Program Analysis for Safe and Secure Software Evolution

Cristian Cadar

Funded by

Imperial Global Singapore
Singapore, 28 November 2024

By Fuzzypiggy, Wikipedia

@ Imperial College London

Imperial College London

SOFTWARE RELIABILITY GROUP

http://srg.doc.ic.ac.uk

Current and recent members

Cristian Cadar

Anastasios Andronidis

Frank Busse

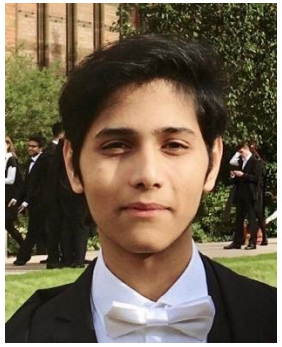
Manuel Carrasco

Karine Even-Mendoza

Martin Nowack

Jordy Ruiz

Daniel Schemmel



Arindam Sharma

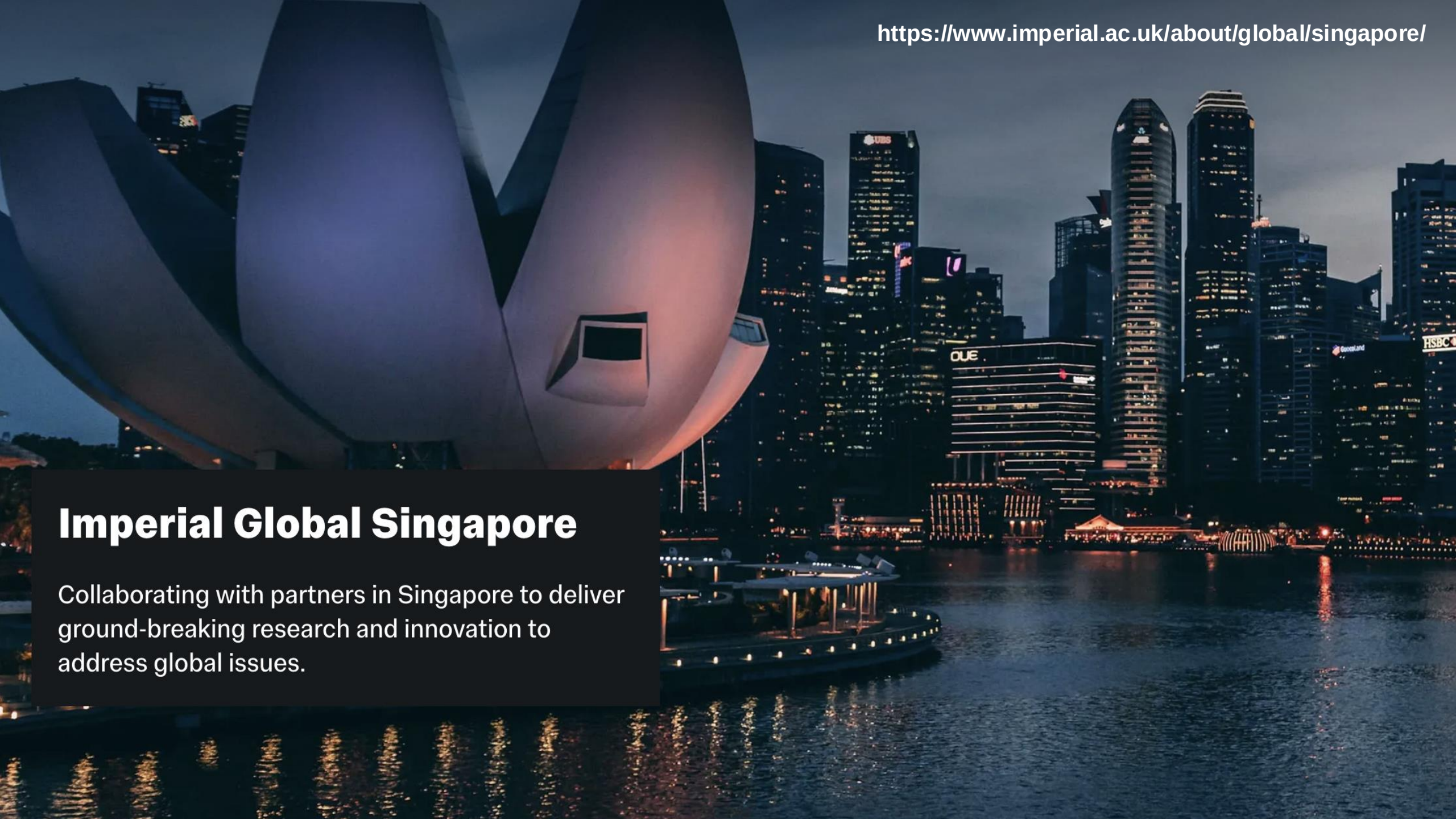
Bachir Bendrissou

Ahmed Zaki

@ Cristian Cadar

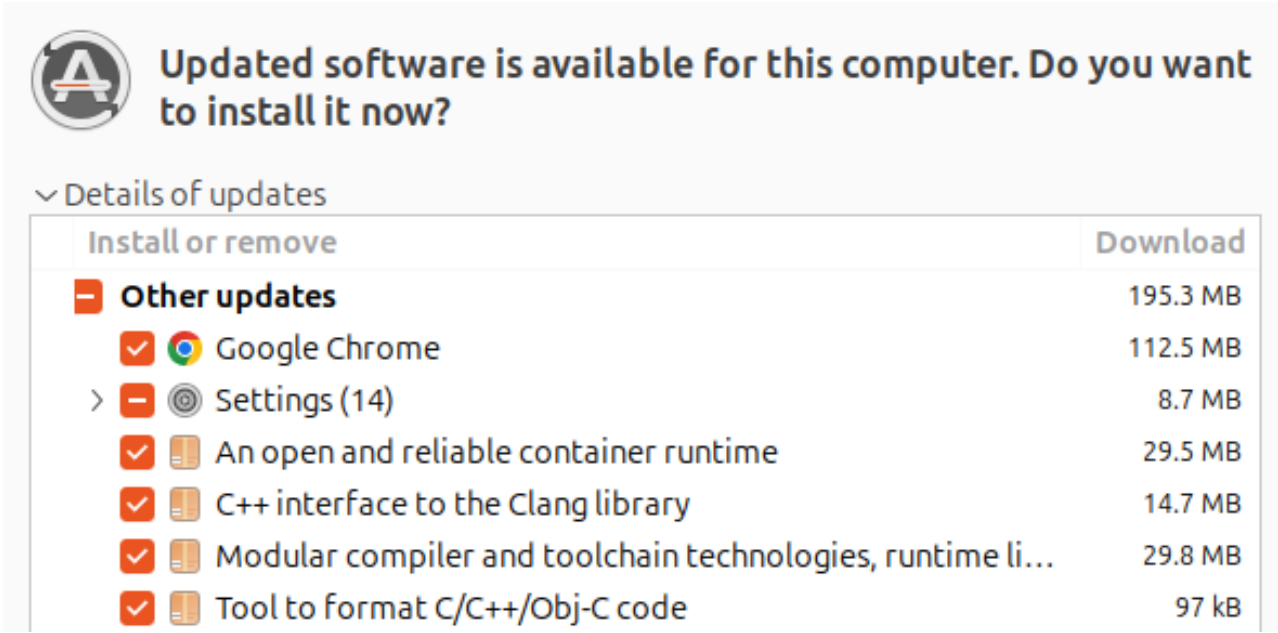
Jiangxiang, Wikipedia

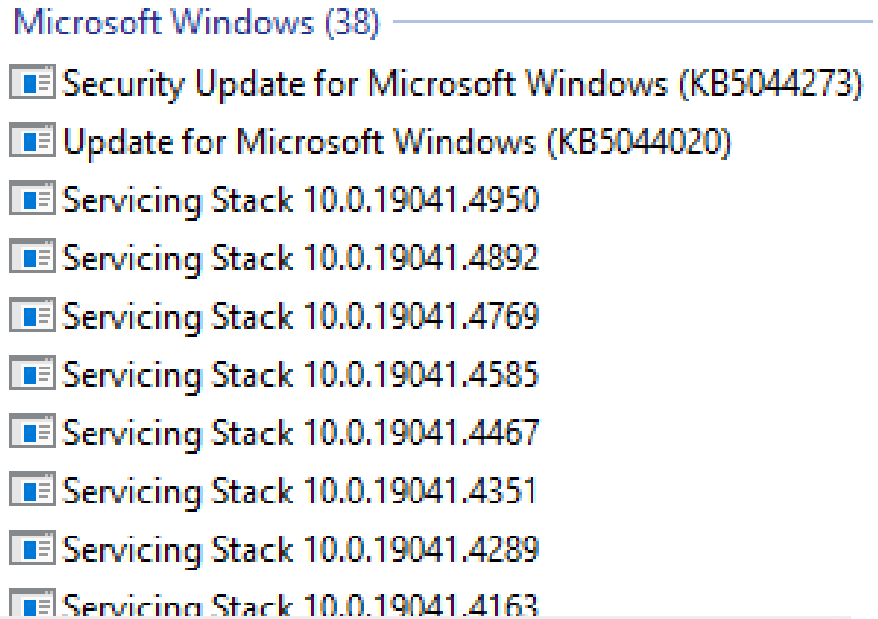
https://www.imperial.ac.uk/about/global/singapore/

