Building Blocks for a Safer Open Source Supply Chain

Reproducible Builds and Software Heritage

Stefano Zacchiroli

Software Heritage
Télécom Paris, Polytechnic Institute of Paris

29 May 2024
INFORTECH Day 2024, Université de Mons
Mons, Belgium

Software Heritage
THE GREAT LIBRARY OF SOURCE CODE

Outline

1. Introduction
2. Open Source Software Supply Chain — Attacks
3. Reproducible Builds
4. Open Source Software Supply Chain — KYSW
5. Software Heritage
6. Conclusion

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

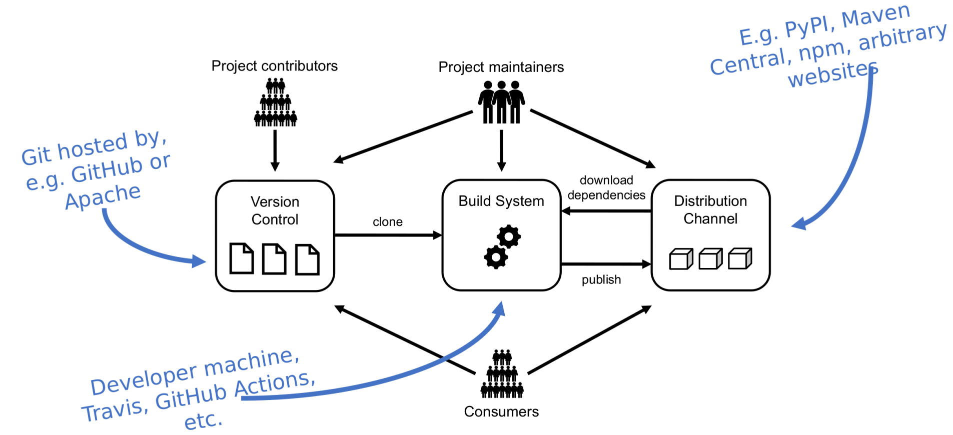
1. Introduction
2. Open Source Software Supply Chain — Attacks
3. Reproducible Builds
4. Open Source Software Supply Chain — KYSW
5. Software Heritage
6. Conclusion

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



Git hosted by, e.g. GitHub or Apache

E.g. PyPI, Maven Central, npm, arbitrary websites

Developer machine, Travis, GitHub Actions, etc.

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

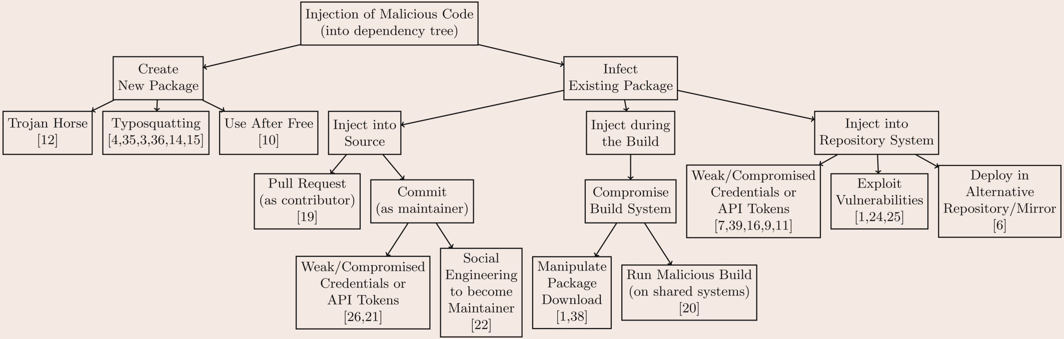
- NotPetya (2017): ransomware concealed in an update of a popular accounting software, hitting Ukranian banks and major corps (B$)
- CCleaner (2017): malicious version of a popular MS Windows maintenance tool, distributed via the vendor website
- SolarWinds (2020): malicious update of the SolarWinds Orion monitoring software, shipping a delayed-activation trojan. Breached into several US Gov. branches as well as Microsoft
- XZ (2024): "Jia Tan" social engineers their way into becoming maintainer of XZ and plant a backdoor targeting SSH, allowing remote command execution

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.
- Example: removing `left-pad`, a 8-line(!) library to align strings, from NPM broke "many thousands of projects" in 2016, including high-profile ones from Big Tech.

