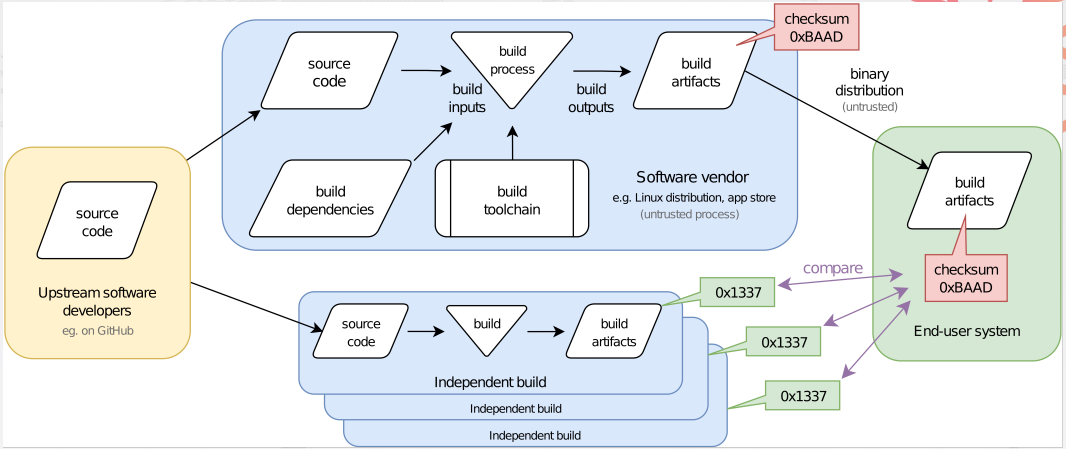
A reproducible build (r-b) process

Precondition/hypothesis: we can "reproducibly build" all relevant (FOSS) products, i.e.:

> *The build process of a software product is reproducible if, after designating a specific version of its source code and all of its build dependencies, every build produces bit-for-bit identical artifacts, no matter the environment in which the build is performed. — [Lamb22]*

(we'll verify later how realistic this is)

R-B approach


source code
build process
build inputs
build outputs
build artifacts
checksum 0xBAAD
binary distribution
(untrusted)
build dependencies
build toolchain
Software vendor
e.g. Linux distribution, app store
(untrusted process)
source code
Upstream software developers
eg. on GitHub
build artifacts
checksum 0xBAAD
End-user system
compare
source code
build
build artifacts
0x1337
0x1337
0x1337
Independent build
Independent build
Independent build

Making Debian reproducible

Experiment

Let's try a large-scale experiment: making all Debian packages build reproducibly from source

- Debian: one of the largest and most popular GNU/Linux distro, esp. in the server/cloud market
- 30'000+ (source) packages, 1+B lines of code
- Initial goal of the reproducible-builds.org initiative, est. 2014

Goals

1. Empirical experiment to identify common causes of non-reproducibility
2. Real impact (if successful) due to Debian popularity in the market

Build reproducibility in the small

How hard could it be to ensure build reproducibility?

Build reproducibility in the small

How hard could it be to ensure build reproducibility?

After controlling for source code, build deps., and toolchain, two main classes of issues arise in practice:

1. Uncontrolled build inputs: when toolchains allow the build process to be affected by the surrounding environment.
   - Intuition: this is the build engineering equivalent of breaking encapsulation in programming
2. Build non-determinism that gets encoded in final built artifacts.

Build reproducibility in the small

How hard could it be to ensure build reproducibility?

After controlling for source code, build deps., and toolchain, two main classes of issues arise in practice:

1. Uncontrolled build inputs: when toolchains allow the build process to be affected by the surrounding environment.
   - Intuition: this is the build engineering equivalent of breaking encapsulation in programming
2. Build non-determinism that gets encoded in final built artifacts.

Let's see some real-world examples…

Build paths

```
fprintf (stderr,
    "DEBUG: boop (%s:%s\n",
    __FILE__, __LINE__);
```

- The `__FILE__` C preprocessor macro "expands to the name of the current input file". This results in non reproducibility when the program is built from different directories, e.g., `/home/lamby/tmp` vs. `/home/zack/tmp`.
- Fix: introducted gcc `-ffile-prefix-map` option (and related `-fdebug-prefix-map`) to support embedding relative (rather than absolute) paths

Filesystem ordering

```
NAME
    readdir - read a directory

SYNOPSIS
    #include <dirent.h>
    struct dirent *readdir(DIR *dirp);

[...] The order in which filenames are read by successive
calls to readdir() depends on the filesystem implementation;
it is unlikely that the names will be sorted in any fashion.
[...]
```

- Fix: impose a deterministic order in build systems/recipes, e.g., via an explicit `sort()`

