

Securing the Software Supply Chain with Software Heritage

Stefano Zacchioli

Institut Polytechnique de Paris & Software Heritage
`stefano.zacchioli@telecom-paris.fr`

7 October 2026 — Open Source Conference Luxembourg, Belval



Software Heritage
THE GREAT LIBRARY OF SOURCE CODE

- 
- 1 Introduction
 - 2 Software Heritage
 - 3 Detecting Git history alterations
 - 4 Detecting Git tag alterations
 - 5 Detecting one-day vulnerabilities across forks
 - 6 Conclusion

About the speaker

- Professor of Computer Science, Télécom Paris, Institut Polytechnique de Paris
- Software Heritage co-founder & Chief Scientific Officer (CSO)
- Free/Open Source Software activist (20+ years); Debian Developer & former 3x Debian Project Leader; former OSI director

Today's agenda

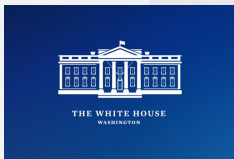
- 1 What is **Software Heritage** (for those who haven't met it yet)
- 2 Detecting **repository alterations**: rewritten histories and mutated tags
- 3 Detecting **one-day vulnerabilities** hiding in forks

Open source security, policy

With great exposure comes great scrutiny...

... by both good and bad actors

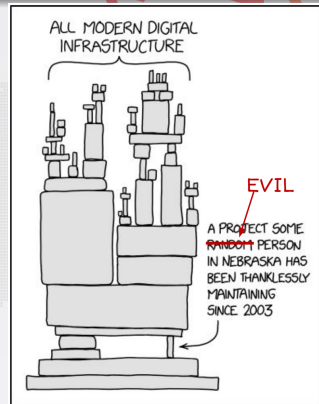
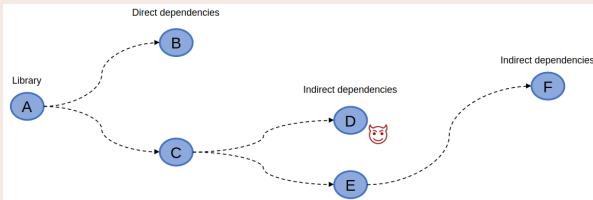
- OSS is more and more **targeted by attackers**: *supply chain attacks*
 - event-stream (2018) · Log4Shell (2021) · node-ipc (2022) · **XZ utils (2024)**
- Increased **policy attention** to secure OSS, e.g.:
 - US: Biden's executive orders (2022, Jan 2025!)
 - EU: **Cyber Resilience Act (CRA)**, progressively coming into effect
 - open source compliance work organised in the **ORC WG**



Software supply chain attacks

Reusing OSS via dependencies

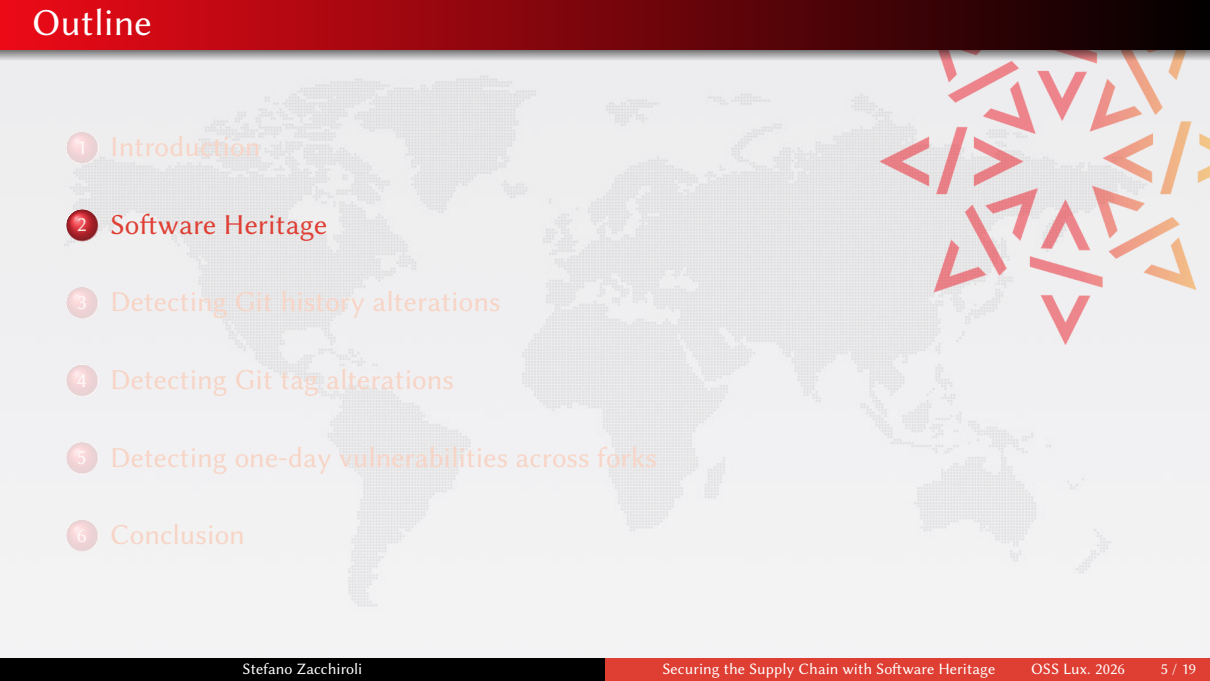
- **Software dependencies**: a popular way of reusing open source software.
- Software product *A* uses functionalities implemented in OSS product *B* ... and so on.