- 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



Injection of Malicious Code (into dependency tree)

Create New Package

Infect Existing Package

Trojan Horse [12]

Typosquatting [4,35,3,36,14,15]

Use After Free [10]

Inject into Source

Inject during the Build

Inject into Repository System

Pull Request (as contributor) [19]

Commit (as maintainer)

Compromise Build System

Weak/Compromised Credentials or API Tokens [7,39,16,9,11]

Exploit Vulnerabilities [1,24,25]

Deploy in Alternative Repository/Mirror [6]

Weak/Compromised Credentials or API Tokens [26,21]

Social Engineering to become Maintainer [22]

Manipulate Package Download [1,38]

Run Malicious Build (on shared systems) [20]

(image from: Ohm et al. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. DIMVA 2020)

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

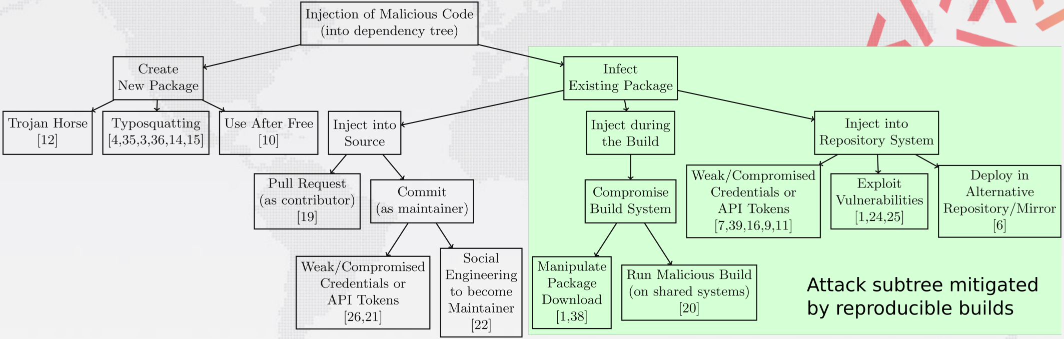
Related attack vectors: Inject into [Package] Repository System (!= VCS)

Outline

1. Introduction
2. Open Source Software Supply Chain — Attacks
3. Reproducible Builds
4. Open Source Software Supply Chain — KYSW
5. Software Heritage
6. Conclusion

Problem statement

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


Injection of Malicious Code (into dependency tree)
Create New Package
Trojan Horse [12]
Typosquatting [4,35,3,36,14,15]
Use After Free [10]
Infect Existing Package
Inject into Source
Pull Request (as contributor) [19]
Commit (as maintainer)
Weak/Compromised Credentials or API Tokens [26,21]
Social Engineering to become Maintainer [22]
Inject during the Build
Compromise Build System
Manipulate Package Download [1,38]
Run Malicious Build (on shared systems) [20]
Inject into Repository System
Weak/Compromised Credentials or API Tokens [7,39,16,9,11]
Exploit Vulnerabilities [1,24,25]
Deploy in Alternative Repository/Mirror [6]

Attack subtree mitigated by reproducible builds

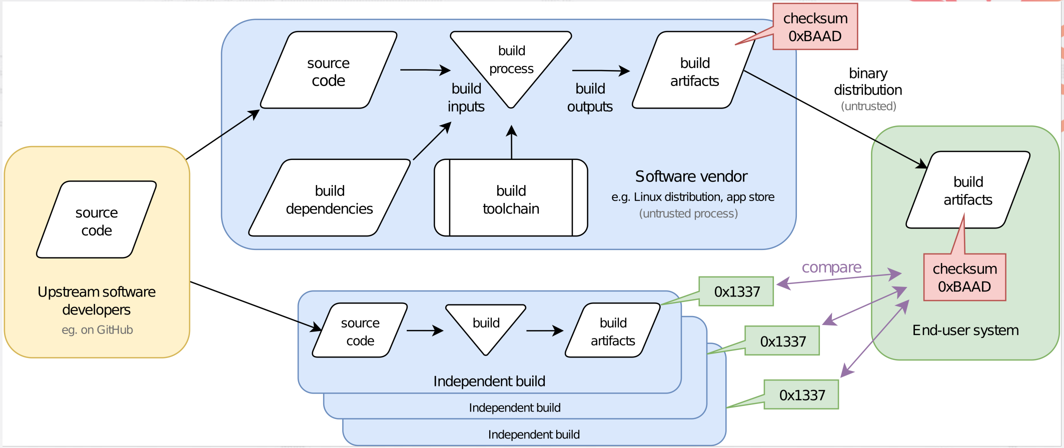
A reproducible build (r-b) process

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

> *The build process of a software product is [bitwise] 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


checksum
0xBAAD
source
code
build
process
build
inputs
build
outputs
build
artifacts
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
compare
checksum
0xBAAD
End-user system
0x1337
source
code
build
build
artifacts
0x1337
0x1337
Independent build
Independent build
Independent build

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 have fixed all micro-issues that impede build reproducibility
- How do we go about making large FOSS software collections reproducible?