Build reproducibility in the large

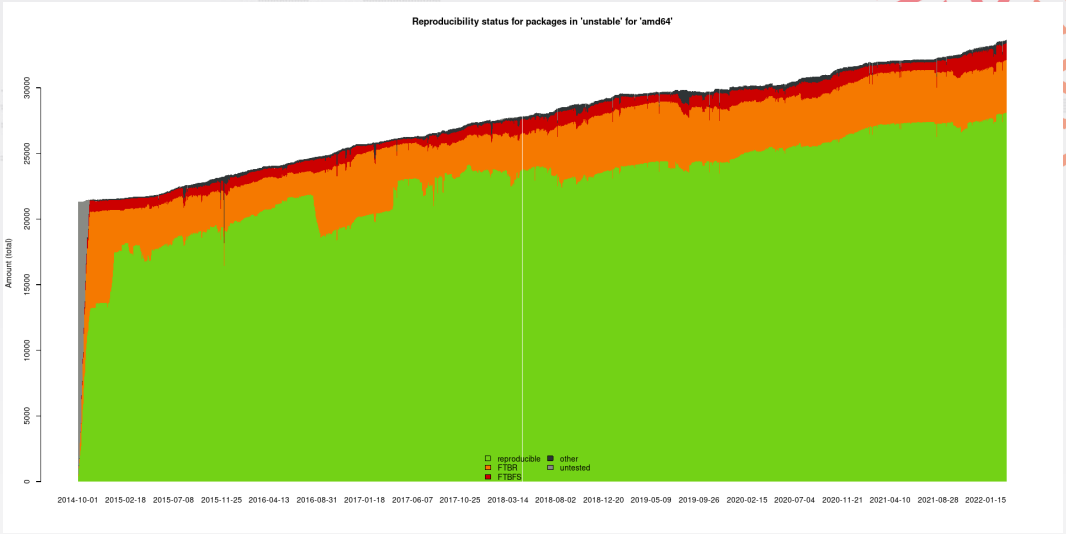
- Let's now assume we know how to fix all micro-issues that affact build reproducibility.
- How do we go about making large FOSS software collections reproducible?
  - Use case: Debian
- Approach: establish a corresponding Quality Assurance process and soft-enforce it using Continuous Integration (CI).

Adversarial rebuilding

How do you find build reproducibility issues, at scale?

- Mass-rebuild all packages…
- …building each of them twice…
- …in two build environments configured to differ as much as possible:
  - Clock set 18 months in the future in 2nd build
  - Changing: hostname, locales, kernel
  - Reverse filesystem ordering using disorderfs
  - 30+ variations in total

Reproducible Debian — Evolution over time



Reproducible Builds

https://reproducible-builds.org/

- 2014: project started by Debian developers for ~~Debian needs~~ fun
- Joined since: Arch Linux, coreboot, F-Droid, Fedora, FreeBSD, Guix, NixOS, openSUSE, Qubes, Tails, …
- 2017 milestone: Tails (live distro used by Snowden to exfiltrate NSA documents) publishes a fully reproducible ISO to improve end-user verifiability
- R-B is an independent project hosted by Software Freedom Conservancy and supported by 3rd-party sponsors (e.g., Google, The Linux Foundation, Ford Foundation, Siemens)

Challenges

- Debian reached 95% reproducible packages, can we go all the way?
  - Yes, it's just busy/constant maintenance work.
  - Working with upstream and spreading r-b culture helps a lot.
- How to make signed buld artifacts reproducible (without distributing signing keys)?
  - Detached signatures. (Painful for distribution channels.)
- How do end-user verify build artifacts before installation?
  - Particularly challenging on locked-down mobile environments/stores.
- How little trusted code is acceptable?
  - Bootstrappable Builds managed to bootstrap from a 6 KiB trusted ELF binary to GCC via TCC.

Outline

Open Source is growing…

Software is eating the world

THE WALL STREET JOURNAL.

Home World U.S. Politics Economy Business Tech Markets Opinion Arts

ESSAY

Why Software Is Eating The World

By Marc Andreessen

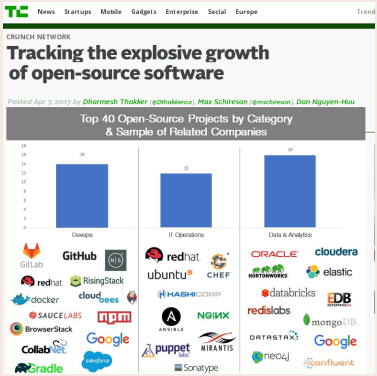
August 20, 2011

This week, Hewlett-Packard (where I am on the board) announced that it is exploring jettisoning its struggling PC business in favor of investing more heavily in software, where it sees better potential for growth. Meanwhile, Google plans to buy up the cellphone handset maker Motorola Mobility. Both moves surprised the tech world. But both moves are also in line with a trend I've observed, one that makes me optimistic about the future

Software companies outperform
or buy out traditional companies

Marc Andreesen, 2011

Open Source is eating the Software World



TC News Startups Mobile Gadgets Enterprise Social Europe Trend

CRUNCH NETWORK

Tracking the explosive growth of open-source software

Posted Apr 7, 2017 by Dharmesh Thakker (@dthakkerca), Max Schireson (@mschireson), Dan Nguyen-Huu

Top 40 Open-Source Projects by Category & Sample of Related Companies

Reuse is the new rule

80% to 90% of a new application is … just reuse! (Sonatype survey, 2017)

Where does reused software come from?