based on xkcd.com/2347

Attacking the software supply chain

- Attacking **undermaintained "leaf" packages** (e.g., D) → efficient attack strategy
- Many documented attacks: event-stream (2018), node-ipc (2022), XZ utils (2024), ...

- 
- 1 Introduction
 - 2 Software Heritage
 - 3 Detecting Git history alterations
 - 4 Detecting Git tag alterations
 - 5 Detecting one-day vulnerabilities across forks
 - 6 Conclusion



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

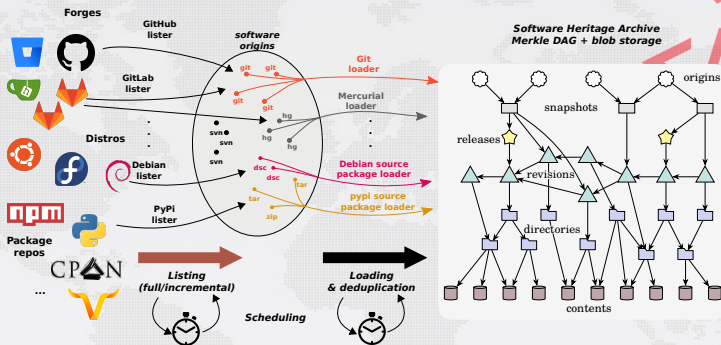


The largest archive ever built



 6,641 origins	 5,812,244 origins	 36,854 origins
 33,338 origins	 30,347 origins	 30,318 origins
 111,363 origins	 2,739 origins	 147,085 origins
 64,348 origins	 35,006 origins	 205,412,724 origins
 25,195 origins	 5,917,204 origins	 3,481 origins
 494 origins	 2,872,464 origins	 79,860 origins
 334 origins	 16,043 origins	 1,124 origins
 668,749 origins	 650,849 origins	 75,337 origins
 4,825,279 origins	 5,321 origins	 184,464 origins
 107,321 origins	 199 origins	 26,985 origins
 7,626 origins	 755,060 origins	 182,012 origins
 382,388 origins	 344 origins	

A peek under the hood: a universal archive



Global development history permanently archived in a uniform data model

- over 29.5 billion unique source files from over 441 million software projects
- ~3PB (compressed) blobs, ~60 B nodes, over 1 trillion edges

Securing open source with Software Heritage

What does Software Heritage bring to the table?

- The largest archive that guarantees the:
 - 1 availability,
 - 2 integrity, and
 - 3 traceability of (OSS) source code.
- A **universal, open knowledge base** of *facts* about open source software...
- ...that can be leveraged by everyone (not only the big players) to secure OSS.

SWHSec project

swhsec.github.io

- 2023–2027 R&D project, funded by French national CampusCyber
- 8 research teams, including SWH core



Software Heritage
THE GREAT LIBRARY OF SOURCE CODE

Axes: (1) extending SWH with security info + (2) code analysis, dependency analysis, **vulnerability tracking**, automatic vulnerability fixing, ... at SWH scale.

- 
- 1 Introduction
 - 2 Software Heritage
 - 3 Detecting Git history alterations**
 - 4 Detecting Git tag alterations
 - 5 Detecting one-day vulnerabilities across forks
 - 6 Conclusion

Git repository alterations

Git allows **rewriting history** (of the version control kind!)

```
$ git rebase --interactive <...>  
$ git push --force
```

- Useful feature! E.g., to clean your code before sharing it
- Also annoying and **risky** on public branches
 - **Hinders reproducibility** and **voids availability** of specific Git objects
 - **Supply chain concerns:** what was altered and why?