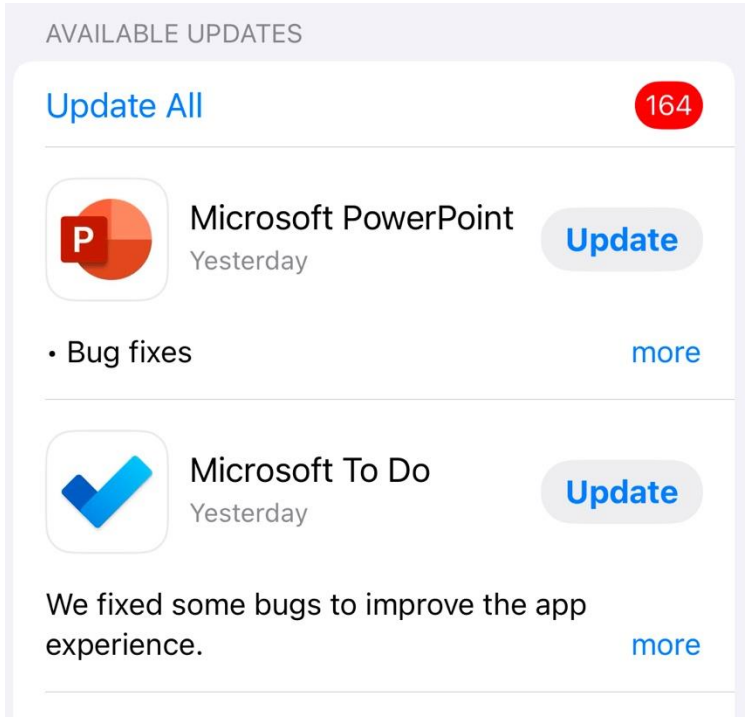


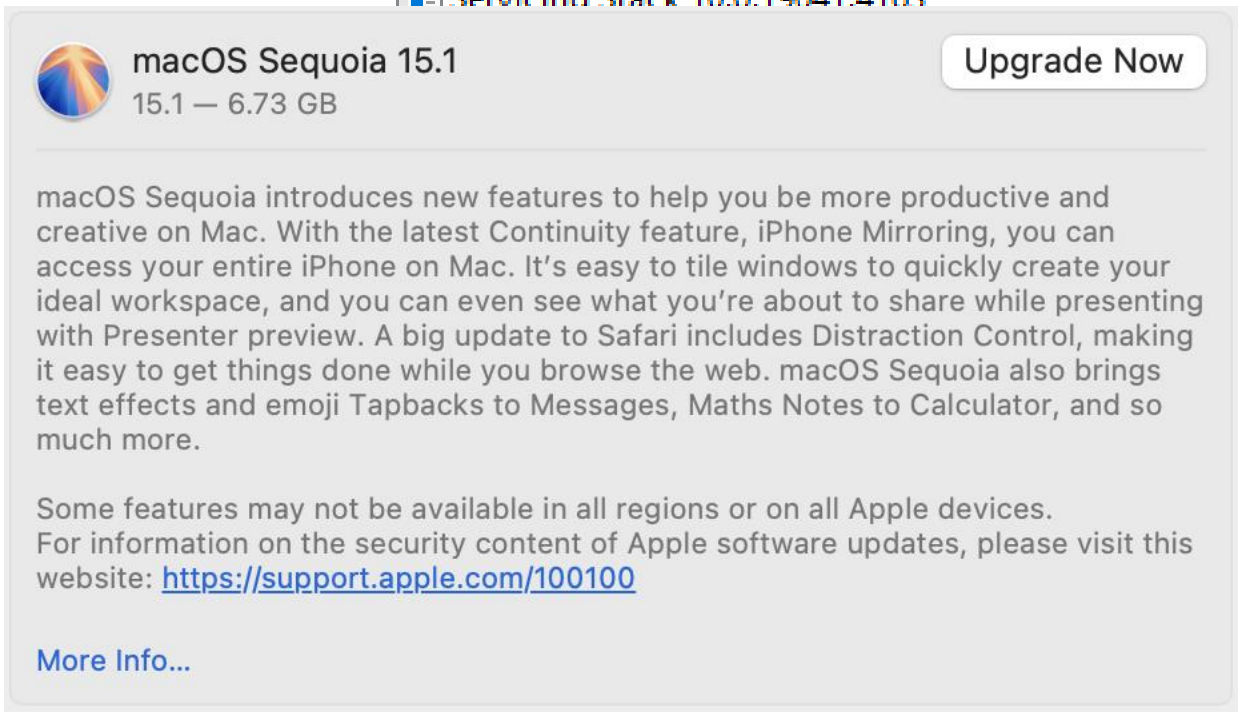
Imperial Global Singapore

Collaborating with partners in Singapore to deliver ground-breaking research and innovation to address global issues.


Updated software is available for this computer. Do you want to install it now?
Details of updates
Install or remove
Download
Other updates
195.3 MB
Google Chrome
112.5 MB
Settings (14)
8.7 MB
An open and reliable container runtime
29.5 MB
C++ interface to the Clang library
14.7 MB
Modular compiler and toolchain technologies, runtime li…
29.8 MB
Tool to format C/C++/Obj-C code
97 kB


Microsoft Windows (38)
Security Update for Microsoft Windows (KB5044273)
Update for Microsoft Windows (KB5044020)
Servicing Stack 10.0.19041.4950
Servicing Stack 10.0.19041.4892
Servicing Stack 10.0.19041.4769
Servicing Stack 10.0.19041.4585
Servicing Stack 10.0.19041.4467
Servicing Stack 10.0.19041.4351
Servicing Stack 10.0.19041.4289
Servicing Stack 10.0.19041.4163


AVAILABLE UPDATES
Update All
164
Microsoft PowerPoint
Yesterday
Update
• Bug fixes
more
Microsoft To Do
Yesterday
Update
We fixed some bugs to improve the app experience.
more


macOS Sequoia 15.1
15.1 — 6.73 GB
Upgrade Now
macOS Sequoia introduces new features to help you be more productive and creative on Mac. With the latest Continuity feature, iPhone Mirroring, you can access your entire iPhone on Mac. It's easy to tile windows to quickly create your ideal workspace, and you can even see what you're about to share while presenting with Presenter preview. A big update to Safari includes Distraction Control, making it easy to get things done while you browse the web. macOS Sequoia also brings text effects and emoji Tapbacks to Messages, Maths Notes to Calculator, and so much more.
Some features may not be available in all regions or on all Apple devices.
For information on the security content of Apple software updates, please visit this website: https://support.apple.com/100100
More Info...

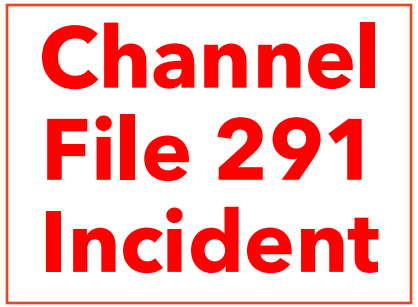
Evolving Software

- Code changes are poorly validated and often introduce bugs & vulnerabilities
- Some with catastrophic impact

Heartbleed (2014)

Shellshock (2014)

Stagefright (2016)



Crowdstrike (2024)

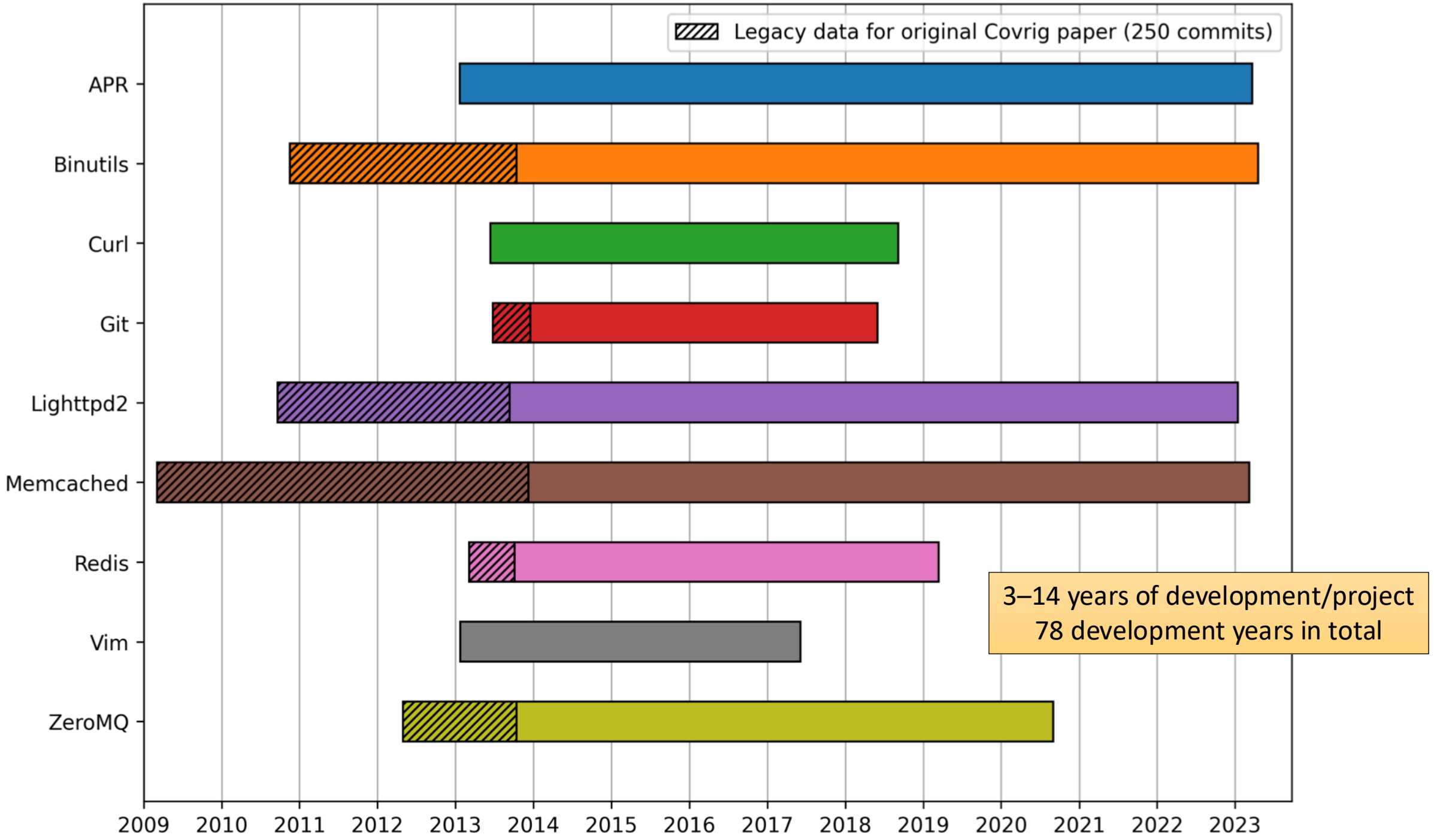
COVRIG: A Framework for the Analysis of Code, Test, and Coverage Evolution in Real Software

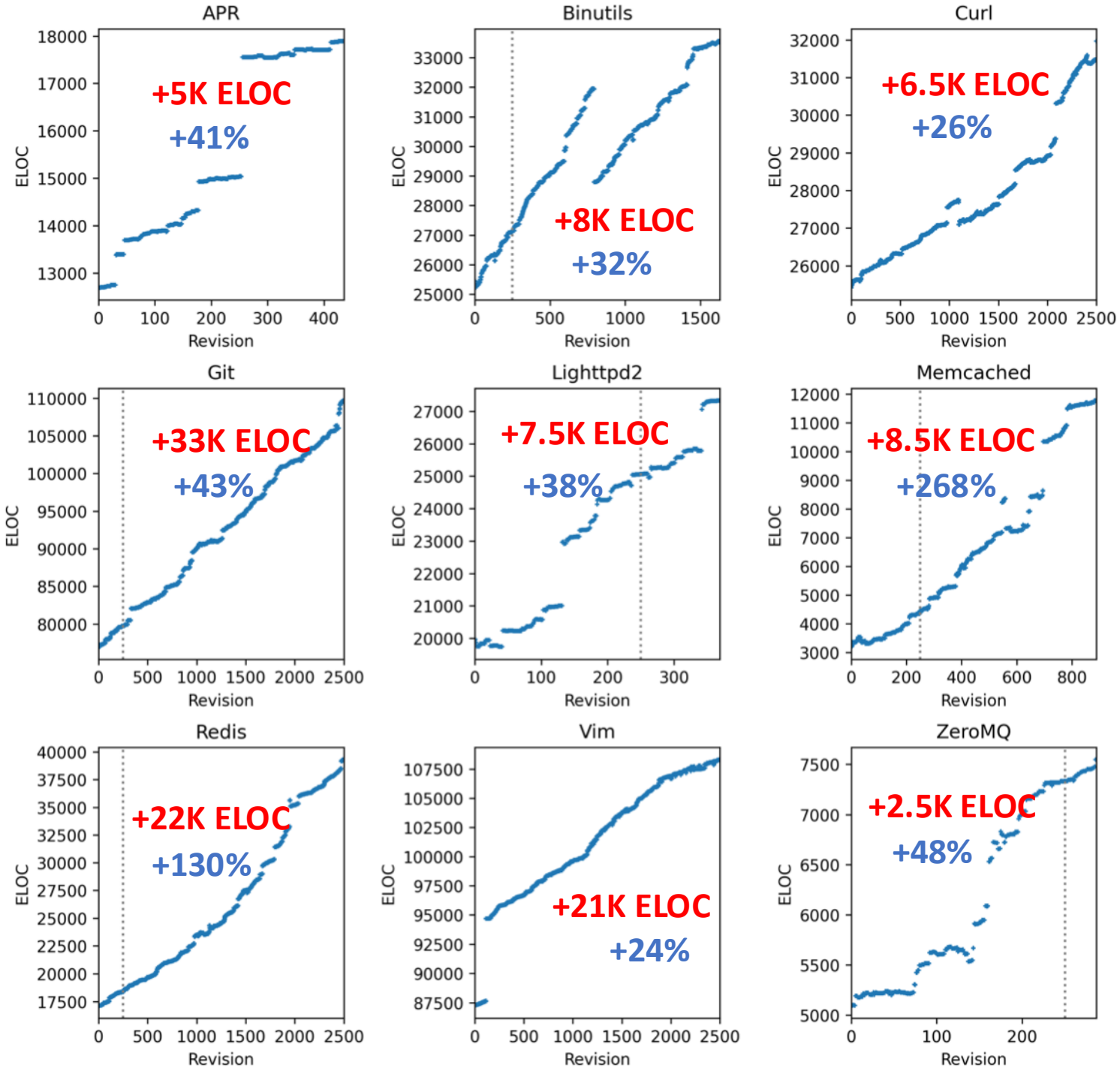
Paul Marinescu, Petr Hosek, Cristian Cadar
Department of Computing
Imperial College London, UK
{p.marinescu,p.hosek,c.cadar}@imperial.ac.uk