Experiment: making all Debian packages build reproducibly from source

- Debian: one of the most popular GNU/Linux distro, esp. in the server market
- 30'000+ (source) packages, 1+B lines of code
- Goals:
  1. Empirical experiment to identify common causes of non-reproducibility
  2. Real impact (if successful) due to Debian popularity in the market

Approach

Establish a corresponding Quality Assurance process and soft-enforce it using Continuous Integration (CI).

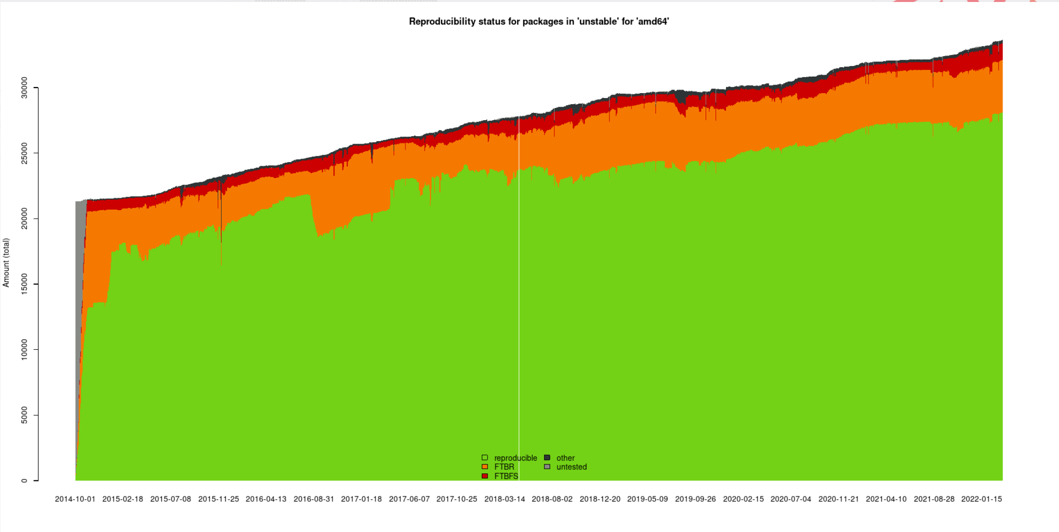
Adversarial rebuilding

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: host name, locales, kernel
  - Reverse filesystem ordering using disorderfs
  - …

30+ variations in total

Reproducible Debian — Evolution over time



The Reproducible Builds ecosystem

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)

Outline

1. Introduction
2. Open Source Software Supply Chain — Attacks
3. Reproducible Builds
4. Open Source Software Supply Chain — KYSW
5. Software Heritage
6. Conclusion

KYSW (Know Your SoftWare)

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

KYSW (Know Your SoftWare)

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

Vertical approach: secure your software

Improve security of *each component* separately

- By law: e.g. EU Cyber Resilience Act
- By practice: e.g. `https://best.openssf.org/`

KYSW (Know Your SoftWare)

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

Vertical approach: secure your software

Improve security of *each component* separately

- By law: e.g. EU Cyber Resilience Act
- By practice: e.g. `https://best.openssf.org/`

Horizontal approach: all the supply chain

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*

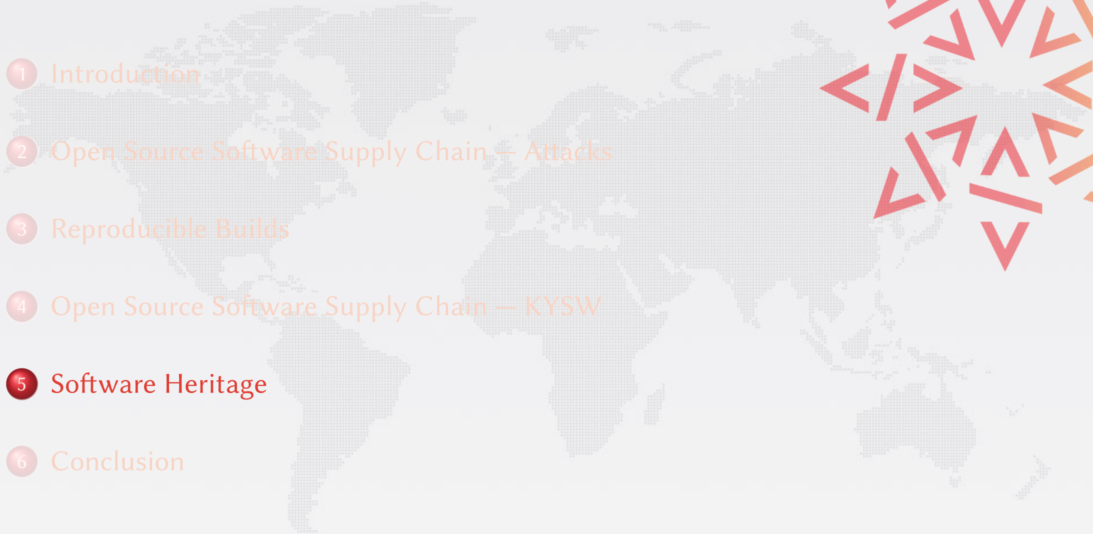
A few key challenging properties

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



1. Introduction
2. Open Source Software Supply Chain — Attacks
3. Reproducible Builds
4. Open Source Software Supply Chain — KYSW
5. Software Heritage
6. Conclusion