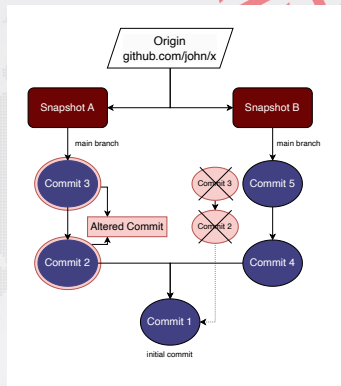


- ❶ How often are public Git histories altered?
- ❷ When this happens, what is changed and why?

Note: forges do not keep the history of these modifications. SWH is the only place where they can be analyzed globally.

A large-scale analysis of Git repository alterations

- 1 Retrieve from Software Heritage **111 M Git repositories** (1) archived at least twice, (2) with different states ("snapshots")
- 2 For each repository, compare snapshots 2-by-2 to detect **altered histories** and the **root cause** of each alteration
- 3 Classify **altered commits** by what changed before/after alteration



Solal Rapaport, Laurent Pautet, Samuel Tardieu, Stefano Zacchiroli

Altered Histories in Version Control System Repositories: Evidence from the Trenches

ASE 2025 <https://arxiv.org/abs/2509.09294>

Key findings on destructive repository alterations

How often?

- **1.22M repositories** contain altered histories (~1.1%)
- **8.7M altered commits** → Pro tip: make sure your important commits are in SWH!

Where?

- Pull request branches: 37.6% → Might be OK, depending on the workflow
- **main/master branches**: 11.4% → Concerning!

What?

- **Commit metadata** (13.3%): author, date, message, ... → Risky for provenance & IP
- **File/dir. changes** (76.8%): *retroactive* file modifications and/or deletions

Case study #1: License changes

- ~800K **retroactively altered license files** on main branches
- Spanning **32k repositories** (76 with 1000+ stars)
- 79% version updates (e.g., GPL 2→3)
- 14% full changes (e.g., MIT→GPL)
- Serious concern: retroactive changes may *de facto* suppress previously granted rights (without an archival copy!)

Case study #2: Removing secrets

- 13M file removals involved files/paths referring to "secrets"
 - Examples: **private keys, certificates, passwords**
- Spanning 75k repositories
- Issue: **History alteration** $\nrightarrow$ **security** (archived copies persist)
- Keys must be rotated, not only purged from Git
- Might indicate **poor security practices**.

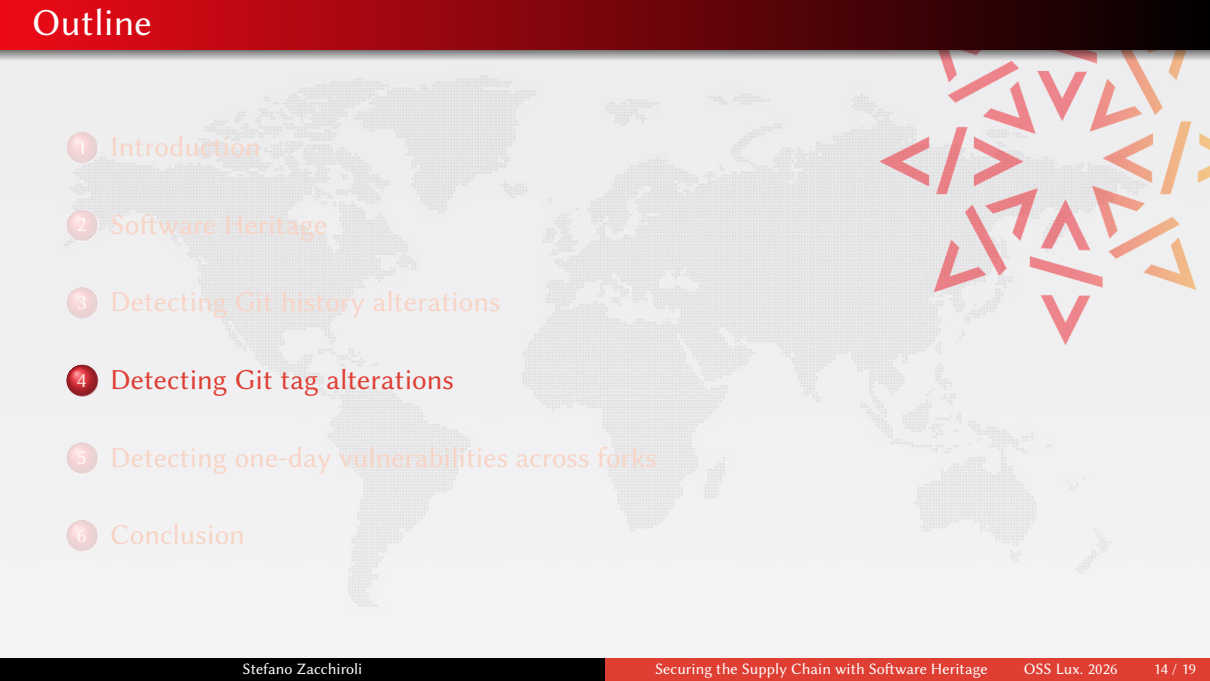
- Imagine you would like to **avoid repositories** with a track record of **history alterations**, or at least be alerted about them, for vetting purposes. How can you?
- For demonstration purposes only, we developed **GitHistorian**, a prototype OSS tool that leverages SWH data to address this need.

```
$ git-historian check https://github.com/example/project --branch main --verbose
Connected to the Software Heritage database!
Found 2 altered history records for 'https://github.com/example/project'
```

