Case Study of Software Engineering Research

What We Talk About, When We Talk About Reproducible Builds

Zhilei Ren

Dalian University of Technology

September 29, 2025

Checksum is important for software security

- Checksum provides a linkage between user downloaded software and the one provided by the repositories
- Build environments can be compromised!

XcodeGhost Case

- A counterfeit version of Xcode infected 4,000 iOS apps in 2015
- More than 200 enterprises influenced
- Injecting malicious code during compiling time
- `https://www.fireeye.com/blog/executive-perspective/2015/09/protecting_our_custo.html`

For open-source software, will it be easier?

- Solution: verify compiled binaries under diverse environments
- Attacks may elude detection due to unexpected factors
- There is still gap between source and binary

Yet another case of unreproducible build

- libical-dev 1.0-1.1 in Debian
- Severity classified as "Release Critical"
- Packages depending on libical-dev may break when rebuilt
- Unreproducible due to non-deterministic hash table traversal

Now, many distros validate package reproducibility!

- Many open-source distributions have initiated validation processes
- Debian, Guix, ArchLinux, and Bitcoin
- As of May 2018, over 90% of Debian's packages can be reproducibly built

The unreproducible build problem

Definition

A build is reproducible if given the same source code, build environment and build instructions, any party can recreate bit-by-bit identical copies of all specified artifacts.

- Manual work! Could it be automated?

Highlights: RepLoc

- An automated localization approach
- 79.28% top-10 accuracy ratio over 671 packages
- We fixed 6 packages from Debian and Guix
- 4 are accepted by the open-source community
- Dataset publicly available: `https://reploc.bitbucket.io`

Challenges

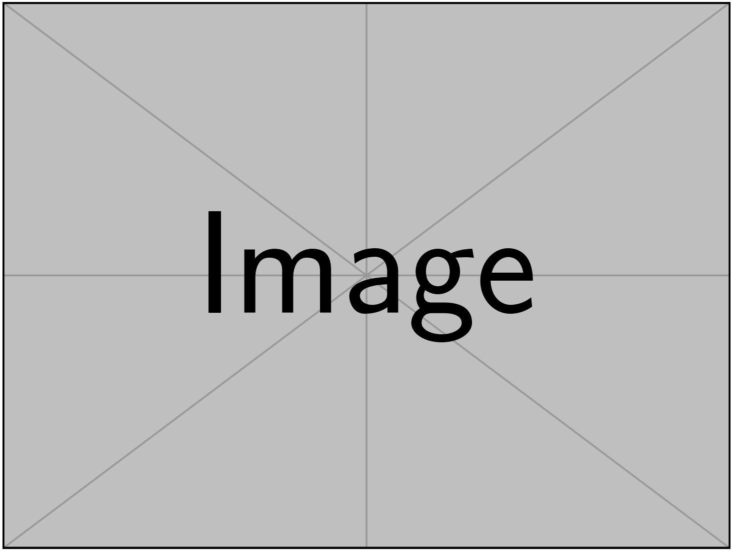
Challenge 1: Insufficient information

- Information may not be sufficient for linking diff log with problematic files
- No enough bug reports/test cases
- Diff logs are not always well informative

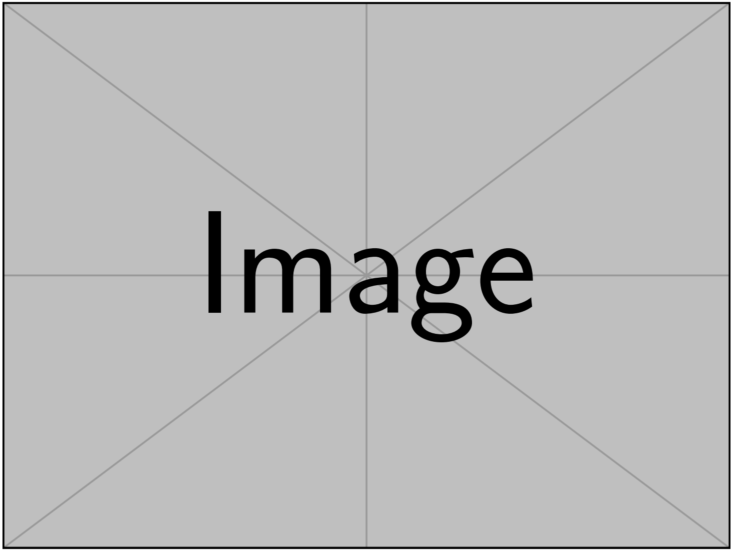
Challenge 2: Diverse causes

- Debian rules (29.82%)
- Auxiliary files (17.21%)
- Scripts (14.60%)
- Makefiles (11.68%)
- C/C++ files (5.94%)