Software Heritage, in a nutshell

www.softwareheritage.org

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, in a nutshell

www.softwareheritage.org

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

Debian
Sourceforge
CPAN
Gitorious
Maven
Inria
Bitbucket
GitHub
BerliOs
CTAN
GitLab
CRAN
GoogleCode
Adullact

find and reference all software source code

Software Heritage, in a nutshell

www.softwareheritage.org

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

Debian
Sourceforge
CPAN
Gitorious
Maven
Inria
Bitbucket
GitHub
BerliOs
CTAN
GitLab
CRAN
GoogleCode
Adullact

find and reference all software source code

Universal archive

damage
storage
disaster
reference
deletion
media
malicious
obsolete
attack
aging
dependencies
tear
dangling
wear
corruption
encryption
format

preserve and share all software source code

Software Heritage, in a nutshell

www.softwareheritage.org

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

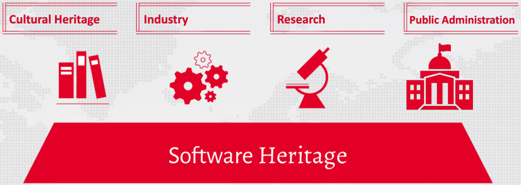
preserve and share all software source code

Research infrastructure

enable analysis of all software source code

The largest software archive, a shared infrastructure

One infrastructure
open and shared



Cultural Heritage

Industry

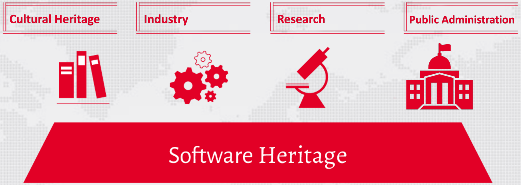
Research

Public Administration

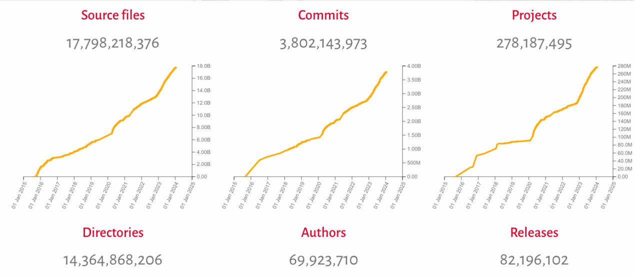
Software Heritage

The largest software archive, a shared infrastructure

One infrastructure
open and shared



The largest archive ever built



Source files

17,798,218,376

Commits

3,802,143,973

Projects

278,187,495

Directories

14,364,868,206

Authors

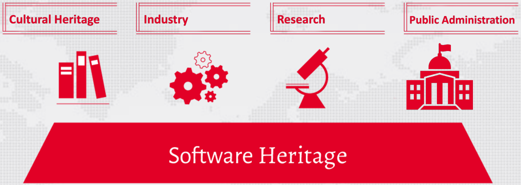
69,923,710

Releases

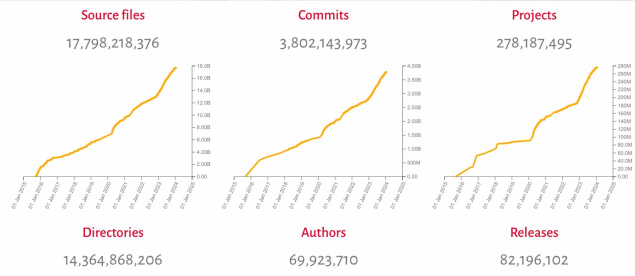
82,196,102

The largest software archive, a shared infrastructure

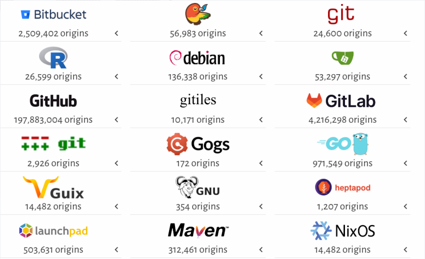
One infrastructure
open and shared



The largest archive ever built

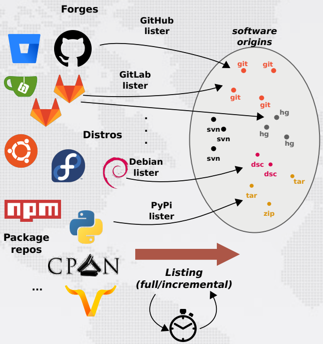


Source files	Commits	Projects
17,798,218,376	3,802,143,973	278,187,495
Directories	**Authors**	**Releases**
14,364,868,206	69,923,710	82,196,102