Record #1:

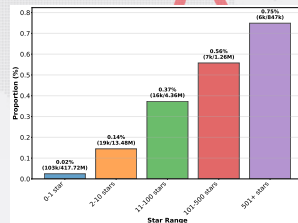
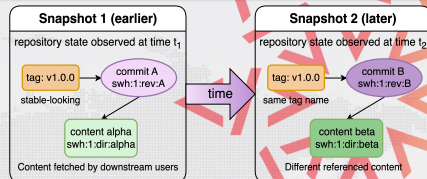
```
Branch Name: refs/heads/master
Altered Commit: swh:1:rev:a1b2c3d4e5f6789...
File Path: assets/private/id_rsa
Status: Removed
```

[...]

- 
- 1 Introduction
 - 2 Software Heritage
 - 3 Detecting Git history alterations
 - 4 Detecting Git tag alterations**
 - 5 Detecting one-day vulnerabilities across forks
 - 6 Conclusion

Mutating the "immutable": Git tag alterations

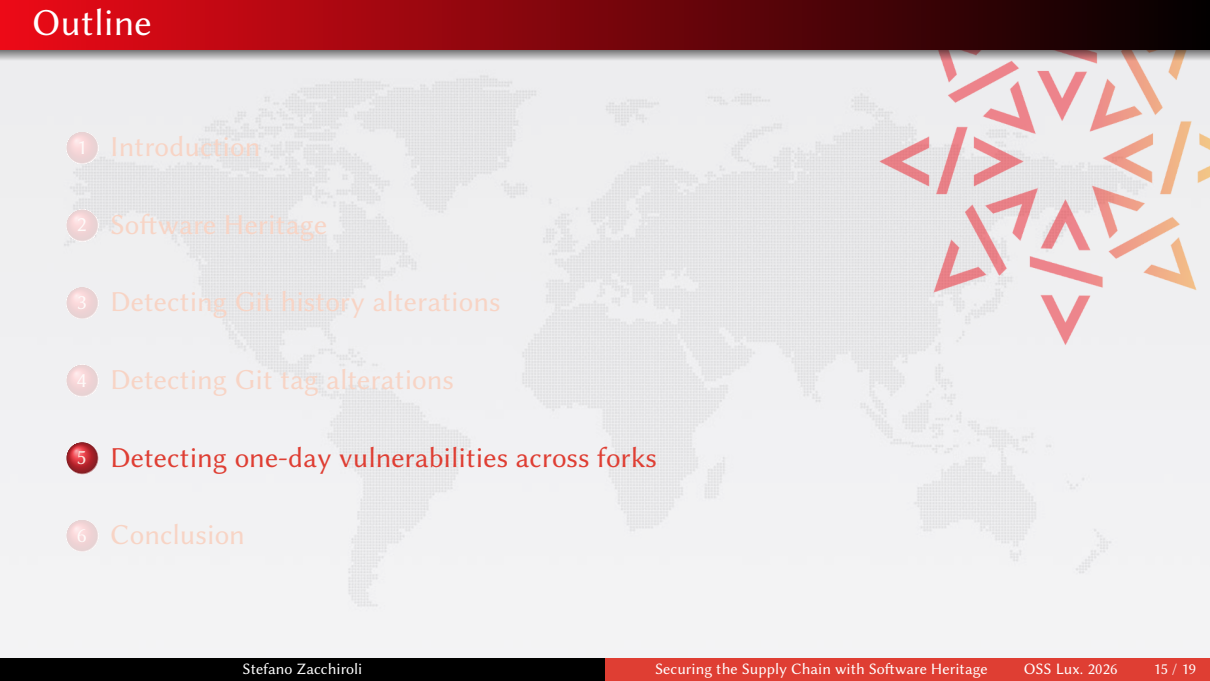
- Tags are used as immutable release anchors, but Git lets you **delete** or **move** them
- SWH snapshots of 328.4 M repos: **10.2M tag alterations** in **189k repos**; of 626k *moves*, **68% point to different code**
- In the wild: **346 tags moved** to a malicious commit (fork hit by the `tj-actions` attack); **7 Nixpkgs packages** fail to rebuild without cache
- Recommendations: **Pin dependencies to hashes/SWHIDs**, not tags; forges: expose tag audit logs