ISSTA 2014

- 6 popular open-source systems
- Analysed 250 revisions per app
- Conclusion: LOTS of code added or modified without being tested

A decade later: Have things changed?
Tom Bailey, Cristian Cadar
[To be published]


Legacy data for original Covrig paper (250 commits)
APR
Binutils
Curl
Git
Lighttpd2
Memcached
Redis
Vim
ZeroMQ
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
3–14 years of development/project
78 development years in total

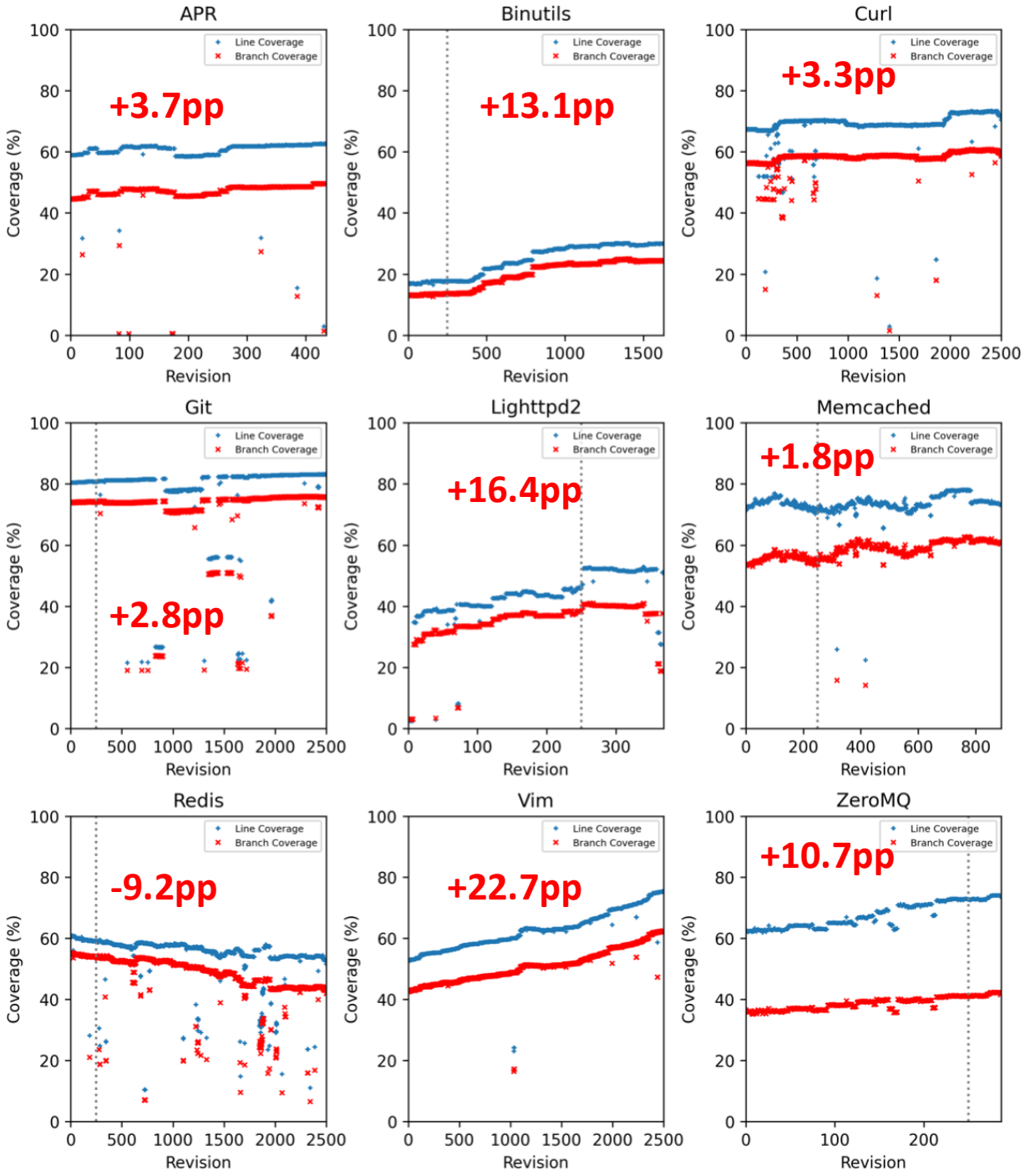


ELOC/time

Code increases of 2.5K – 33K ELOC, 24% – 268%

Coverage Evolution

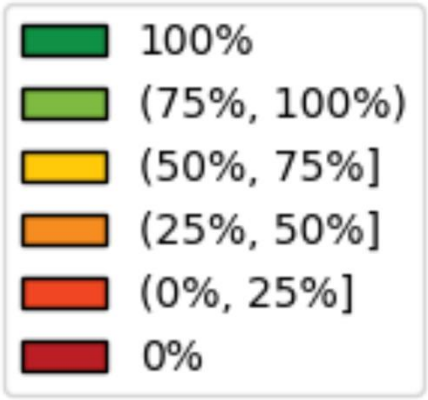
Coverage increases by 2.8 – 22.7pp
It decreases in Redis by 9.2pp

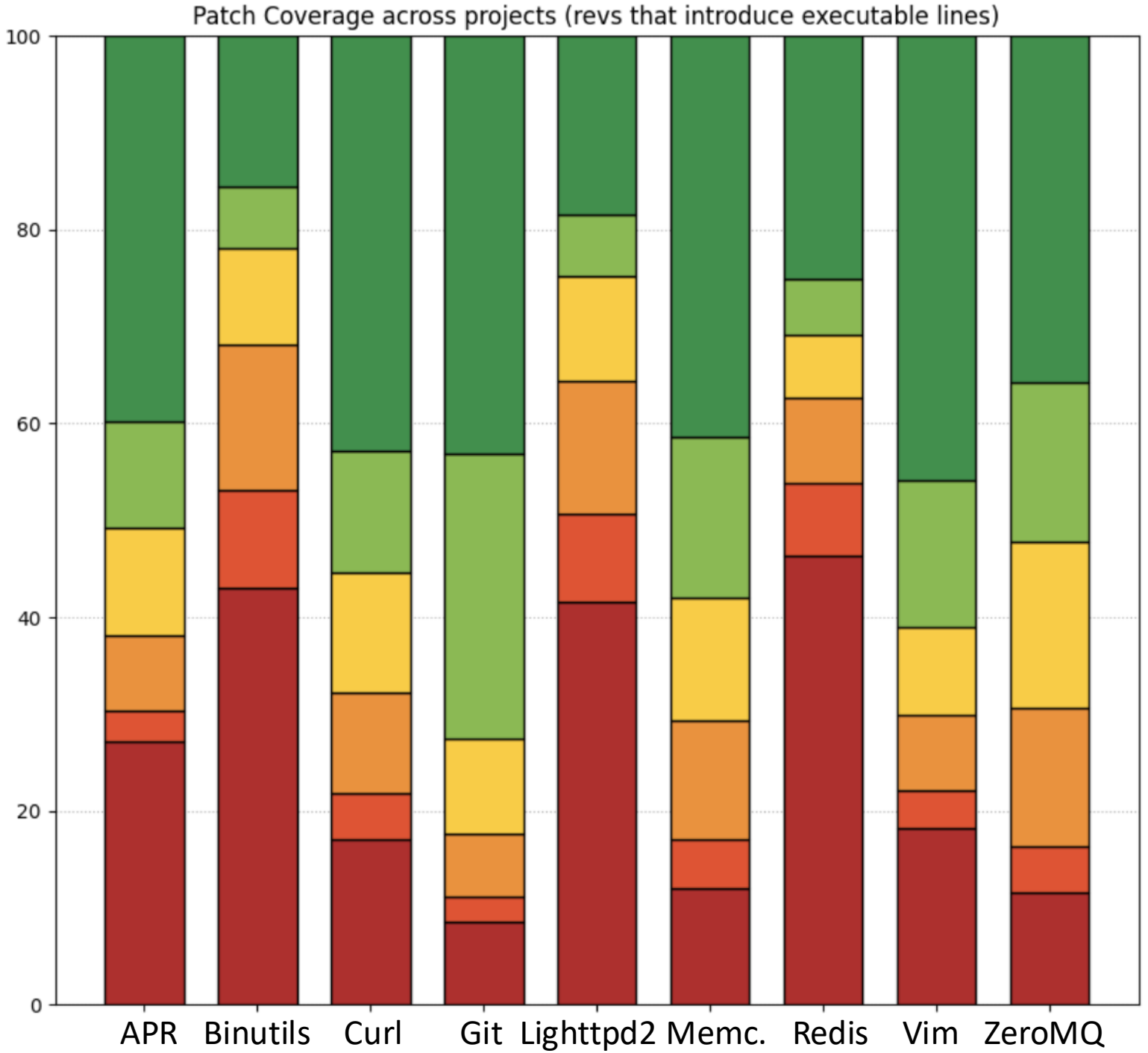


Line coverage

Branch coverage

Patch Coverage





Can Program Analysis Tools Help?

Clang Static Analyzer

cbmc

KLEE

AFL++

EVOSUITE

Grammarinator

ANTLRv4 grammar-based test generator

Clang Static Analyzer

cbmc

AFL++

Designed for whole program testing

EVOSUITE

Csmith

Grammarinator

ANTLRv4 grammar-based test generator

Dafny

Finding and Understanding Bugs in C Compilers

Xuejun Yang Yang Chen Eric Eide John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eeide, regehr}@cs.utah.edu

PLDI 2011

Revolutionary tool for finding compiler bugs, incl. miscompilations

Found hundreds of bugs in compilers such as Clang and GCC

oss-fuzz

Massive deployment of greybox fuzzing

AFL++
libFuzzer
Honggfuzz

- **Fuzzing campaigns take a day (clock time)**
- **Reporting a bug can also take a day**

Trophies

As of August 2023, OSS-Fuzz has helped identify and fix over 10,000 vulnerabilities and 36,000 bugs across 1,000 projects.