The RepLoc framework



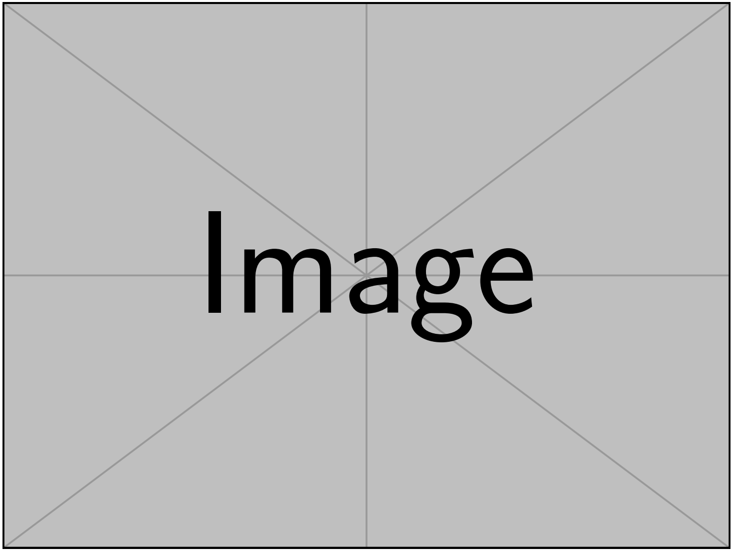
Query augmentation



Heuristic filtering

- Manually learn notes from Debian documentation
- Capture issues as 14 Perl Compatible Regular Expression (PCRE) rules

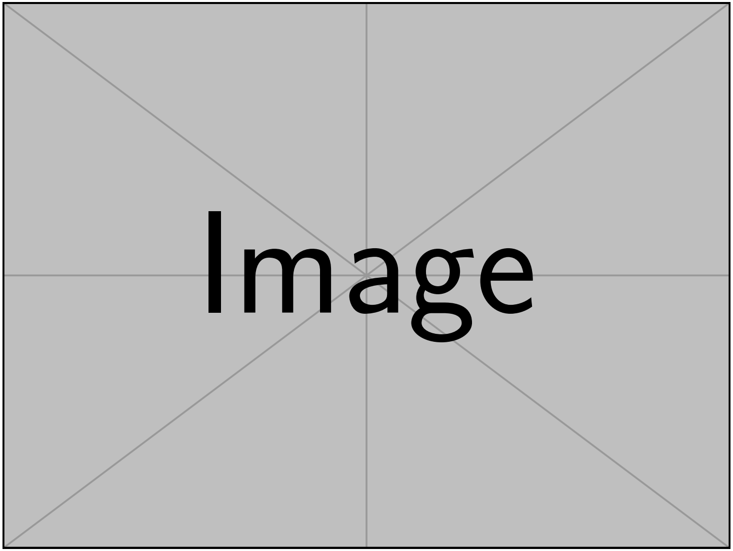
File ranking



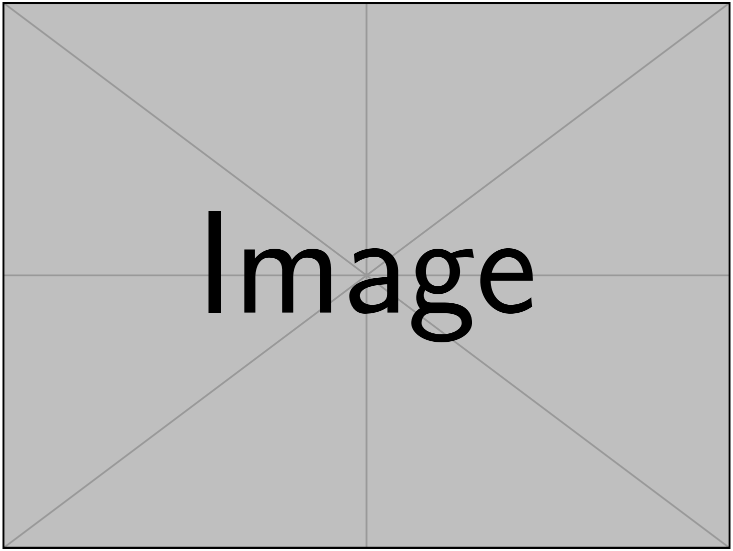
Empirical results: Experimental environment