Do *you* know where it comes from?

- the software you ship
- the software you use
- the software you acquire
- the software that
  - has that bug
  - has that vulnerability

KYSW: Know Your SoftWare

Like KYC in banking, KYSW is now essential all over IT…

Sec. 4. Enhancing Software Supply Chain Security

ensuring and attesting, to the extent practicable, to the integrity and provenance of open source software

May 2021 POTUS Executive Order

A long road ahead

Vertical approach

improve security of *each component* separately

Horizontal approach

explore *the whole supply chain*

A few key challenging properties

findability	needs qualified metadata
availability	needs an archive and a system of identifiers
integrity	needs crypto
traceability	needs a global provenance database
reproducibility	needs groundbreaking tools

A long road ahead

Vertical approach

improve security of *each component* separately

Horizontal approach

explore *the whole supply chain*

A few key challenging properties

findability	needs qualified metadata
availability	needs an archive and a system of identifiers
integrity	needs crypto
traceability	needs a global provenance database
reproducibility	needs groundbreaking tools

We need a *global coordinated effort…*
and a *common, open, shared* infrastructure to track *all (Open Source) software*!

Outline

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Collect, preserve and share *all* software source code

Preserving our heritage, enabling better software and better science for all

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Collect, preserve and share *all* software source code

Preserving our heritage, enabling better software and better science for all

Reference catalog

find and reference all software source code

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Collect, preserve and share *all* software source code

Preserving our heritage, enabling better software and better science for all

Reference catalog

find and reference all software source code

Universal archive

preserve and share all software source code

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Collect, preserve and share *all* software source code

Preserving our heritage, enabling better software and better science for all

Reference catalog

find and reference all software source code

Universal archive

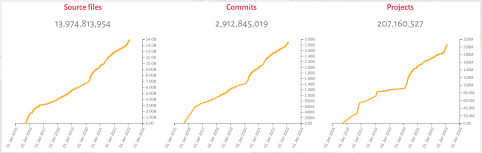
preserve and share all software source code

Research infrastructure

enable analysis of all software source code

One infrastructure
open and shared

One infrastructure
open and shared



Largest archive

One infrastructure
open and shared

Largest archive

Technology

- transparency and FOSS
- replicas all the way down

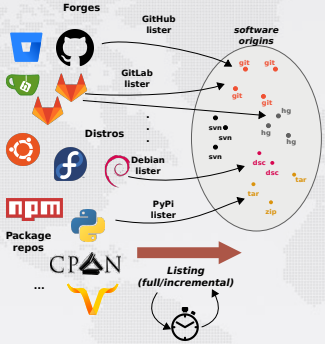
Content (billions!)

- intrinsic identifiers
- facts and provenance

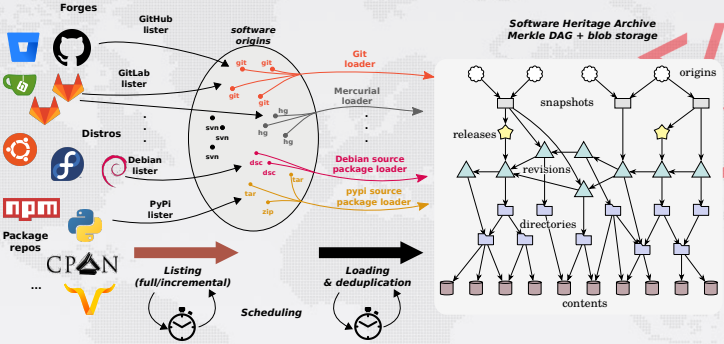
Organization

- non-profit
- multi-stakeholder

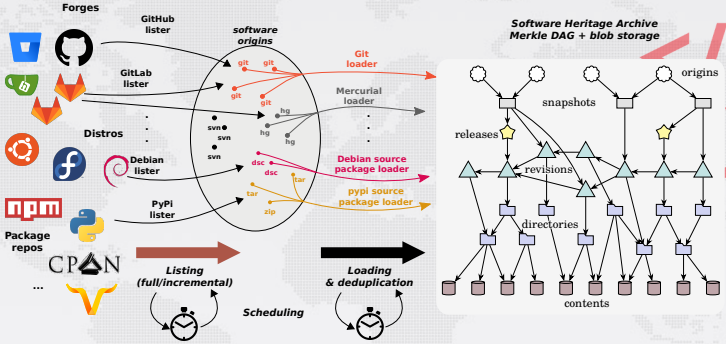
A peek under the hood: a universal archive



A peek under the hood: a universal archive



A peek under the hood: a universal archive



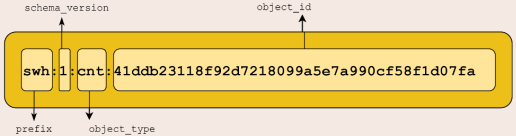
Global development history permanently archived in a uniform data model