Whole-Program Fuzzing
i.e. Fuzzing from Scratch

Expensive and wasteful

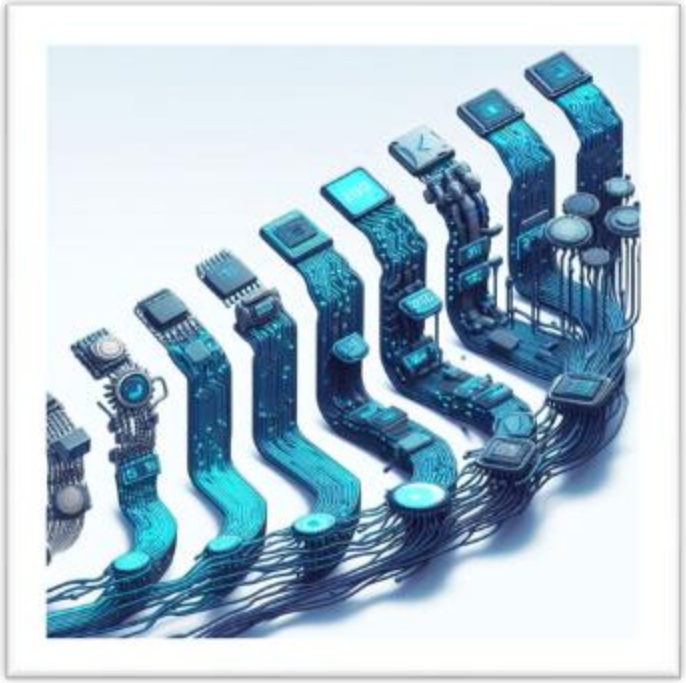
- Lots of **wasteful repetition** across versions
- Same bugs found over and over again, with the need for deduplication
- New bugs are often **missed** with patch sometimes not even reached
- Bugs reported with significant delay: **expensive context switching**

Developers need feedback within *minutes* of patch submission
***Quick directed fuzzing* campaigns required in a CI/CD context**

Testing Evolving Software

Reuse testing results

of previous versions

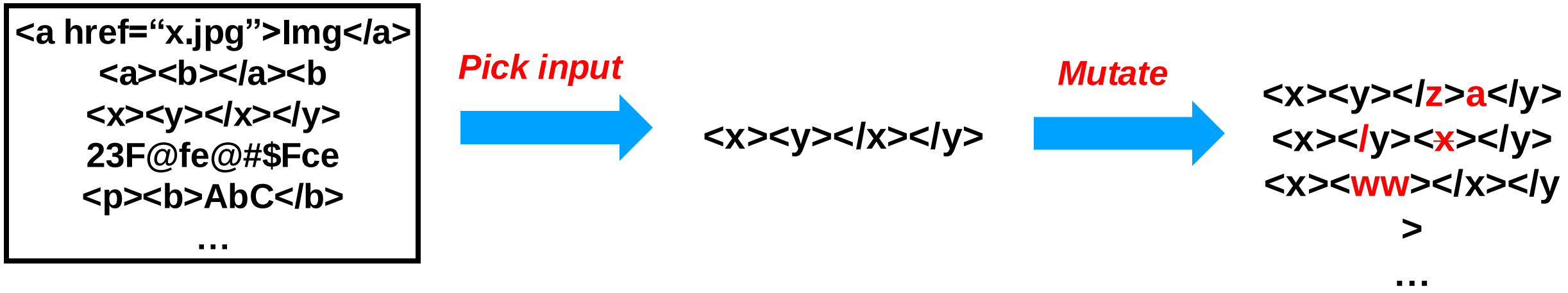


Direct testing effort

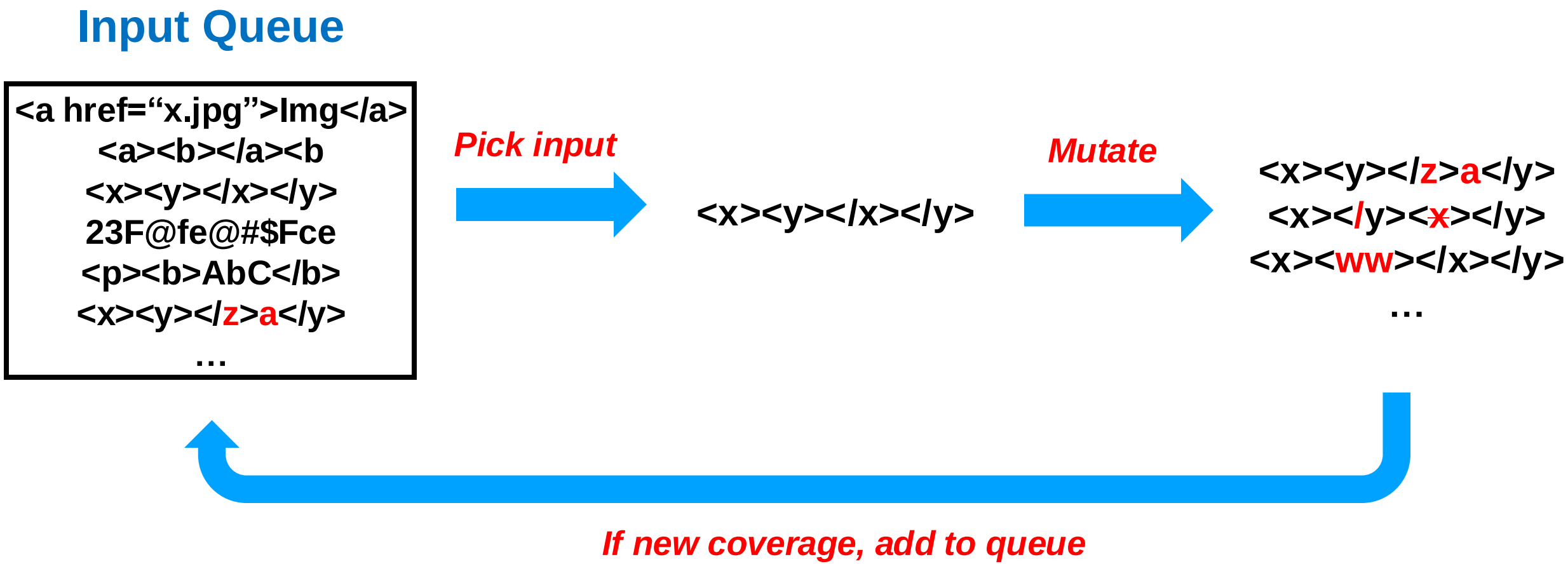
toward the changes

Greybox Fuzzing:
Coverage-guided Mutation-based Fuzzing

Input Queue



Greybox Fuzzing:
Coverage-guided Mutation-based Fuzzing



AFLGo:
State-of-the-Art Directed Greybox Fuzzing

- AFLGo is a pioneering tool for directed greybox fuzzing
- It extends traditional fuzzing by targeting specific code areas
- Computes distance estimates to prioritize inputs close to the target

- But distance computation can be expensive
- Fuzzing budget may be exhausted before any fuzzing is done

Directed Greybox Fuzzing

Marcel Böhme
National University of Singapore, Singapore
marcel.boehme@acm.org

Van-Thuan Pham*
National University of Singapore, Singapore
thuanpv@comp.nus.edu.sg

Manh-Dung Nguyen
National University of Singapore, Singapore
dungnguy@comp.nus.edu.sg

Abhik Roychoudhury
National University of Singapore, Singapore
abhik@comp.nus.edu.sg

PaZZER = Patch + Fuzzer

- Designed to be practical for short CI/CD runs
- Aims to find a sweet spot between time spent in distance computation and effectiveness
- Relies on less precise but quick distance estimates (using only the call graph)
- Computes distances incrementally (LPA*, Anytime-D*)

IMPERIAL

Google

Pazzer Case Study

ObjDump (>0.5 million LOC)

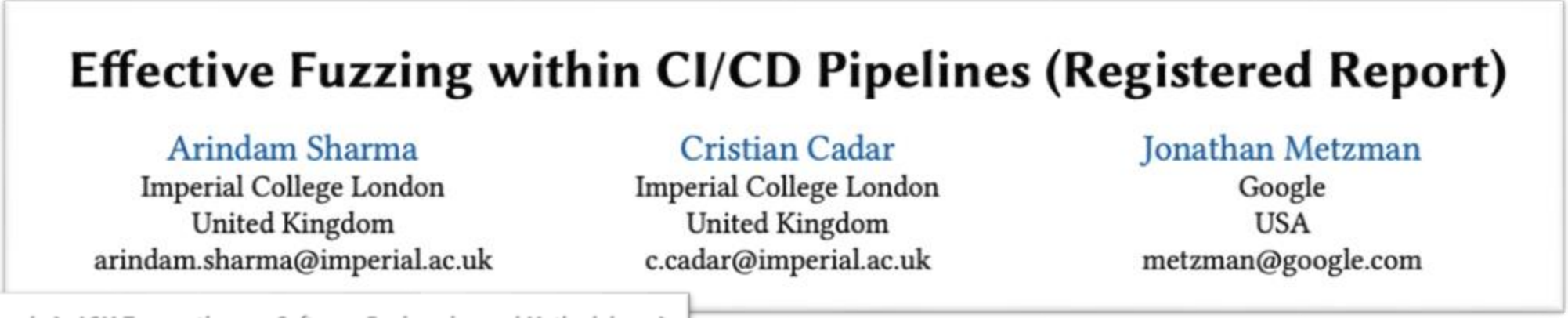
CVE-2018-8392

Time-to-Exposure (TTE)

AFLGo		
Distance	**Fuzzing**	**Total**
34 min	4 min	38 min

Pazzer (non-incremental)		
Distance	**Fuzzing**	**Total**
< 3 min	< 5 min	7 min

Pazzer (incremental)		
Distance	**Fuzzing**	**Total**
14 sec	< 5 min	5 min



Effective Fuzzing within CI/CD Pipelines (Registered Report)

Arindam Sharma
Imperial College London
United Kingdom
arindam.sharma@imperial.ac.uk

Cristian Cadar
Imperial College London
United Kingdom
c.cadar@imperial.ac.uk

Jonathan Metzman
Google
USA
metzman@google.com

Home > ACM Journals > ACM Transactions on Software Engineering and Methodology >

Sections

Registered Papers

Journal Special Issue on Fuzzing: What about Preregistration?

22 Apr 2021

co-authored by Marcel Böhme (Monash University), László Szekeres (Google), Baishakhi Ray (Columbia University), Cristian Cadar (Imperial College London)

Preregistration Model:

- Common in other fields, such as medicine
- Registered report = paper minus evaluation results
- Judged based on idea, preliminary results & planned evaluation methodology
- If the registered report is accepted, the full paper is accepted if the methodology is followed
- On the author side:
  - Avoids overclaims and (inadvertend) p-hacking
  - Avoids duplicated efforts when results are poor and allows useful negative results to be published
  - Early feedback, before expensive experiments are run, can be more easily incorporated
- On the reviewing side:
  - Avoids confirmation bias, where reviewers are more likely to favour results confirming their own view
  - Avoids results bias, where reviewers give more consideration to positive or surprising results

Dynamic Symbolic Execution (DSE)

Program analysis technique for ***automatically exploring paths*** through a program

Applications in:

- Bug finding
- Test generation
- Vulnerability detection and exploitation
- Equivalence checking
- Debugging
- Program repair
- Bounded verification
- etc. etc.