- Intel Core i7 4.20 GHz CPU
- 16 GB RAM
- GNU/Linux kernel 4.9.0
- Perl 5.24, Java 1.8
- 671 packages from Debian's BTS
- 4 categories: Timestamps, Locale, Randomness, File-ordering

RQ1: Is RepLoc sensitive to weighting parameter ?



RQ2: How effective is RepLoc?



RQ3: How efficient is RepLoc?

- Median execution time: 5.14 seconds

RQ4: Is RepLoc helpful in localizing unfixed packages?

- Fixed 6 packages from Debian and Guix:
    - regina-rexx
    - manpages-tr
    - fonts-uralic
    - skalibs
    - djvulibre
    - libjpeg-turbo
- 4 patches accepted (1 for Debian, 3 for Guix)

A case of fixed issue for djvulibre for Guix

- `https://debbugs.gnu.org/cgi/bugreport.cgi?bug=2801`
- Patch modifies djvu.scm to add reproducible phase
- Guix maintainer's response: pushed with cosmetic changes
- Issue with .svgz files not handled by new phase

Conclusions and future work

Contributions

- First work to address localization task for unreproducible builds
- Effective framework integrating heuristic filtering and query augmentation
- Fixed six previously unfixed packages
- Dataset available at `https://reploc.bitbucket.io`

Future work

- More accurate approaches using static analysis or dynamic file tracking
- Generalize knowledge to other repositories
- Automated fixing of unreproducible issues

Acknowledgements

- Debian reproducible builds team: Holger Levsen, Chris Lamb, Ximin Luo, ...
- Guix distribution: Ludovic Courtès, ...
- OSCAR Lab: `http://oscar-lab.org`

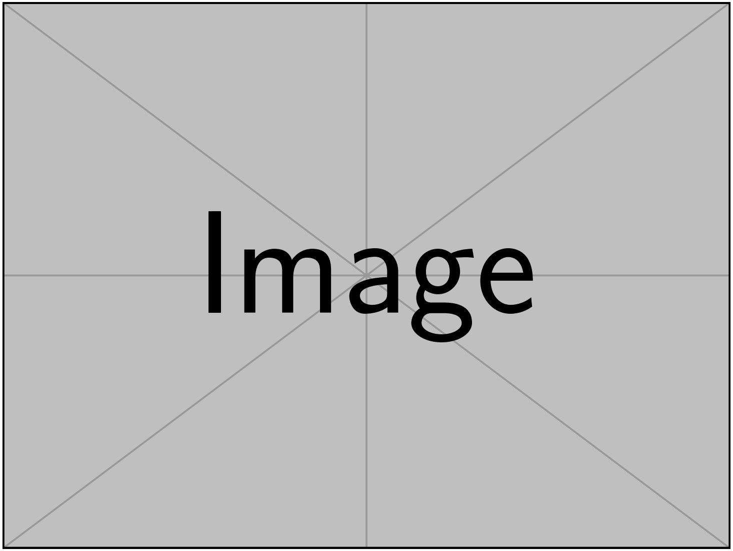
Ideas currently working on

- Why RepLoc works:
  - Heuristic rules for frequent patterns (static)
  - Build log for inconsistent-files-related commands (dynamic)
- Why RepLoc is not good enough:
  - Heuristic rules treat all files uniformly
  - Not all build commands captured by build logs

System call interposition approach

- Improve localization by analyzing system calls during build process
- Use strace to monitor interactions between processes and kernel
- Track dependency types: read/write operations, rename operations