- over 14 billion unique source files from over 210 million software projects
- ~1PB (compressed) blobs, ~30 B nodes, ~400 B edges

Software Heritage Identifiers (SWHID)

see swhid.org

25+B intrinsic, decentralised, cryptographically strong identifiers, SWHIDs

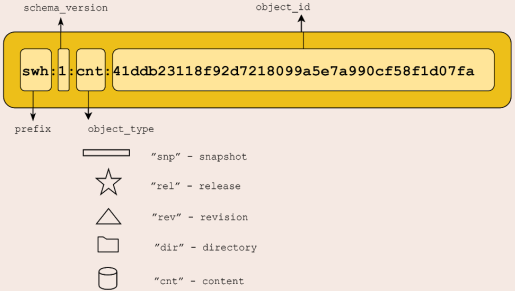


Intrinsic Identifiers for software artefacts

Software Heritage Identifiers (SWHID)

see swhid.org

25+B intrinsic, decentralised, cryptographically strong identifiers, SWHIDs



Software Heritage Identifiers (SWHID)

see swhid.org

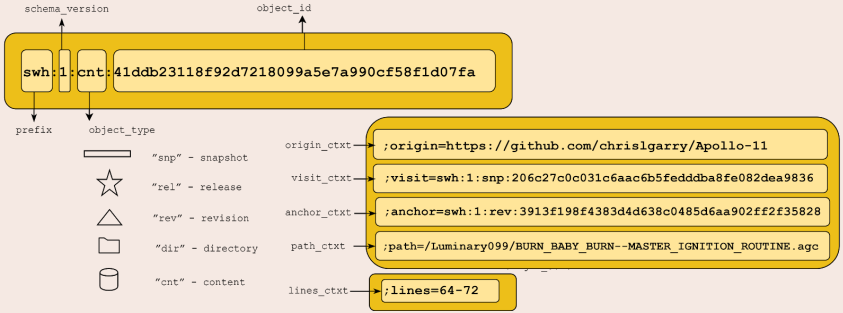
25+B intrinsic, decentralised, cryptographically strong identifiers, SWHIDs

Intrinsic Identifiers for software artefacts

Software Heritage Identifiers (SWHID)

see swhid.org

25+B intrinsic, decentralised, cryptographically strong identifiers, SWHIDs



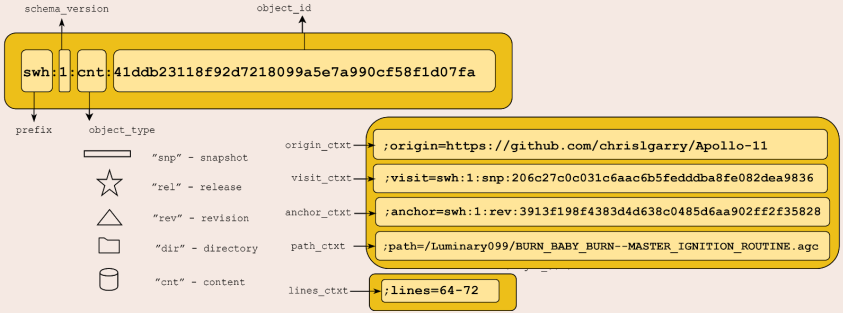
Emerging standard : Linux Foundation SPDX 2.2; IANA registered; WikiData P6138

Intrinsic Identifiers for software artefacts

Software Heritage Identifiers (SWHID)

see swhid.org

25+B intrinsic, decentralised, cryptographically strong identifiers, SWHIDs



Emerging standard : Linux Foundation SPDX 2.2; IANA registered; WikiData P6138

Full fledged *source code references* for reproducibility

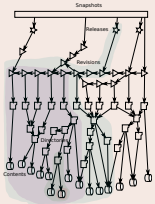
Examples: Apollo 11 AGC excerpt, Quake III rsqrt; Guidelines available, see ICMS 2020

A quick tour

- Browse (e.g. Apollo 11, and your work may be already there !)
- Trigger archival, use the updateswh browser extension, configure the webhooks
- Get and use SWHIDs (full specification available online)
- Cite software with biblatex-software package from CTAN
  - Overleaf ACMART template available
- Example in journals: article from IPOL
- Example with Parmap: devel on Github, archive in SWH, curated deposit in HAL
- Extracting all the software products for Inria, for CNRS, for CNES, for LIRMM or for Rémi Gribonval using HalTools
- Curated deposit in SWH via HAL, see for example: LinBox, SLALOM, Givaro, NS2DDV, SumGra, Coq proof, ...
- Example use in research articles:
  - compare Fig. 1 and conclusions in the 2012 version and the updated version
  - SWHID in a replication experiment

A revolutionary infrastructure for industry

The *graph* of public software development



All of the software development in a single graph!

- lookup by content hash
- wayback machine for software development
  - http://archive.softwareheritage.org/
- … and much more

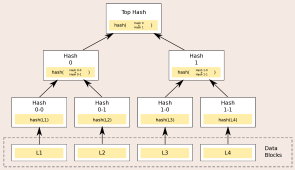
A revolutionary infrastructure for industry

The *graph* of public software development

All of the software development in a single graph!

