Conclusion to mapping known vulnerabilities to SWH

Val Lorentz (@vlorentz)

Software Heritage

2026-10-13 - SWHSec Plenary



Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Outline



1. Reminders from last year
2. Finding introducing commits with SZZ
3. Implementation on swh-graph

osv.dev database

- aggregates other databases, mostly in OSV format themselves
- each report contains a bunch of ids and a description
- `affected` software packages
  - 38k unique (1.6M total) are in SWH
  - 55k unique (1.4M) are in ecosystems not supported by SWH. Mostly RHEL (Red Hat) and Ubuntu Pro
  - 800 unique (56k) are missing from SWH
  - 7k unique are bogus

Example of `affected`

```
{    "affected": [{
         "package": {
             "ecosystem": "PyPI",
             "name": "pyexample"
         },
         "ranges": [{
             "type": "GIT",
             "repo": "https://github.com/...",
             "events": [ ... ]
         }]
     }]
}
```



Versions in `affected` field

- either a range of version numbers, or of Git commits
- Four types of `events`:
  - `introduced`
  - `fixed`
  - `last_affected`
  - `limit`, which is like `last_affected` but restricts to a path from `introduced`

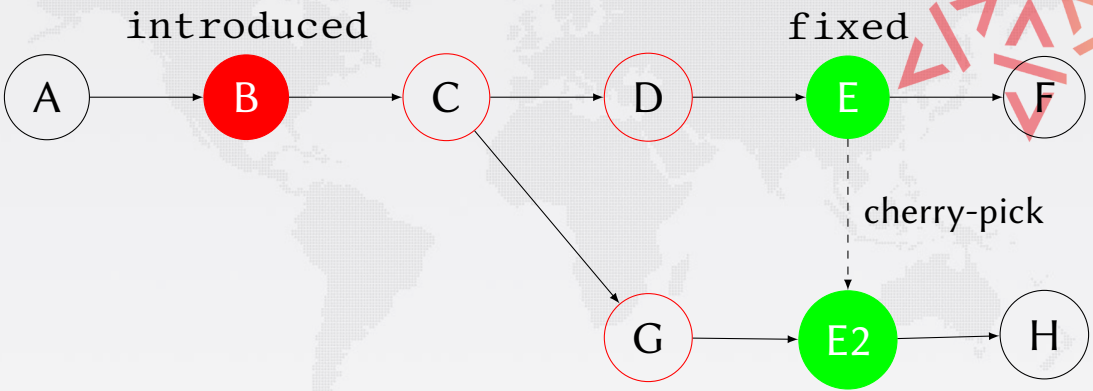
Example of `events`

```
{
    "events": [
        {"introduced": "1.0"},
        {"fixed":      "1.2"},
        {"introduced": "2.0"},
        {"fixed":      "2.1"},
    ]
}
```

Limitations

- `introduced` is often `"0"`, meaning "assume it's vulnerable since the very first version"
- `fixed` is not exhaustive
- `last_affected` and `limit` are lies

fixed



Expanding `fixed` with cherry-picks

- It's not exhaustive because it's missing cherry-picks
- Cherry-pick detection based on commit message.
  - 287M are not deemed to be cherry-picks
  - 6.3M have at least one valid `"cherry picked from commit"` stanza (300k have at least two). Out of these:
    - 0.4M reference unknown commits and don't have a repository URL
    - 6.4M reference known commits

Outline



1. Reminders from last year
2. Finding introducing commits with SZZ
3. Implementation on swh-graph

SZZ history



- SZZ algorithm from MSR2005
- further refinements over the years
- V-SZZ in ICSE2022

SZZ example

1	2	3	fix
line1	line1	line1	line1
line2	bad	bad	fixed
line3	line3	line3	line3
line4	line4	line4b	line4b
line5	line5	line5	line5

V-SZZ example (edit)

1	2	3	4	fix
line1	line1	line1	line1	line1
line2	bad	bad	baaad	fixed
line3	line3	line3	line3	line3
line4	line4	line4b	line4b	line4b
line5	line5	line5	line5	line5

V-SZZ example (move)

1	2	3	4	fix
line1	line1	line1	line1	line1
line2	bad	bad	line4	line4
line3	line3	line3b	line3b	line3b
line4	line4	line4	bad	fixed
line5	line5	line5	line5	line5

Outline



1. Reminders from last year
2. Finding introducing commits with SZZ
3. Implementation on swh-graph

Database

- swh-graph for commit graph and directory trees
- RocksDB cache for all contents traversed by V-SZZ:
- 5.5M unique files, 285GB uncompressed, 63GB compressed

Results

- About a month to download all needed contents
- Takes 60 CPU-hours to run on all vulnerabilities
- Not really usable:
  - Advertised `fixed` commits are often a version bump, or the last commit before a release
  - Even when they are correct, SZZ often blames refactorings
- Code at https://gitlab.softwareheritage.org/swh/devel/swh-osv