Solal Rapaport, Laurent Pautet, Samuel Tardieu, Stefano Zacchiroli, Théo Zimmermann

Mutating the "Immutable": A Large-Scale Study of Git Tag Alterations

ACM REP 2026 <https://doi.org/10.1145/3820002.3828585>

- 
- 1 Introduction
 - 2 Software Heritage
 - 3 Detecting Git history alterations
 - 4 Detecting Git tag alterations
 - 5 Detecting one-day vulnerabilities across forks
 - 6 Conclusion

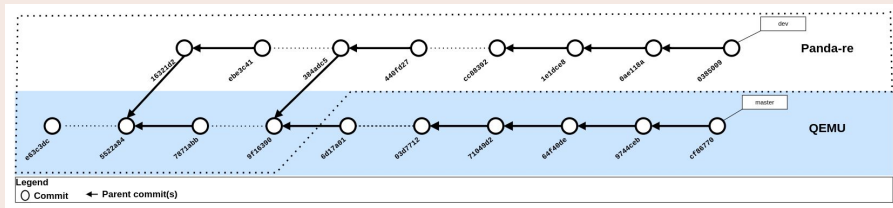
One-day vulnerabilities in open source

One-day vulnerabilities

- Def.: vulnerabilities that are **publicly known, but not fixed yet** in software you use.
- Challenge: **identify them quickly and exhaustively**, then apply countermeasures.
- Many tools available to detect one-day vulnerabilities via declared dependencies.

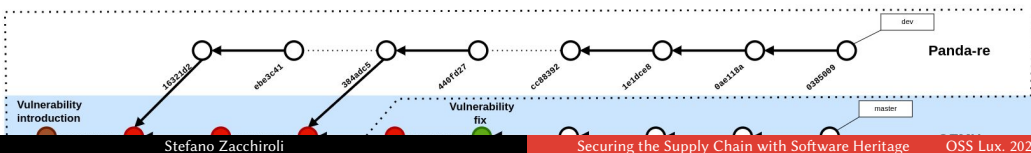
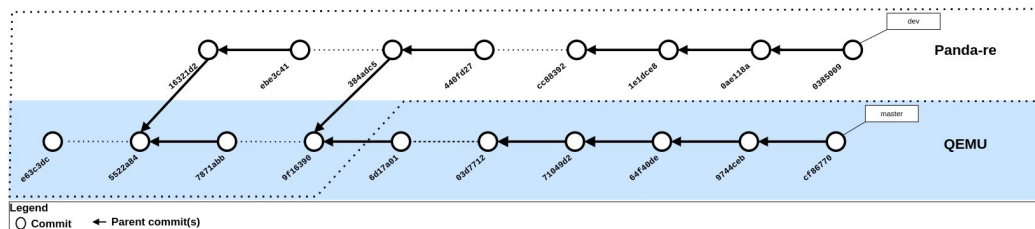
Reusing OSS via forks

...but OSS is also **reused via forking**: (1) start from existing OSS (e.g., the QEMU emulator), (2) create your own (e.g., Panda-re), (3) periodically integrate changes.



Vulnerability propagation through forks

- Any change to a piece of software (*commit*) can **introduce a new vulnerability**.
- Or it can **fix an existing vulnerability**.
- What happens if a project is forked **between introduction and fix** of a vulnerability?
- It inherits the vulnerability, ... until the change with the fix is integrated.