Bitbucket 2,509,402 origins ‹	56,983 origins ‹	git 24,600 origins ‹
R 26,599 origins ‹	debian 136,338 origins ‹	53,297 origins ‹
GitHub 197,883,004 origins ‹	gitiles 10,171 origins ‹	GitLab 4,216,298 origins ‹
git 2,926 origins ‹	Gogs 172 origins ‹	GO 971,549 origins ‹
Guix 14,482 origins ‹	GNU 354 origins ‹	heptapod 1,207 origins ‹
launchpad 503,631 origins ‹	Maven 312,461 origins ‹	NixOS 14,482 origins ‹

A peek under the hood: a universal archive



Forges

GitHub lister

GitLab lister

Distros

Debian lister

PyPi lister

Package repos

...

software origins

git git git git

svn svn svn

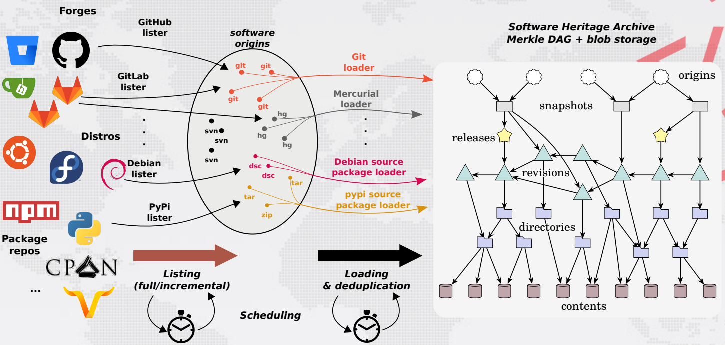
hg hg hg

dsc dsc

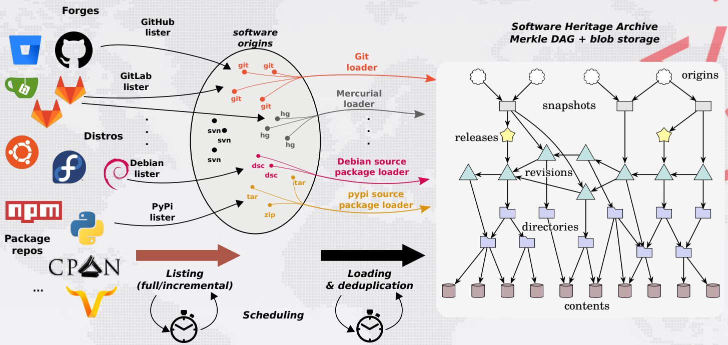
tar tar zip

Listing (full/incremental)

A peek under the hood: a universal archive



A peek under the hood: a universal archive



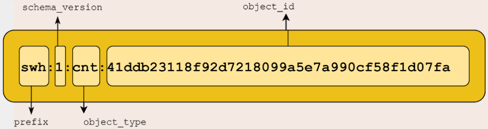
Global development history permanently archived in a uniform data model

- over 18 billion unique source files from over 290 million software projects
- ~1.5PB (compressed) blobs, ~35 B nodes, ~500 B edges

Referencing all source code artifacts with SWHIDs

Software Heritage Identifiers (SWHID)

link to full docs



swh:1:cnt:41ddb23118f92d7218099a5e7a990cf58f1d07fa

35+B

intrinsic,

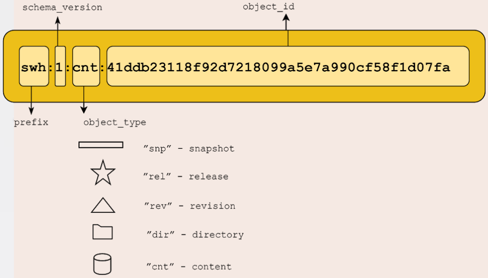
decentralised,

cryptographic

Referencing all source code artifacts with SWHIDs

Software Heritage Identifiers (SWHID)