- lookup by content hash
- wayback machine for software development
  - http://archive.softwareheritage.org/
- … and much more

The *global ledger* of public code

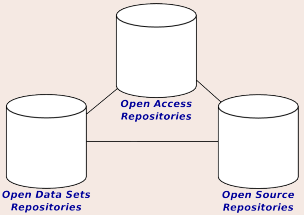


All of a software development… in a single Merkle graph!
Widely used crypto (e.g., Git, blockchains, IPFS, …)

- built-in deduplication
- intrinsic, unforgeable identifiers at all levels
- simplifies traceability (licensing, supply chain management)

A revolutionary infrastructure for research and innovation

A *pillar* of Open Science

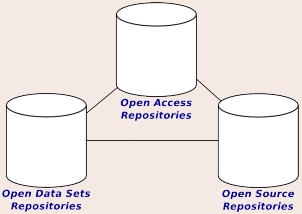


The *reference archive* of Research Software for Open Science

- curated deposit of research software
  - in collaboration with HAL, CCSD and Inria IES
  - now open *to all researchers*!
- intrinsic identifiers for reproducibility

A revolutionary infrastructure for research and innovation

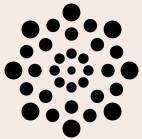
A *pillar* of Open Science



The *reference archive* of Research Software for Open Science

- curated deposit of research software
  - in collaboration with HAL, CCSD and Inria IES
  - now open *to all researchers*!
- intrinsic identifiers for reproducibility

Reference platform for *Big Code*



- unique observatory of all software development
- big data, machine learning paradise: classification, trends, coding patterns, code completion…

Industry use cases (selection)

Open Source complete and corresponding source code distribution (Intel)

Software Heritage members can:

- archive source code in Software Heritage, distribute only the SWHID

Industry use cases (selection)

Open Source complete and corresponding source code distribution (Intel)

Software Heritage members can:

- archive source code in Software Heritage, distribute only the SWHID

Traceability and integrity (OIN for the *Linux System Definition*)

Software Heritage members can:

- archive source code in Software Heritage
- track it and verify its integrity using its SWHID

Industry use cases (selection)

Open Source complete and corresponding source code distribution (Intel)

Software Heritage members can:

- archive source code in Software Heritage, distribute only the SWHID

Traceability and integrity (OIN for the *Linux System Definition*)

Software Heritage members can:

- archive source code in Software Heritage
- track it and verify its integrity using its SWHID

And much more!

- an open source, open data source code scanner for open compliance (`swh-scanner`)
- security (large project with French Government)
- supply chain management, long term archive *add your use case here*

swh-scanner

Vision

`swh-scanner` is an open source and open data source code scanner for open compliance workflows, backed by the largest public archive of FOSS source code.

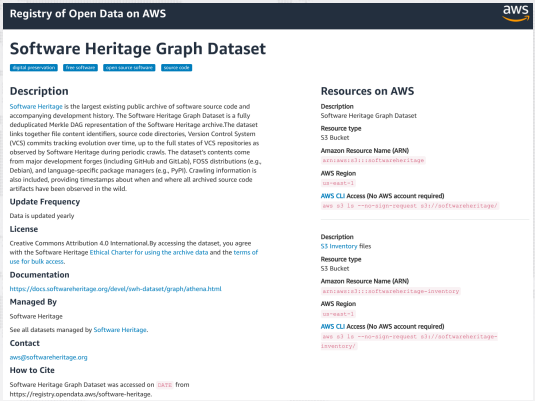
Design

- Query Software Heritage as source of truth about public code
- Leverages the Merkle DAG model and SWHIDs for maximum scanning efficiency
  - E.g., no need to query the back-end for files contained in a known directory
- File-level granularity
- Output: source tree partition into known (= published before) v. unknown

Source: gitlab.softwareheritage.org/swh/devel/swh-scanner
License: GPL-3+
Package: pypi.org/project/swh.scanner

The Software Heritage archive as an open dataset

https://registry.opendata.aws/software-heritage/



Registry of Open Data on AWS

aws

Software Heritage Graph Dataset

digital preservation · free software · open source software · source code

Description

Software Heritage is the largest existing public archive of software source code and accompanying development history. The Software Heritage Graph Dataset is a fully deduplicated Merkle DAG representation of the Software Heritage archive.The dataset links together file content identifiers, source code directories, Version Control System (VCS) commits tracking evolution over time, up to the full states of VCS repositories as observed by Software Heritage during periodic crawls. The dataset's contents come from major development forges (including GitHub and GitLab), FOSS distributions (e.g., Debian), and language-specific package managers (e.g., PyPI). Crawling information is also included, providing timestamps about when and where all archived source code artifacts have been observed in the wild.

Update Frequency

Data is updated yearly

License

Creative Commons Attribution 4.0 International.By accessing the dataset, you agree with the Software Heritage Ethical Charter for using the archive data and the terms of use for bulk access.