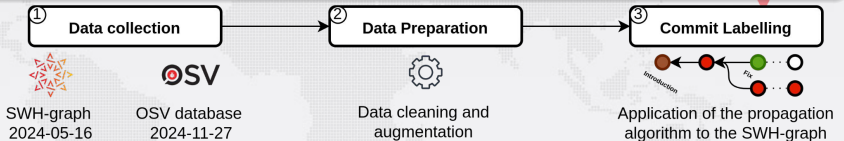
swh-vuln: chasing one-day vulnerabilities across forks... at SWH scale

Approach

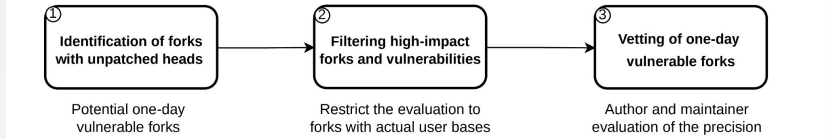
Start from a **public DB of vuln. introduced/fixed** in public commits (e.g., [OSV.dev](https://osv.dev)).

- 1 "Color" the entire graph of public code development history **with vulnerability info.**
 - Software Heritage is the only place where this can be done at the scale of all forks, across all public code, independently of specific development platforms.
- 2 **Inform maintainers** of vulnerable forks. (After validation.)

Phase 1: color the commit graph with vuln. info



Phase 2: find and vet vulnerable forks



Results (highlights)

- Starting from 7162 repos in OSV, we identified **1.7 M forks potentially vulnerable** in their most recent commit.
 - 86.3 M potentially vulnerable commits specific to forks, not findable with current tools.
- We manually verified 152 cases, confirming **135 high-severity vulnerabilities in popular forks**; 9 were further confirmed by maintainers.



panda-re/panda

Fork of QEMU, system emulator and virtualizer

CVE-2019-13164



go-nv/goenv

Fork of pyenv, for Go

CVE-2022-35861

XPERIA
Sony Smartphone

sonyxperiadev/kernel

Fork of Linux, for the Xperia dev community

CVE-2021-45485

CVE-2021-4154

bitcoin-sv/bitcoin-sv

Fork of Bitcoin

CVE-2021-37492



Romain Lefeuvre, Charly Reux, Stefano Zacchiroli, Olivier Barais, Benoit Combemale

Chasing One-day Vulnerabilities Across Open Source Forks

SCORED 2026. Preprint: <https://arxiv.org/abs/2511.05097>

- 
- 1 Introduction
 - 2 Software Heritage
 - 3 Detecting Git history alterations
 - 4 Detecting Git tag alterations
 - 5 Detecting one-day vulnerabilities across forks
 - 6 Conclusion