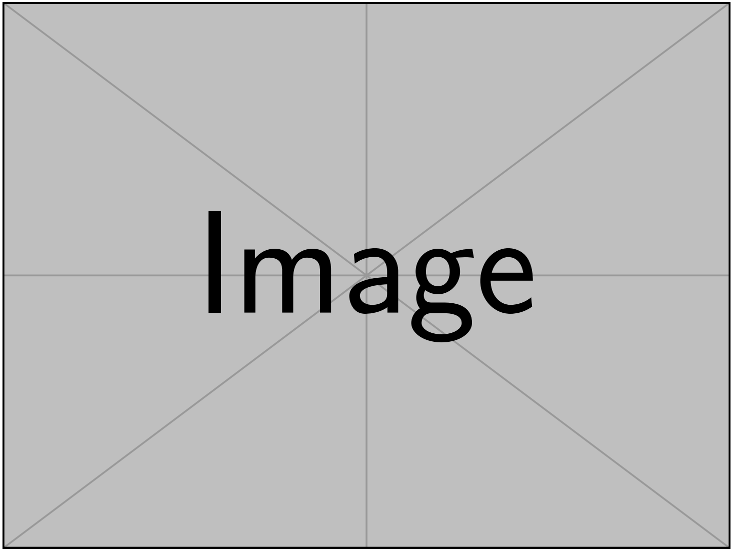
Process tree and dependency graph



System call trace example

```
2608 2607 2608 1532307317.518497 execve("/bin/date", ["date"], [...])
2608 2607 2608 1532307317.526037 write(1<pipe:[55019]>, "2018...", 43) = 4
2607 2603 2607 1532307317.515482 read(3<pipe:[55019]>, "2018...", 128) = 4
2607 2603 2607 1532307317.535603 write(1</home/.../b.txt>, "bb2018...", 45
```

REPTRACE Framework



Dependency Graph Generation and Augmentation

- Difference-induced dependency (DID)
- Runtime-value-induced dependency (RID)
- Filter noisy dependencies
- Establish dependencies based on text similarity

Graph-Traversal-based Causality Analysis

- Root-cause localization starting from inconsistent artifacts
- File-level localization for scripts and build commands
- Use CLOEXEC flags to classify files

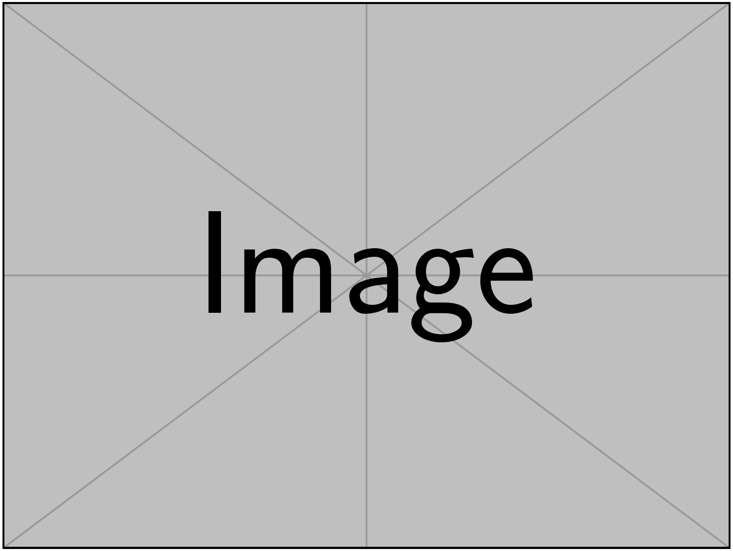
Evaluation Results

- RQ1: Command-level localization effective (A@1 0.6611, A@10 0.9000)
- RQ2: DID reduces search scope, RID adds acceptable edges
- RQ3: Not sensitive to parameter, best around [0.30; 0.70]
- RQ4: Higher accuracy than RepLoc (Top-1 Accuracy 0.667)

RepFix: Automated Patching

- First study to generate patches for unreproducible builds automatically
- Tracing-based fine-grained localization
- History-based patch generation
- Patch validation criteria

RepFix Framework



RepFix Evaluation

- RQ1: Fixed 64 out of 116 packages
- RQ2: Fine-grained localization and token-based patch generation effective
- RQ3: Validation most time-consuming (avg 364.40s)
- RQ4: Effectively solves real-world unreproducible build issues

Conclusions

- REPTRACE: Effective framework for root cause identification
- RepFix: Automated patching for unreproducible builds
- Both show promising results on real-world packages
- Future work: More accurate dependency identification, automatic patching

Thank you!

Q&A