Documentation

https://docs.softwareheritage.org/devel/swh-dataset/graph/athena.html

Managed By

Software Heritage

See all datasets managed by Software Heritage.

Contact

aws@softwareheritage.org

How to Cite

Software Heritage Graph Dataset was accessed on DATE from https://registry.opendata.aws/software-heritage.

Resources on AWS

Description
Software Heritage Graph Dataset

Resource type
S3 Bucket

Amazon Resource Name (ARN)
arn:aws:s3:::softwareheritage

AWS Region
us-east-1

AWS CLI Access (No AWS account required)
aws s3 ls --no-sign-request s3://softwareheritage/

Description
S3 Inventory files

Resource type
S3 Bucket

Amazon Resource Name (ARN)
arn:aws:s3:::softwareheritage-inventory

AWS Region
us-east-1

AWS CLI Access (No AWS account required)
aws s3 ls --no-sign-request s3://softwareheritage-inventory/

- all the file contents (the leaves of the graph ~1PB uncompressed)
- regular dumps of the graph (in ORC file format)

Selected research works using Software Heritage

Davide Rossi, Stefano Zacchiroli
Worldwide Gender Differences in Public Code Contributions [...]
ICSE SEIS 2022: The 44th International Conference on Software Engineering

Daniele Serafini, Stefano Zacchiroli
Efficient Prior Publication Identification for Open Source Code
OSS+OpenSym 2022: 18th International Conference on Open Source Systems

Thibault Allançon, Antoine Pietri, Stefano Zacchiroli
The Software Heritage Filesystem (SwhFS): Integrating Source Code Archival with Development
ICSE 2021: The 43rd International Conference on Software Engineering

Antoine Pietri, Guillaume Rousseau, Stefano Zacchiroli
Forking Without Clicking: on How to Identify Software Repository Forks
MSR 2020: 17th Intl. Conf. on Mining Software Repositories. IEEE

Paolo Boldi, Antoine Pietri, Sebastiano Vigna, Stefano Zacchiroli
Ultra-Large-Scale Repository Analysis via Graph Compression
SANER 2020, 27th Intl. Conf. on Software Analysis, Evolution and Reengineering. IEEE

Roberto Di Cosmo, Guillaume Rousseau, Stefano Zacchiroli
Software Provenance Tracking at the Scale of Public Source Code
Empirical Software Engineering 25(4): 2930-2959 (2020)

Outline

Reproducible Builds <-> Software Heritage

- Software Heritage provides key ingredients for R-B pipelines: on-demand archival (e.g., of VCS commits referenced by build recipes) + long-term availability
- We have implemented this by integrating the GNU Guix package manager with Software Heritage

Software Heritage and GNU Guix join forces to enable long term reproducibility

Connecting reproducible deployment to a long-term source code archive

Ludovic Courtès — March 29, 2019

GNU Guix can be used as a "package manager" to install and upgrade software packages as is familiar to GNU/Linux users, or as an environment manager, but it can also provision containers or virtual machines, and manage the operating system running on your machine.

One foundation that sets it apart from other tools in these areas is *reproducibility*. From a high-level view, Guix allows users to *declare* complete software environments and instantiate them. They can share those environments with others, who can replicate them or adapt them to their needs. This aspect is key to reproducible computational experiments: scientists need to reproduce software environments before they can reproduce experimental results, and this is one of the things we are focusing on in the context of the Guix-HPC effort. At a lower level, the project, along with others in the Reproducible Builds community, is working to ensure that software build outputs are reproducible, bit for bit.

Work on reproducibility at all levels has been making great progress. Guix, for instance, allows you to travel back in time. That Guix can travel back in time *and* build software reproducibly is a great step forward. But there's still an important piece that's missing to make this viable: a stable source code archive. This is where Software Heritage (SWH for short) comes in.

When source code vanishes

- https://www.softwareheritage.org/2019/04/18/software-heritage-and-gnu-guix-join-forces-to-enable-long-term-reproducibility
- https://guix.gnu.org/blog/2019/connecting-reproducible-deployment-to-a-long-term-source-code-archive/

Learn more

reproducible-builds.org

softwareheritage.org

Roberto Di Cosmo, Stefano Zacchiroli
Software Heritage: Why and How to Preserve Software Source Code
iPRES 2017: Intl. Conf. on Digital Preservation

Chris Lamb, Stefano Zacchiroli
Reproducible Builds: Increasing the Integrity of Software Supply Chains
IEEE Softw. 39(2): 62-70 (2022)

Appendix

Outline

Open compliance vs Source code scanning

Definition (Open Compliance)

The pursuit of compliance with *license obligations* and other *best practices* for the management of open source software components, using only open technologies such as: open source software, open data information, and open access documentation.

Why

Reduced lock-in risks, lower total cost of ownership (TCO), crowdsourcing, alignment with FOSS community ethos.

We still lack a source code scanning tool that is compliant with Open Compliance principles and addresses industry practical needs.

Can we build one on top of Software Heritage?

swh-scanner

Vision

`swh-scanner` is an open source and open data source code scanner for open compliance workflows, backed by the largest public archive of FOSS source code.