From Symbolic Execution to Dynamic Symbolic Execution

- Symbolic execution introduced in the 70s
- Revived mid-2000 in its “dynamic” form by the DART and EGT projects
- Significant interest in the last few years
- Many dynamic symbolic execution tools available:
  - KLEE, CREST, SPF, S2E, Mayhem, FuzzBall, Angr, SymCC, PEX, Otter, SymJS, PyExZ3, Manticore, Triton, SymEx-VP, Owi, Symbooglix, SymDroid, Kite, etc.
- Started to be explored and adopted by industry:
  - Fujitsu, Microsoft, Hitachi, Bloomberg, Intel, NASA, Samsung, Huawei, Baidu, etc.
  - Microsoft’s SAGE found 1/3 of file fuzzing bugs during development of Win 7

https://klee-se.org/
https://github.com/klee/

Popular dynamic symbolic executor primarily developed and maintained at Imperial

Works at the LLVM level: C (full support), C++, Rust

Active user and developer base:

- 100+ contributors to KLEE and its subprojects
- 400+ mailing list subscribers
- 600+ forks
- 2500+ stars
- 400+ participants across the first four KLEE workshops

4th International KLEE Workshop on Symbolic Execution

15–16 April 2024 • Lisbon, Portugal • Co-located with ICSE 2024

https://klee-se.org/
https://github.com/klee/

Academic impact:

- ACM SIGOPS Hall of Fame Award and ACM CCS Test of Time Award
- Over 4,500 citations to original KLEE paper (OSDI 2008)
- From many different research communities: testing, verification, systems, software engineering, PL, security, etc.
- Many different systems using KLEE: AEG, Angelix , BugRedux , Cloud9, GKLEE, KleeNet, KLEE-UC, S2E, SemFix, etc.

Growing impact in industry:

- **Baidu**: [KLEE 2018]
- **Fujitsu**: [PPoPP 2012], [CAV 2013], [ICST 2015], [IEEE Software 2017], [KLEE 2018]
- **Google**: [2x KLEE 2021]
- **Hitachi**: [CPSNA 2014], [ISPA 2015], [EUC 2016], [KLEE 2021]
- **Intel**: [WOOT 2015]
- **NASA Ames**: [NFM 2014]
- **Samsung**: 2 x [KLEE 2018], [KLEE 2024]
- **Trail of Bits** [blog.trailofbits.com/]
- **etc.**

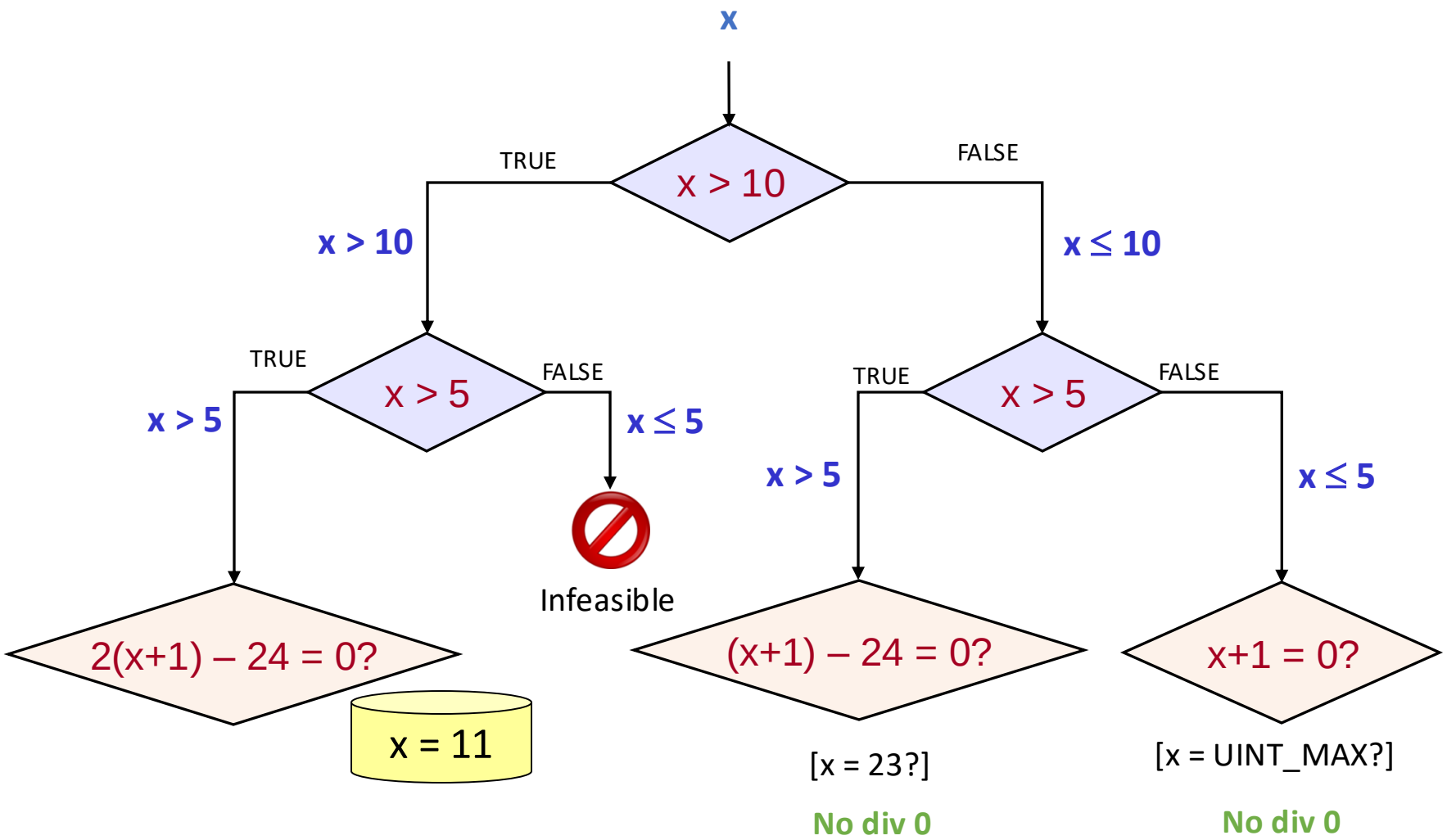
Dynamic Symbolic Execution

```
int foo(unsigned x) {
  int r = x + 1;

  if (x > 10)
    r = 2 * r;

  if (x > 5)
    r = r - 24;

  return x / r;
}
```



34

Dynamic Symbolic Execution

Key advantages:

- Systematically explores unique control-flow paths
- Produces test cases
- No false positives

- Reasons about all possible values on each explored path
- Per-path verification

Key challenges:

- Efficiently solving lots of constraints
- Path explosion, particularly in the presence of loops

DSE for Evolving Software
Direct DSE Effort Toward the Change

1. Use distance estimates to favour paths close to the change
2. Prune paths unrelated to the change

KATCH = KLEE + PATCH

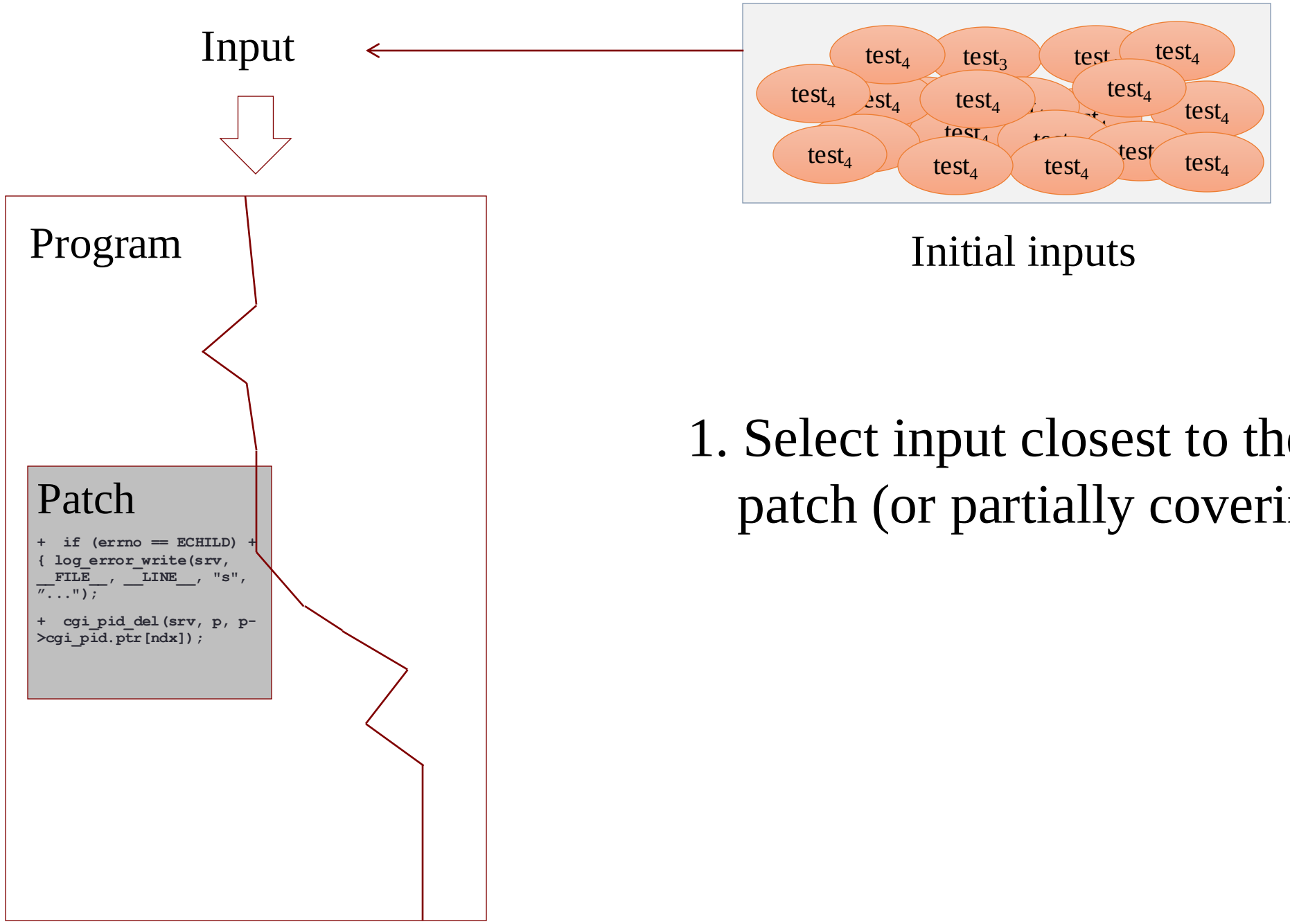
- Similar in spirit to AFLGo and Pazzer
  - But KATCH was published earlier
- Use distance estimates to guide path exploration

KATCH: High-Coverage Testing of Software Patches

Paul Dan Marinescu
Department of Computing
Imperial College London, UK
p.marinescu@imperial.ac.uk