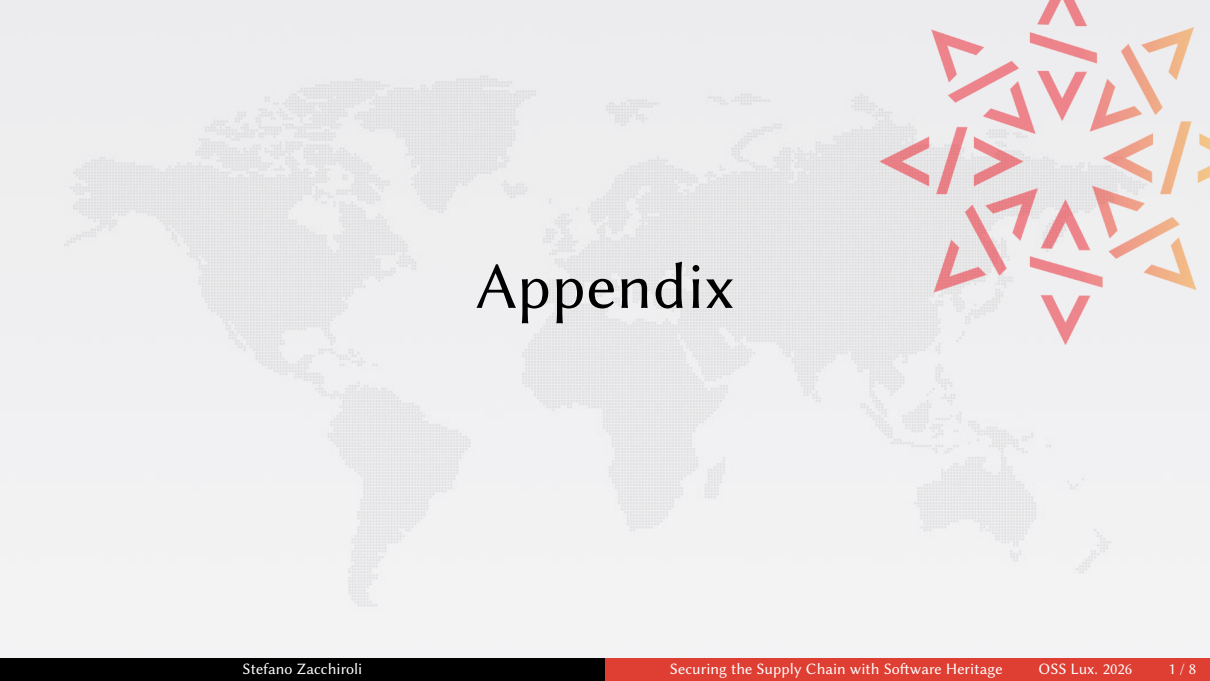
Conclusion

- Software Heritage preserves *all* public source code **and its full history**, across forges, forks, and rewrites
- Forges show *the present*, the archive keeps *the past* → **supply chain auditing**: 1.22 M rewritten histories, 10.2 M tag alterations, 1.7 M potentially vulnerable forks
- **Archive what you depend on**: otherwise it can silently change or vanish
- Pin dependencies to **intrinsic identifiers** (commit hashes, SWHIDs), not tags

Support Software Heritage!

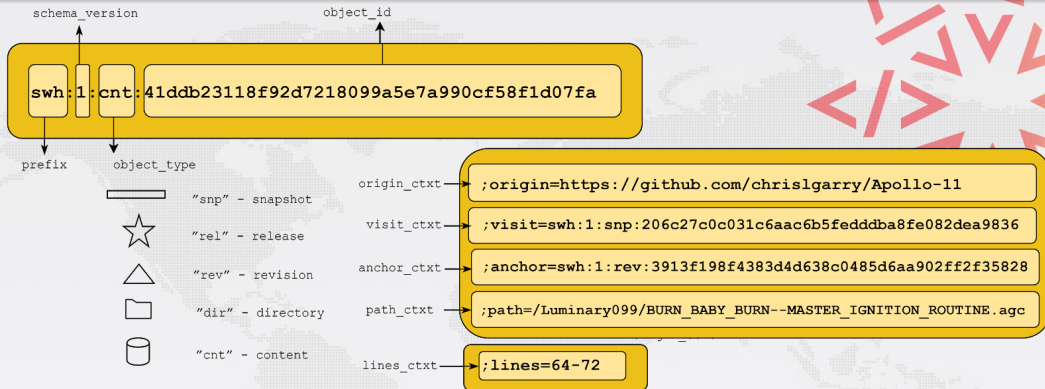
- Open, non-profit infrastructure: **usable by everyone**, not only big players
- Your supply chain depends on open source: **become a sponsor** and help secure it!
→ softwareheritage.org/support/sponsors

softwareheritage.org — Stefano Zacchioli / stefano.zacchioli@telecom-paris.fr / [@zacchiro@mastodon.xyz](https://mstdn.social/@zacchiro)



Appendix

- 
-  Solal Rapaport, Laurent Pautet, Samuel Tardieu, Stefano Zacchiroli
Altered Histories in Version Control System Repositories: Evidence from the Trenches
ASE 2025. <https://arxiv.org/abs/2509.09294>
 -  Solal Rapaport, Laurent Pautet, Samuel Tardieu, Stefano Zacchiroli, Théo Zimmermann
Mutating the “Immutable”: A Large-Scale Study of Git Tag Alterations
ACM REP 2026. <https://doi.org/10.1145/3820002.3828585>
 -  Romain Lefeuvre, Charly Reux, Stefano Zacchiroli, Olivier Barais, Benoit Combemale
Chasing One-day Vulnerabilities Across Open Source Forks
SCORED 2026. <https://arxiv.org/abs/2511.05097>



An emerging standard

- Adoption: [SPDX 2.2](#), IANA-registered "swh:" URI prefix, WikiData [P6138](#), ...
- Breaking news: [standard ISO/IEC 18670:2025](#)

Examples

- [Apollo 11 AGC excerpt](#)
- [Quake III rsqrt](#)