link to full docs

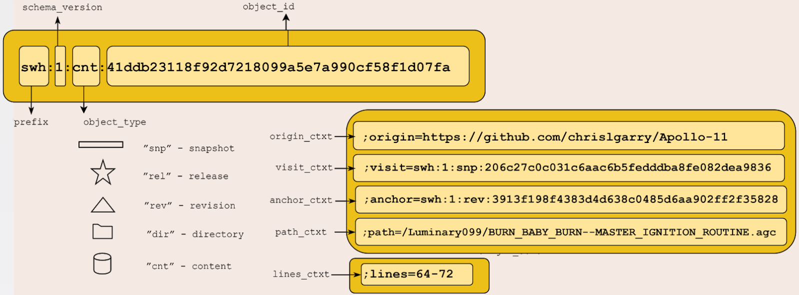


35+B
intrinsic,
decentralised,
cryptographic

Referencing all source code artifacts with SWHIDs

Software Heritage Identifiers (SWHID)

link to full docs



```
swh:1:cnt:41ddb23118f92d7218099a5e7a990cf58f1d07fa
;origin=https://github.com/chrislgarry/Apollo-11
;visit=swh:1:snp:206c27c0c031c6aac6b5fedddba8fe082dea9836
;anchor=swh:1:rev:3913f198f4383d4d638c0485d6aa902ff2f35828
;path=/Luminary099/BURN_BABY_BURN--MASTER_IGNITION_ROUTINE.agc
;lines=64-72
```

35+B

intrinsic,

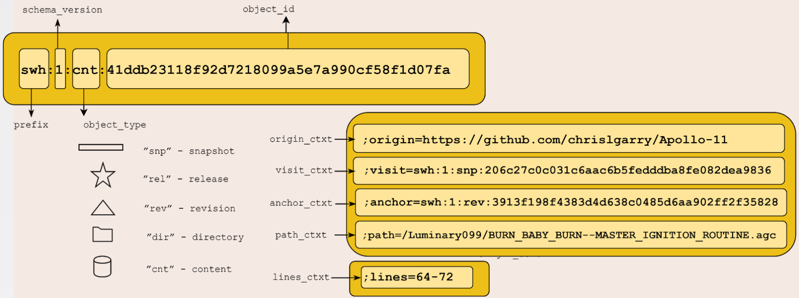
decentralised,

cryptographic

Referencing all source code artifacts with SWHIDs

Software Heritage Identifiers (SWHID)

link to full docs



schema_version

object_id

swh:1:cnt:41ddb23118f92d7218099a5e7a990cf58f1d07fa

prefix

object_type

"snp" - snapshot

"rel" - release

"rev" - revision

"dir" - directory

"cnt" - content

origin_ctxt → ;origin=https://github.com/chrislgarry/Apollo-11

visit_ctxt → ;visit=swh:1:snp:206c27c0c031c6aac6b5fedddba8fe082dea9836

anchor_ctxt → ;anchor=swh:1:rev:3913f198f4383d4d638c0485d6aa902ff2f35828

path_ctxt → ;path=/Luminary099/BURN_BABY_BURN--MASTER_IGNITION_ROUTINE.agc

lines_ctxt → ;lines=64-72

35+B

intrinsic, decentralised, cryptographic

Full fledged *source code references* for traceability, integrity and reproducibility

- Linux Foundation SPDX 2.2
- IANA-registered "swh:"
- WikiData property P6138

Examples: Apollo 11 AGC excerpt, Quake III rsqrt

Guidelines available, see the HOWTO

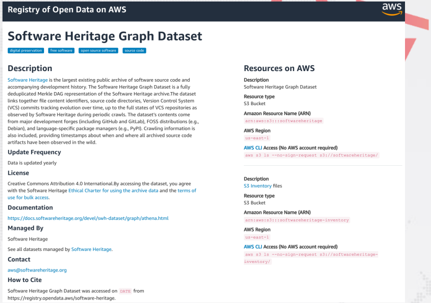
ISO standardization underway, see swhid.org

The Software Heritage archive as an open dataset

1. All the file contents (the leaves of the graph ~1.5 PiB uncompressed)
2. Regular dumps of the graph (with all metadata, in ORC file format)

Antoine Pietri, Diomidis Spinellis, Stefano Zacchiroli
The Software Heritage Graph Dataset: Public software development under one roof
MSR 2019: 16th Intl. Conf. on Mining Software Repositories. IEEE

Self-hosted (10-20 TiB)
docs.softwareheritage.org/devel/swh-dataset/graph/dataset.html