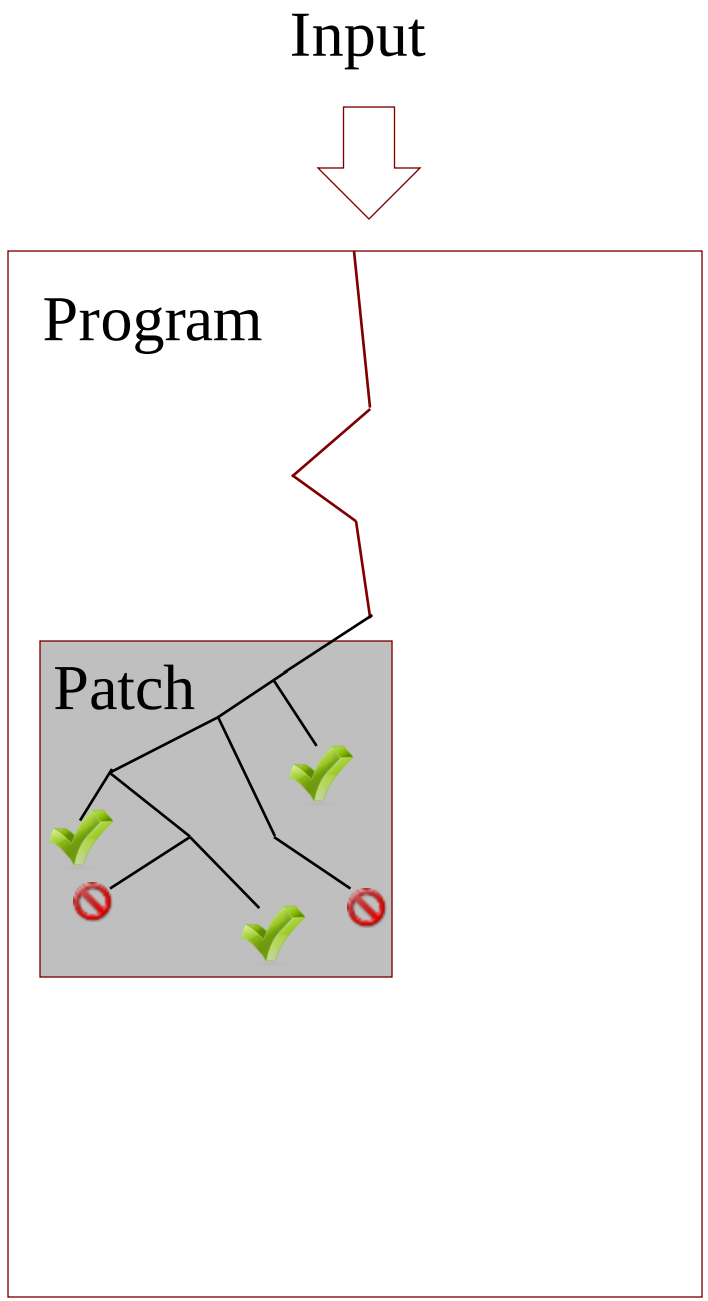
Cristian Cadar
Department of Computing
Imperial College London, UK
c.cadar@imperial.ac.uk

KATCH



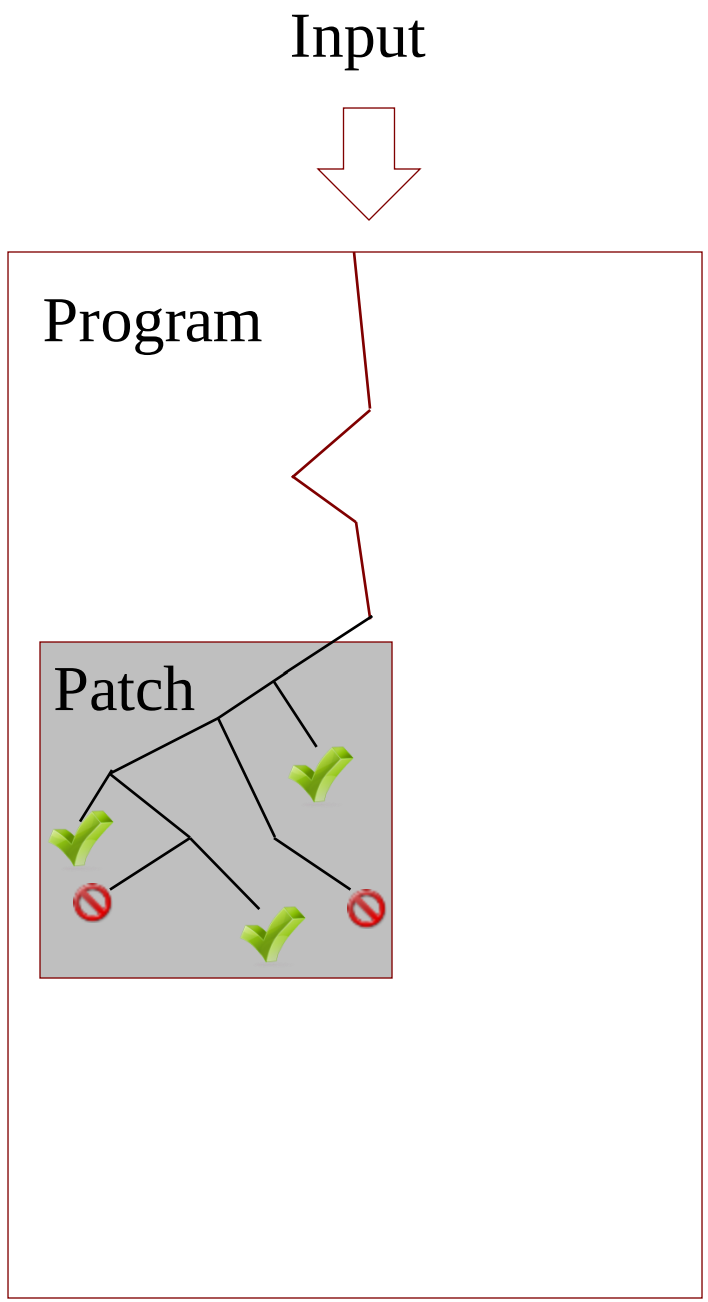
1. Select input closest to the patch (or partially covering it)

KATCH



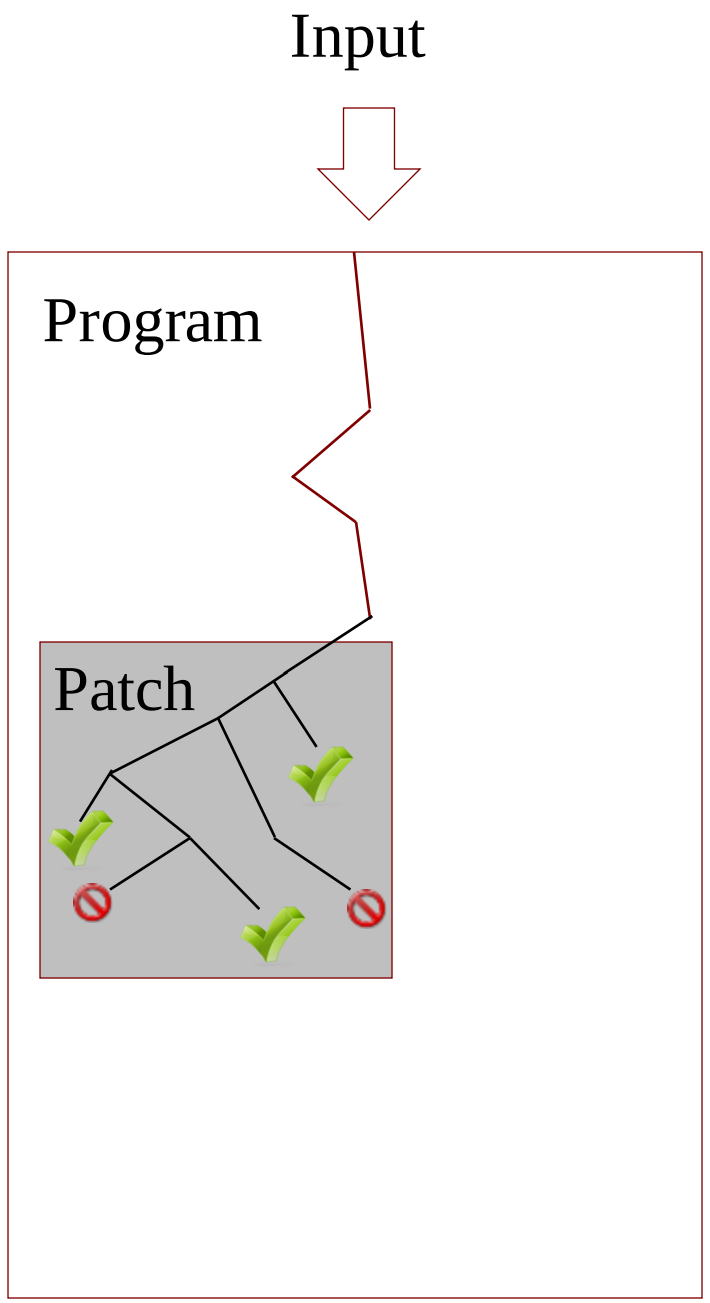
2. Greedily drive exploration toward uncovered basic blocks in the patch using distance estimates

KATCH



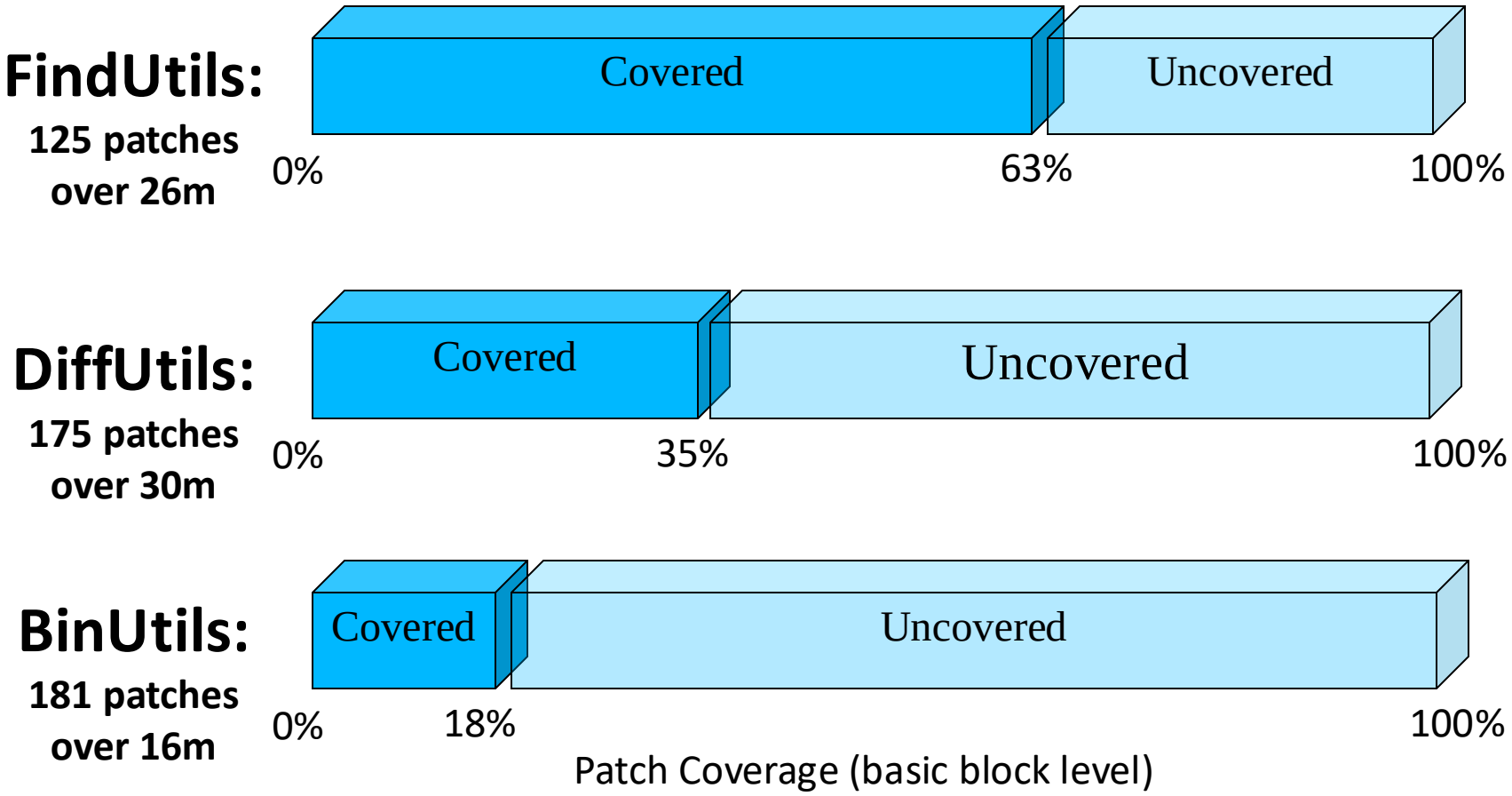
3. If stuck, identify the constraints that disallow execution to reach the patch, and backtrack