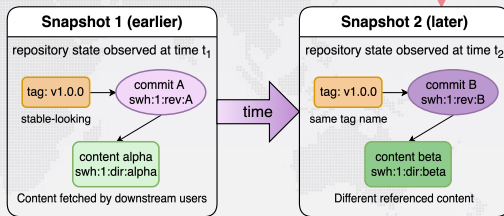
Mutating the "immutable": Git tag alterations

Tags as release anchors

- Releases are often *referenced by tag* (v1.2.3): Go modules, Git dependencies in npm, Cargo, pip, GitHub Actions, CI pipelines, package recipes, ...
- Commonly assumed to be **immutable**

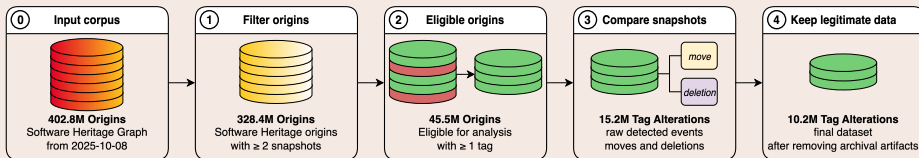
...but Git tags are *mutable*

- Tags can be **deleted**, or **moved** (force-pushed) to point to a different commit
- Git docs: "people **MUST** be able to trust their tag-names"
- Forges keep **no record** of what a tag used to point to



A large-scale analysis of Git tag alterations

- 1 Retrieve from Software Heritage repos archived ≥ 2 times (328.4 M, 2016–2025), keep Git ones with tags (45.5 M)
- 2 Compare **consecutive snapshots**: did a tag disappear, or point to a different target (SWHID)?
- 3 Classify: **deletions** vs. **moves** (different code, commit metadata, or release)



Solal Rapaport, Laurent Pautet, Samuel Tardieu, Stefano Zacchiroli, Théo Zimmermann

Mutating the “Immutable”: A Large-Scale Study of Git Tag Alterations

ACM REP 2026 <https://doi.org/10.1145/3820002.3828585>

Key findings on tag alterations

How often?

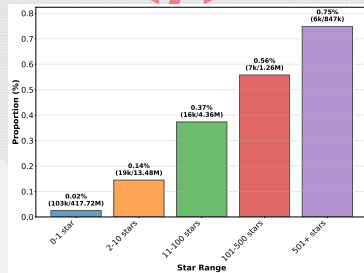
- 10.2 M tag alterations in 189 k repos, growing steadily

How?

- Deletions (93.8%) → availability: builds break
- Moves (6.2%, ~626k): same name, new target; 68% point to different code → integrity!

Where?

- 74% of alterations are in repos with 0–1 GitHub stars, but...
- ...the more popular a repository, the more likely it has altered tags



repositories with ≥ 1 alteration, by star count

Tag alterations in the wild: attacks and broken builds

Supply-chain attack: tj-actions/changed-files (March 2025)

- Widely used GitHub Action; attackers moved its version tags to a malicious commit
- Payload dumped CI runner memory (**secrets, tokens, credentials**) into workflow logs
- Every workflow pinned to a version tag (@v39, @v47, ...) ran it on the next run
- Captured by Software Heritage in a fork (zendesk/changed-files): **346 tags moved** to the same malicious commit on 14 March 2025

Broken builds: Nixpkgs cross-analysis

- **32 packages** reference altered tags; **7 fail to rebuild** without the binary cache (4: tag deleted, 3: hash mismatch)

```
$ nix build github:NixOS/nixpkgs/08ffd21[...]#libv3270.src --rebuild
source> trying https://[...]/refs/tags/5.5.0.tar.gz
error: hash mismatch in fixed-output derivation '/nix/[...]':
    specified: sha256-Cn/to1/[...]=
    got:      sha256-tW3CPga[...]=
```

What to do about tag alterations — recommendations

Build tools & package managers

- Tags are fine as *human-friendly names*, but **pin dependencies to cryptographic identifiers** (commit hashes, SWHIDs)
- Record both: tag name + hash it resolved to (as, e.g., Nixpkgs and Go already do)

Forges

- Expose **audit logs of tag mutations** (creation, move, deletion)
- GitHub immutable releases (Aug 2025): good step, but opt-in, GitHub-only, and for releases rather than Git tags

Everyone

- **Monitor tag alterations** as a standard supply-chain security practice
- Software Heritage keeps every snapshot it has seen → past tag states remain auditable