Registry of Open Data on AWS

Software Heritage Graph Dataset

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/
```

Hosted on public clouds
registry.opendata.aws/software-heritage

Selected research works using Software Heritage

- Jesús M. González-Barahona, Sergio Raúl Montes León, Gregorio Robles, Stefano Zacchiroli
  The Software Heritage license dataset (2022 edition)
  Empir. Softw. Eng. 28(6): 147 (2023)

- Romain Lefeuvre, Jessie Galasso, Benoît Combemale, Houari A. Sahraoui, Stefano Zacchiroli:
  Fingerprinting and Building Large Reproducible Datasets
  ACM-REP 2023: 27-36

- Davide Rossi, Stefano Zacchiroli
  Worldwide Gender Differences in Public Code Contributions [...]
  ICSE SEIS 2022: The 44th 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)

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!

- cybersecurity: just launched SWHSec project swhsec.github.io
- AI: providing high-quality data for ethical code LLMs
- an open (source & data) source code scanner for open compliance

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

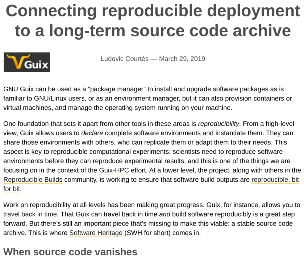
Outline

1. Introduction
2. Open Source Software Supply Chain — Attacks
3. Reproducible Builds
4. Open Source Software Supply Chain — KYSW
5. Software Heritage
6. Conclusion

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

Guix

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

Ludovic Courtès, Timothy Sample, Simon Tournier, Stefano Zacchiroli
Source Code Archiving to the Rescue of Reproducible Deployment.
ACM REP 2024 (to appear)

Learn more

Reproducible Builds

reproducible-builds.org

Software Heritage

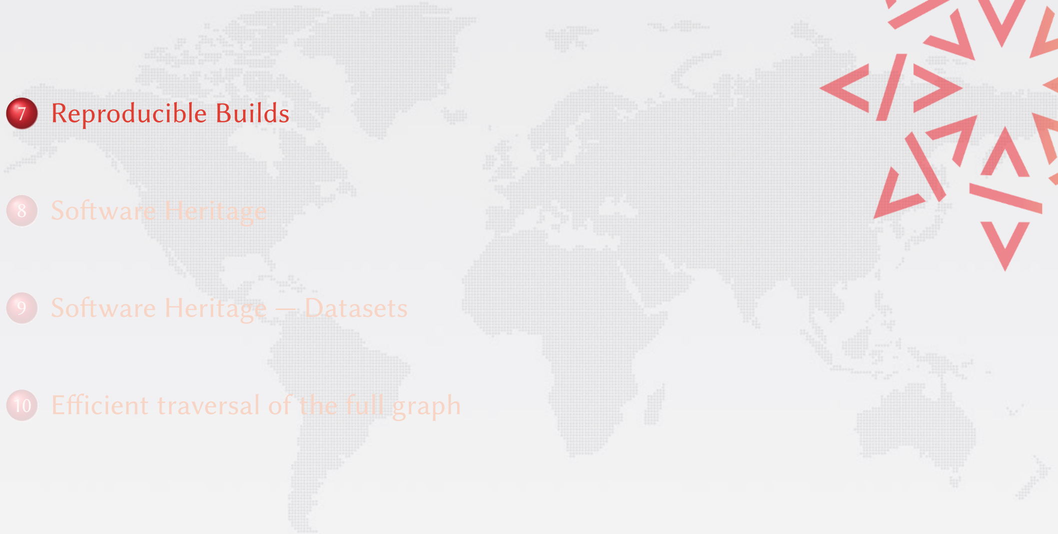
THE GREAT LIBRARY OF SOURCE CODE

softwareheritage.org

- Piergiorgio Ladisa, Henrik Plate, Matias Martinez, Olivier Barais
  SoK: Taxonomy of Attacks on Open-Source Software Supply Chains
  IEEE S&P 2023
- Chris Lamb, Stefano Zacchiroli
  Reproducible Builds: Increasing the Integrity of Software Supply Chains
  IEEE Softw. 39(2): 62-70 (2022)
- Roberto Di Cosmo, Stefano Zacchiroli
  Software Heritage: Why and How to Preserve Software Source Code
  iPRES 2017: Intl. Conf. on Digital Preservation

Appendix

Outline



7 Reproducible Builds

8 Software Heritage

9 Software Heritage — Datasets

10 Efficient traversal of the full graph

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

7 Reproducible Builds

8 Software Heritage

9 Software Heritage — Datasets

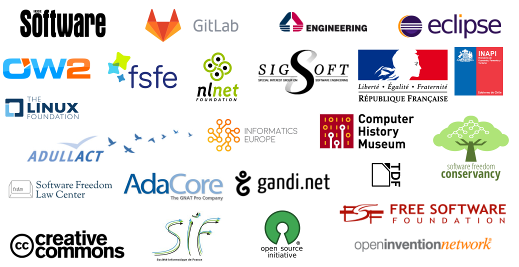
10 Efficient traversal of the full graph

An international, non profit initiative built for the long term

Sharing the vision

UNESCO

United Nations
Educational, Scientific and
Cultural Organization



And many more ...
www.softwareheritage.org/support/testimonials

An international, non profit initiative built for the long term

Sharing the vision

UNESCO
United Nations
Educational, Scientific and
Cultural Organization

IEEE Software · GitLab · Engineering · eclipse

OW2 · fsfe · nlnet foundation · SIGSOFT · Liberté • Égalité • Fraternité République Française · INAPI

The Linux Foundation

Adullact · Informatics Europe · Computer History Museum · software freedom conservancy

Software Freedom Law Center · AdaCore The GNAT Pro Company · gandi.net · TDF

creative commons · SIF Société Informatique de France · open source initiative · Free Software Foundation · openinventionnetwork

And many more …
www.softwareheritage.org/support/testimonials

Donors, members, sponsors

Inria

Diamond sponsor

cea

Platinum sponsors

cnrs · intel · Ministère de l'Enseignement Supérieur et de la Recherche · Microsoft · Huawei

Gold sponsors

Hugging Face · openinventionnetwork · servicenow

Société Générale · Sorbonne Université · Université Paris Cité

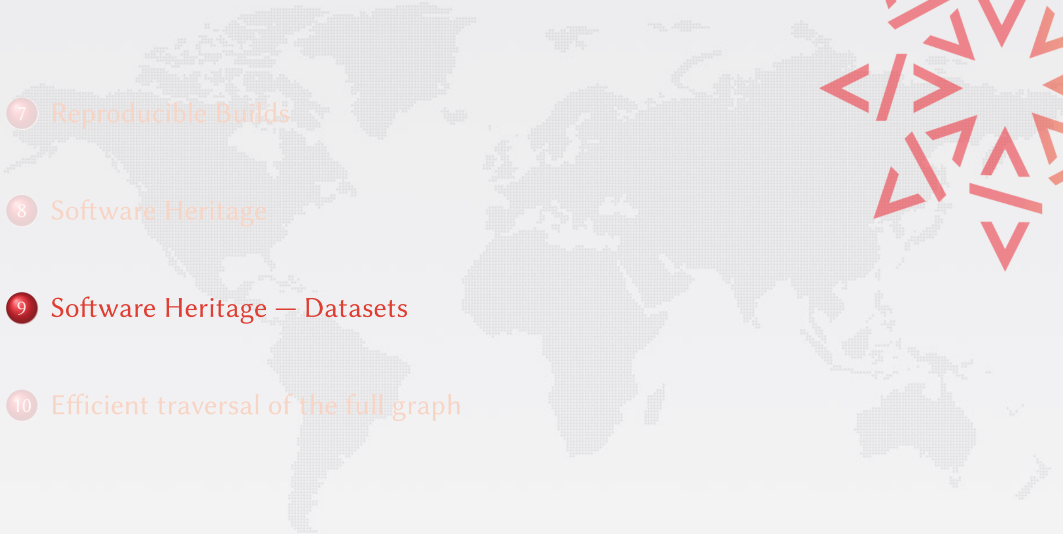
Silver sponsors

AdaCore · Ministère des Armées · DGA · Liberté • Égalité • Fraternité République Française · GitHub · Google · Università di Pisa

Bronze sponsors

SCANOSS · Scuola Normale Superiore · Alma Mater Studiorum · Université de Lorraine

Outline



7 Reproducible Builds

8 Software Heritage

9 Software Heritage — Datasets

10 Efficient traversal of the full 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/
```