KATCH

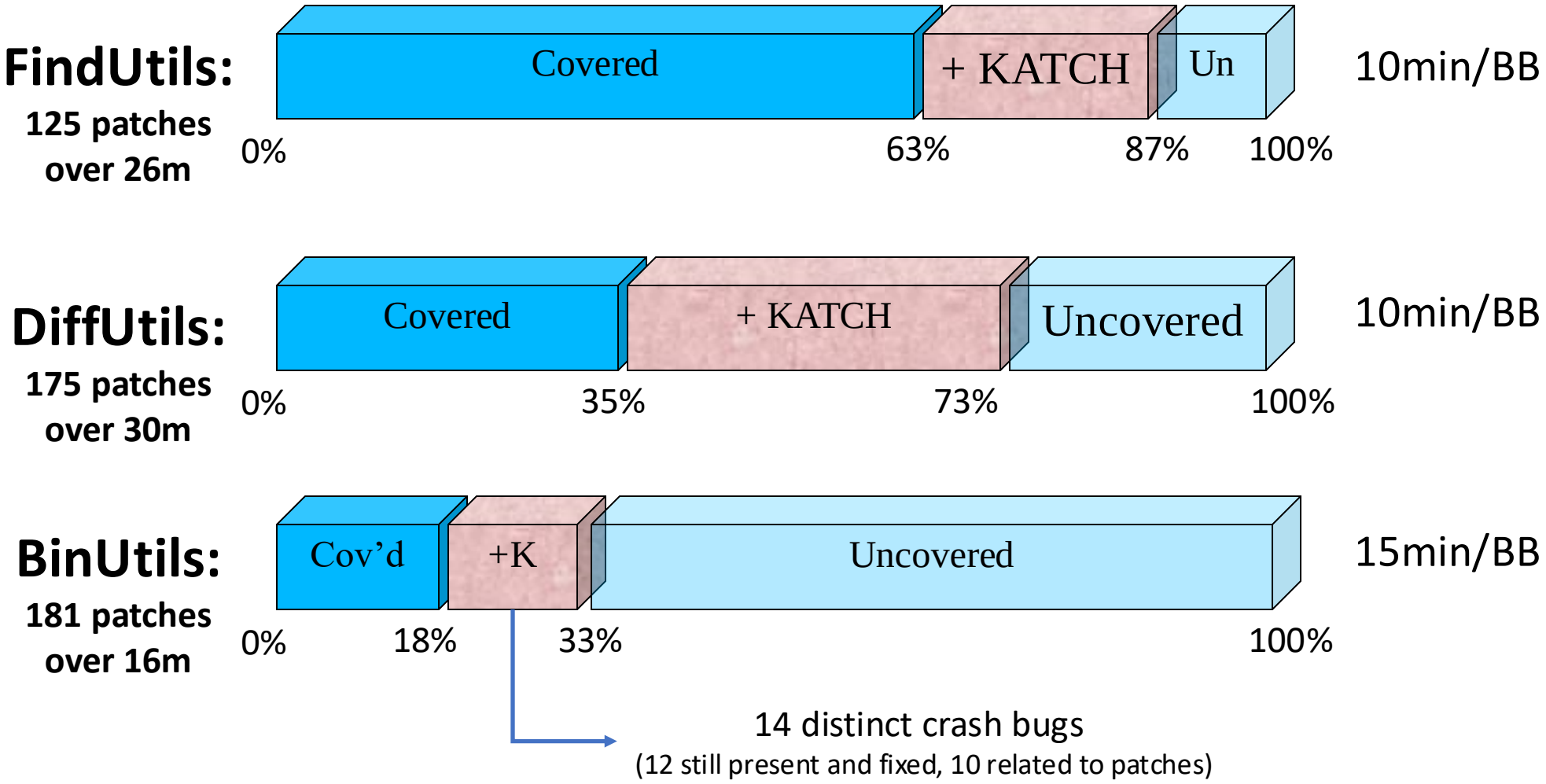


Combines **dynamic symbolic execution** with various program analyses such as **weakest preconditions** for input selection, and **definition switching** for backtracking

Developers' Patch Testing

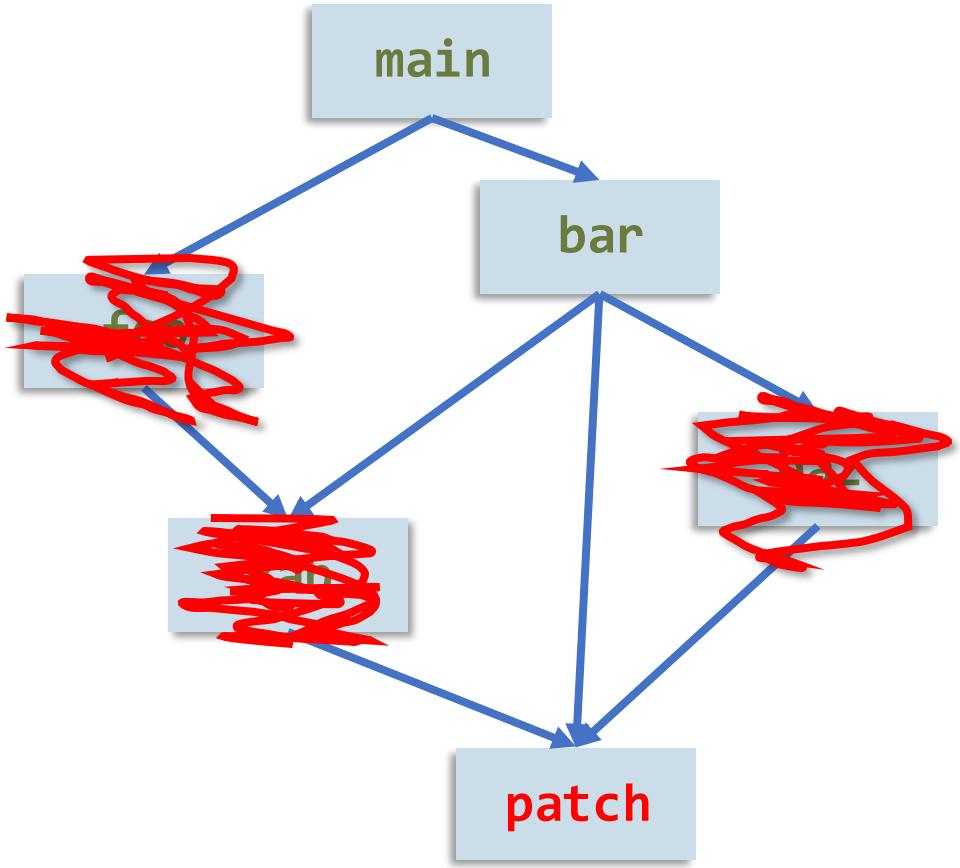


KATCH Patch Testing



Prune Search Space Unrelated to Patch

- Many code fragments are unrelated to the patch
  - But DSE can spend lots of time unnecessarily analyzing them
- Determining precisely if a part of the code is unrelated is hard
  - Often, most computation in a code fragment is unrelated, but not all



Chopped Symbolic Execution

IDEA:

1) Guess unrelated code fragments via lightweight analysis
2) Compute the side effects of these code fragments (*write set*)
3) Speculatively skip these code fragments
4) If their side effects are ever needed, go back and execute relevant skipped paths

Chopped Symbolic Execution

David Trabish
Tel Aviv University
Israel
davivtra@post.tau.ac.il

Andrea Mattavelli
Imperial College London
United Kingdom
amattave@imperial.ac.uk

Noam Rinetzky
Tel Aviv University
Israel
maon@cs.tau.ac.il

Cristian Cadar
Imperial College London
United Kingdom
c.cadar@imperial.ac.uk

Preliminary Experience:
Reproducing Security Vulnerabilities

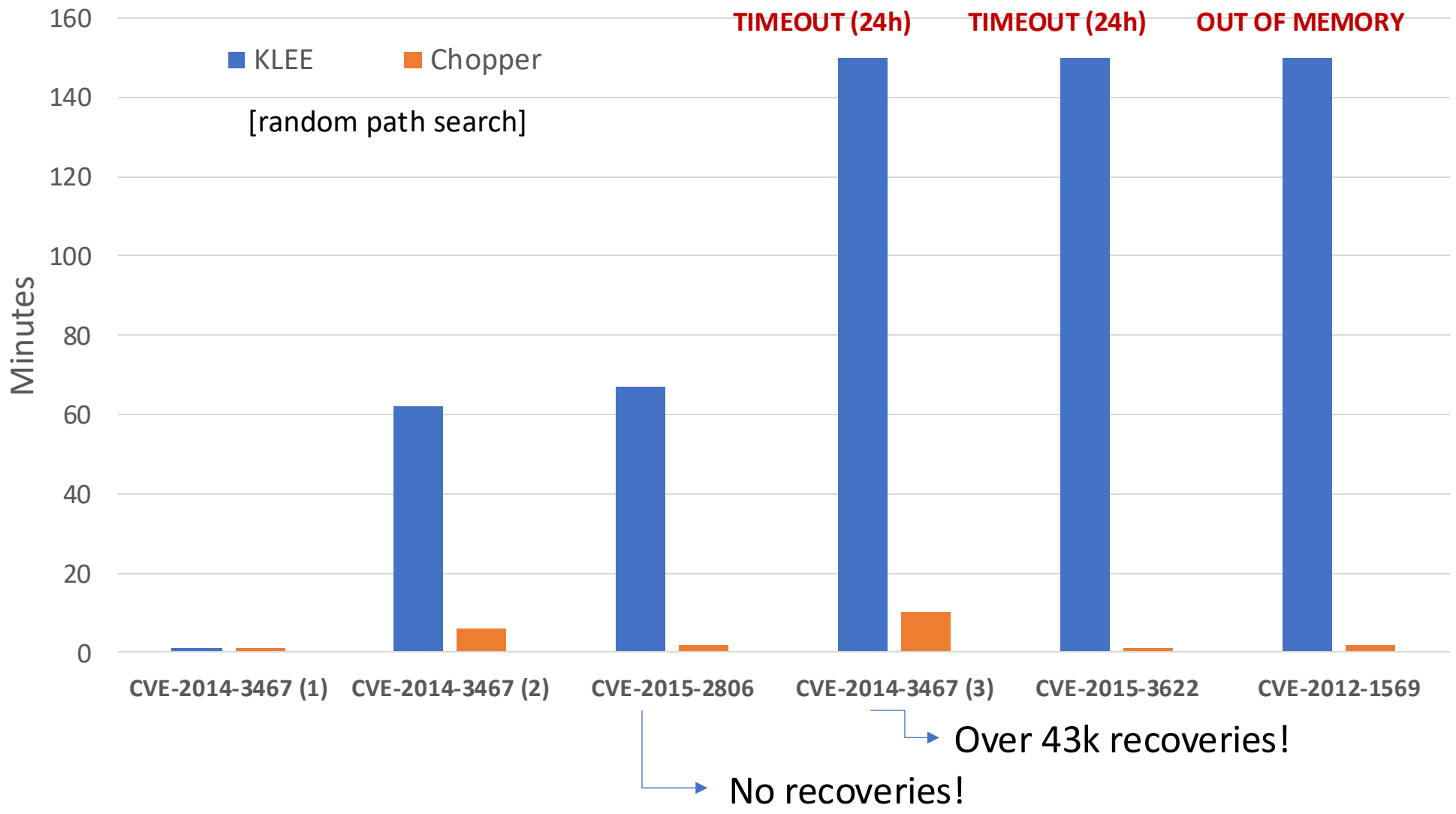
Goal: given vulnerable location, generate an input that triggers the vulnerability

- Time limit: 24 hours

Benchmark: GNU libtasn1

- ASN.1 protocol used in many networking and cryptographic applications, such as for public key certificates and e-mail

Reproducing Security Vulnerabilities



Testing Evolving Software

Direct testing effort

toward the changes

Start directly from the changed code!