Design

- Query Software Heritage as source of truth about public code
- Leverages the Merkle DAG model and SWHIDs for maximum scanning efficiency
  - E.g., no need to query the back-end for files contained in a known directory
- File-level granularity
- Output: source tree partition into known (= published before) v. unknown

Source: gitlab.softwareheritage.org/swh/devel/swh-scanner
License: GPL-3+
Package: pypi.org/project/swh.scanner

swh-scanner demo — Summary output

```
$ pip install swh.scanner
```

```
$ swh scanner scan /srv/src/linux/linux-5.9.1/kernel
Files:                    471
    known:        471 (100%)
directories:               20
      fully-known:         20 (100%)
  partially-known:          0 (  0%)

(see other --output-format for more details)
```

swh-scanner demo — Machine-readable output

```
$ swh scanner scan --help
...
  -f, --output-format [summary|text|json|ndjson|sunburst]
  The output format  [default: summary]
  -i, --interactive                Show the result in a dashboard
...

$ swh scanner scan --output-format ndjson /srv/src/linux/linux-5.9.1/kernel
{".": {"swhid": "swh:1:dir:5f18abc022c8aa2652008...", "known": true}}
{"cpu_pm.c": {"swhid": "swh:1:cnt:44a259338e33d1...", "known": true}}
{"sys.c": {"swhid": "swh:1:cnt:ab6c409b1159b1538...", "known": true}}
{"audit.c": {"swhid": "swh:1:cnt:7efaece534a9f69...", "known": true}}
{"torture.c": {"swhid": "swh:1:cnt:1061492f14bd9...", "known": true}}
{"smpboot.c": {"swhid": "swh:1:cnt:2efe1e206167c...", "known": true}}
{"gen_kheaders.sh": {"swhid": "swh:1:cnt:c1510f0...", "known": true}}
{"task_work.c": {"swhid": "swh:1:cnt:d621006f007...", "known": true}}
{"Kconfig.hz": {"swhid": "swh:1:cnt:38ef6d06888e...", "known": true}}
{"up.c": {"swhid": "swh:1:cnt:c6f323dcd45bb9efe1...", "known": true}}
...
```

swh-scanner demo — Efficiency

```
$ du -sh --exclude=.git /srv/src/linux/git
4,1G /srv/src/linux/git
```

```
$ time swh scanner scan /srv/src/linux
Files:                  78277
    known:      78267 ( 99%)
directories:             5085
      fully-known:       5081 ( 99%)
  partially-known:          4 (  0%)

38,65s user 4,71s system 81% cpu 53,127 total
```

```
$ swh scanner scan --output-format ndjson /srv/src/linux/git | grep false
...
{"scripts/kconfig/symbol.o": {"swhid": "swh:1:cnt:874f19...", "known": false}}
...
```

swh-scanner — Going further

Roadmap

- License information → in-house scanning + forge metadata (e.g., GitHub)
- Provenance information → Software Heritage crawling info
- "Serious UI" for interactive dashboard, based on UX design and user testing
- Increase granularity to snippet/SLOC investments

(Some of these are low-hanging fruits, some require substantial R&D investments.)

Feedback welcome

- Feel free to play with swh-scanner, feedback is very welcome!
- Caveat: API rate-limit (talk to us for lifting it)

Outline

A peek at the dataset

Accessing graph leaves (a.k.a. contents)

```
$ aws s3 ls --no-sign-request s3://softwareheritage/
       PRE content/
       PRE graph/
```

A peek at the dataset

Accessing graph leaves (a.k.a. contents)

```
$ aws s3 ls --no-sign-request s3://softwareheritage/
       PRE content/
       PRE graph/
```

File contents can be accessed using their SHA1 checksum

```
$ aws s3 cp --no-sign-request \
  s3://softwareheritage/content/8624bcdae55baeef00cd11d5dfcfa60f68710a02 .
```

Notice that file contents are compressed:

```
$ zcat 8624bcdae55baeef00cd11d5dfcfa60f68710a02 | head
  GNU GENERAL PUBLIC LICENSE
     Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

A peek at the dataset, cont'd

Annual dumps of (inner nodes of) the full graph

```
$ aws s3 ls --no-sign-request s3://softwareheritage/graph/
```

```
2018-09-25/
2019-01-28-popular-3k-python/
2019-01-28-popular-4k/
2020-05-20/
2020-12-15/
2021-03-23-cpython-3-5/
2021-03-23-popular-3k-python/
2021-03-23/
2022-04-25/
```

How to use

- online full documentation
- Antoine Pietri's PhD Thesis

How to cite

Antoine Pietri, Diomidis Spinellis, Stefano Zacchiroli. *The Software Heritage Graph Dataset: Public software development under one roof.* MSR 2019. (bibtex)

Example: most popular commit verbs (stemmed)

Query using Amazon Athena