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/   2021-03-23-cpython-3-5/
   2019-01-28-popular-4k/          2021-03-23-popular-3k-python/
   2020-05-20/                     2021-03-23/
   2020-12-15/                     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

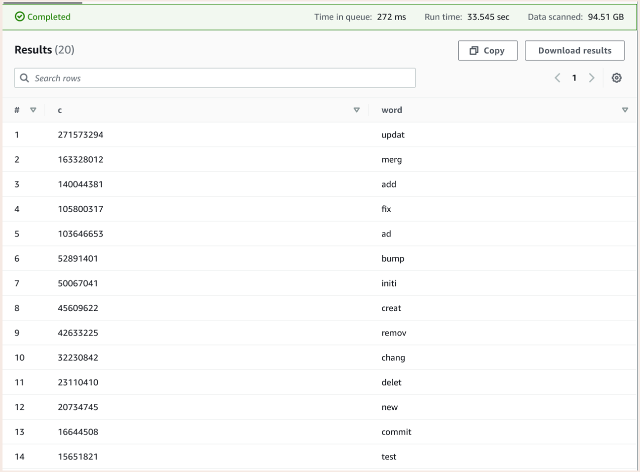
Example: most popular commit verbs (stemmed)

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)

#	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

7 Reproducible Builds

8 Software Heritage

9 Software Heritage — Datasets

10 Efficient traversal of the full graph

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

Examples *assume graph service on localhost:50091*

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!)

Examples *assume graph service on localhost:50091*

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
```