```
SELECT COUNT(*) AS C, word FROM (
   SELECT word_stem(lower(split_part(
    trim(from_utf8(message)),' ', 1)))
   AS word FROM revision
   WHERE length(message) < 1000000)
WHERE word != ''
GROUP BY word
ORDER BY C
DESC LIMIT 20;
```

Total cost: approximately .5 euros

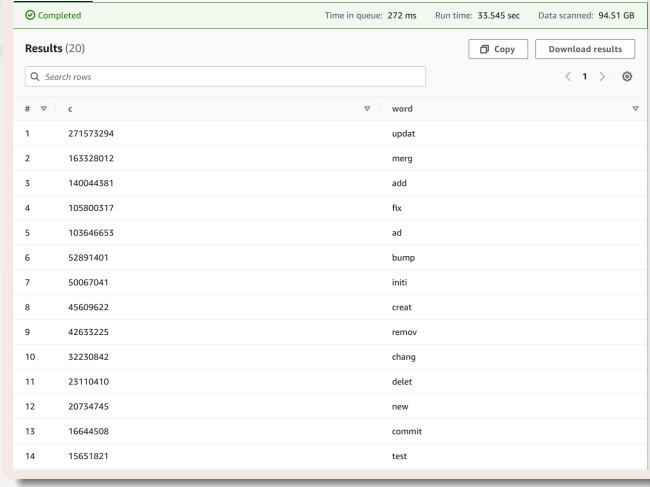
Example: most popular commit verbs (stemmed)

Query using Amazon Athena

```
SELECT COUNT(*) AS C, word FROM (
   SELECT word_stem(lower(split_part(
    trim(from_utf8(message)),' ', 1)))
   AS word FROM revision
   WHERE length(message) < 1000000)
WHERE word != ''
GROUP BY word
ORDER BY C
DESC LIMIT 20;
```

Total cost: approximately .5 euros

Results



Completed — Time in queue: 272 ms — Run time: 33.545 sec — Data scanned: 94.51 GB

Results (20) — Copy — Download results

Search rows — 1

#	c	word
1	271573294	updat
2	163328012	merg
3	140044381	add
4	105800317	fix
5	103646653	ad
6	52891401	bump
7	50067041	initi
8	45609622	creat
9	42633225	remov
10	32230842	chang
11	23110410	delet
12	20734745	new
13	16644508	commit
14	15651821	test

Outline

Going beyond SQL

State-of-the-art graph compression from social networks

Paolo Boldi, Antoine Pietri, Sebastiano Vigna, Stefano Zacchiroli

Ultra-Large-Scale Repository Analysis via Graph Compression

SANER 2020, 27th Intl. Conf. on Software Analysis, Evolution and Reengineering. IEEE

Results

Full graph structure (25 B nodes, 350 B edges) in 200 GiB RAM

- traversal time is tens of ns per edge
- bidirectional traversals implemented
- beware: metadata access is still *off RAM*

Java and gRPC APIs available

docs.softwareheritage.org/devel/swh-graph/grpc-api.html

Find all origins containing a given content

```
grpc_cli call localhost:50091 swh.graph.TraversalService.Traverse "\
src: 'swh:1:cnt:8722d84d658e5e11519b807abb5c05bfbfc531f0', direction: BACKWARD, \
mask: {paths: ['swhid','ori.url']}, return_nodes: {types: 'ori'}"
```

Gives a list of origins including "https://github.com/rdicosmo/parmap", encoded as "swh:1:ori:8903a90cff8f07159be7aed69f19d66d33db3f86" (beware: this is not a SWHID!)

Find all origins containing a given content

```
grpc_cli call localhost:50091 swh.graph.TraversalService.Traverse "\
src: 'swh:1:cnt:8722d84d658e5e11519b807abb5c05bfbfc531f0', direction: BACKWARD, \
mask: {paths: ['swhid','ori.url']}, return_nodes: {types: 'ori'}"
```

Gives a list of origins including "https://github.com/rdicosmo/parmap", encoded as "swh:1:ori:8903a90cff8f07159be7aed69f19d66d33db3f86" (beware: this is not a SWHID!)

Shortest provenance path of a content in a given origin

```
grpc_cli call localhost:50091 swh.graph.TraversalService.FindPathBetween "\
src: 'swh:1:ori:8903a90cff8f07159be7aed69f19d66d33db3f86', \
dst: 'swh:1:cnt:8722d84d658e5e11519b807abb5c05bfbfc531f0', \
mask: {paths: ['swhid']}" | egrep 'swhid'
connecting to localhost:50091
  swhid: "swh:1:ori:8903a90cff8f07159be7aed69f19d66d33db3f86"
  swhid: "swh:1:snp:1527a93b039d70f6a781b05d76b77c6209912887"
  swhid: "swh:1:rev:82df563aecf86b9164eee7d10d40f2d8cbd1c78d"
  swhid: "swh:1:dir:484db39bb2825886191837bb0960b7450f9099bb"
  swhid: "swh:1:dir:4d15e44b378fe39dd23817abee756cd47ad14575"
  swhid: "swh:1:cnt:8722d84d658e5e11519b807abb5c05bfbfc531f0"
Rpc succeeded with OK status
```