E.g., construct fuzz drivers for any changed function

So far, mostly a manual process!

CHALLENGES

- Initialising state
- Constructing (complex) data structures
- Calling APIs in the right order
- Checking the result is correct (*test oracle*)

OSS-Fuzz and Fuzz Targets

APPLICATION	OSS-FUZZ	COVERAGE	FUZZ TARGETS
APR			
BINUTILS	✓	32.21%	26
CURL	✓	21.67%	17
GIT	✓	10. 68%	11
LIGHTTPD	✓	35.39%	1
MEMCACHED			
REDIS			
VIM			
ZEROMQ			

According to Fuzz Introspector, https://introspector.oss-fuzz.com/, 27 November 2024

Enter Large Language Models

oss-fuzz-gen

A Framework for Fuzz Target Generation and Evaluation

This framework generates fuzz targets for real-world `C` / `C++/Java/Python` projects with various Large Language Models (LLM) and benchmarks them via the `OSS-Fuzz` platform.

Inputs:

- Guidance about the task
- Target function's signature and source code
- Header files are available in the target project
- Examples of cross-references that use the target function

Taken from https://github.com/google/oss-fuzz-gen

oss-fuzz-gen

Hi, please write a fuzz harness for me.

The target project is `https://github.com/memononen/nanosvg` which is a open source project written in C. […]

I would like for you to write the harness targeting the function `NSVGimage * nsvgParse(char *, const char *, float).`

[…]

Example cross reference from function NSVGimage * nsvgParseFromFile(const char *, const char *, float) […] is:

```
NSVGimage* nsvgParseFromFile(const char*
filename, const char* units, float dpi) {

  ...

  if (fread(data, 1, size, fp) != size)
     goto error;

  data[size] = '\0'; // Must be null terminated.

  fclose(fp);

  image = nsvgParse(data, units, dpi);

  ...

}
```

oss-fuzz-gen

```
int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
    // Prepare input data
    char* input = (char*)malloc(size + 1);
    memcpy(input, data, size);
    input[size] = '\0'; // Must be null terminated
    const char* units = "px";
    float dpi = 96.0f;

    // Call target function
    NSVGimage* image = nsvgParse(input, units, dpi);

    // Cleanup
    if (input) free(input);
    if (image) nsvgDelete(image);

    return 0;
}
```

Ongoing and Future Work

- Provide richer usage examples!
- These seem key to good fuzz driver generation

- Shift focus to library APIs
- Critical pieces of infrastructure
- Unlike internal functions, they are meant to be called directly
- Key advantage: availability of diverse clients that provide real-world usage examples

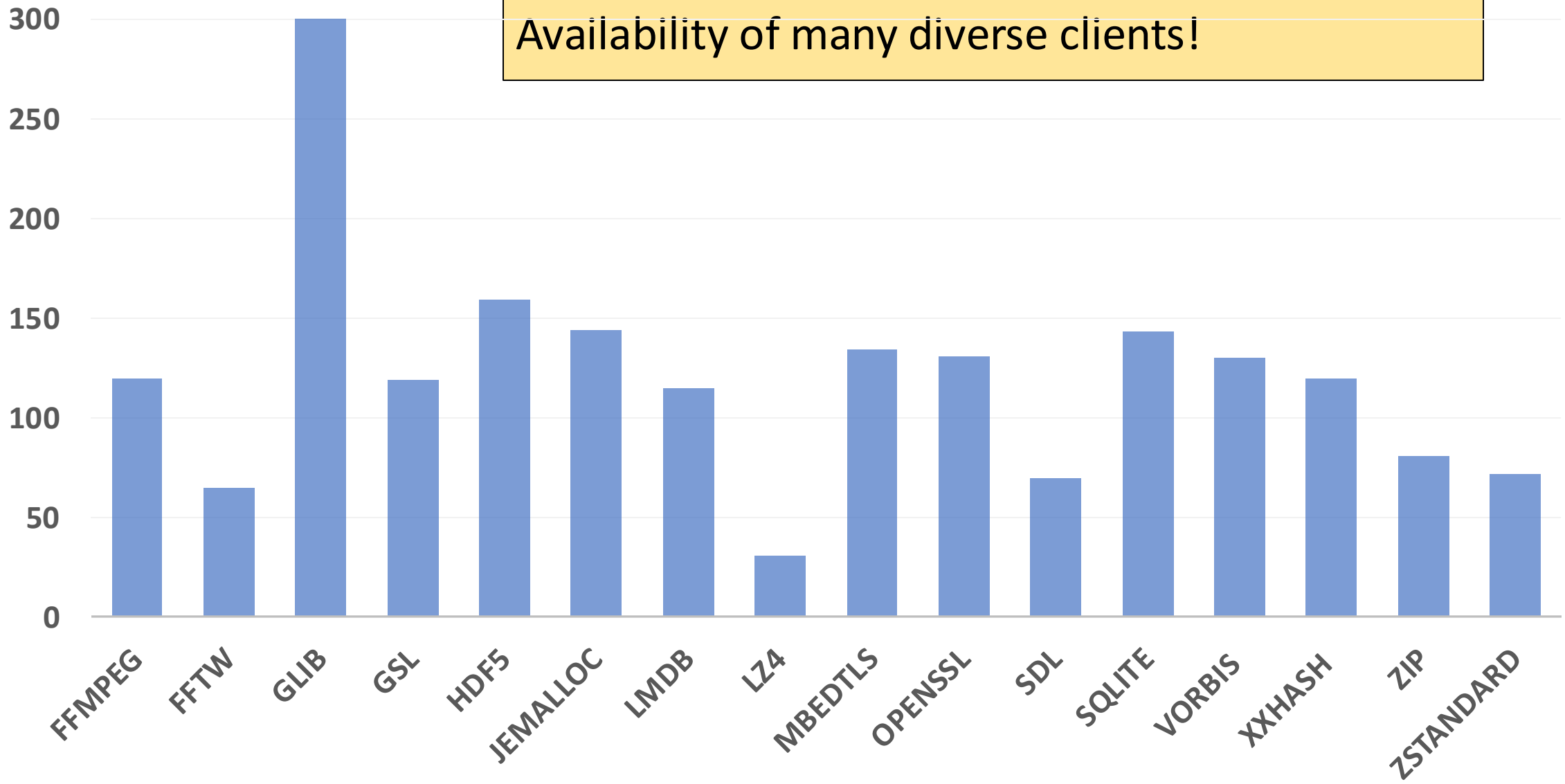
Ahmed Zaki and Cristian Cadar, ongoing work

Clients

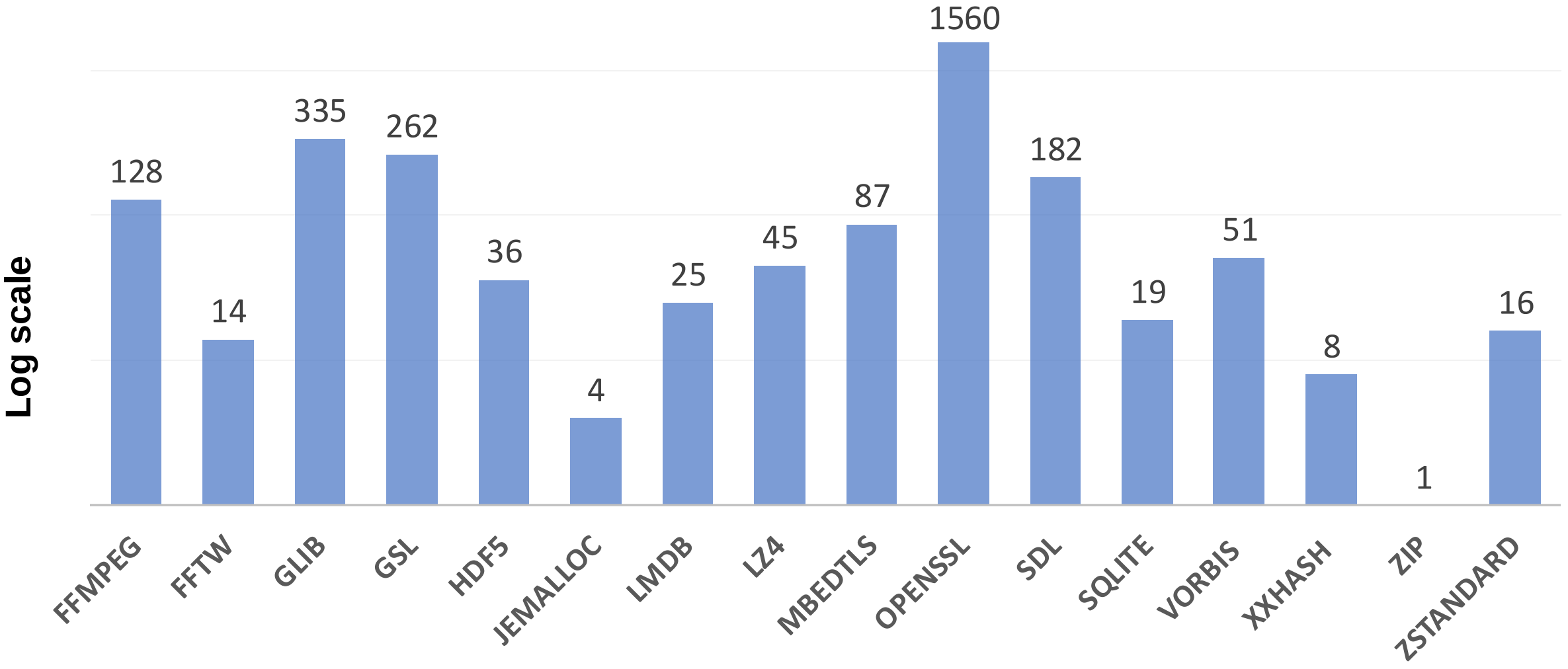
16 libraries from the CCScanner database

- Thousands to hundreds of thousands lines of code
- Some with thousands of APIs

Availability of many diverse clients!



APIs not exercised by library test suites, but used by clients



Key Insight:
Extract API Usage Examples from Clients

Requirement:

Small, self-contained examples, as independent of client code as possible

Solution:

Use program analysis to slice out the minimum relevant code sequence

Fuzz Driver Generation via Program Analysis & LLMs

```
...
key.mv_size = sizeof(int);
key.mv_data = &key_data;
data.mv_size = strlen(expected_data);
data.mv_data = expected_data;
E(mdb_put(txn, dbi, &key, &data, 0));
E(mdb_txn_commit(txn));


// Begin a new transaction for
reading
E(mdb_txn_begin(env, NULL,
MDB_RDONLY, &txn));
```

```
// Perform the database get operation
rc = mdb_get(txn, dbi, &key, &data);
if (rc == MDB_NOTFOUND) {
  printf("Key not found.\n");
} else {
  CHECK(rc == MDB_SUCCESS, "mdb_get");
  CHECK(data.mv_size ==
strlen(expected_data), "Data size
mismatch");
  CHECK(strncmp((char *)data.mv_data,
        expected_data, data.mv_size)
        == 0, "Data content mismatch");
...
```

Program Analysis for Safe and Secure Software Evolution

Cristian Cadar

SOFTWARE RELIABILITY GROUP
Imperial College London

Funded by Engineering and Physical Sciences Research Council, erc

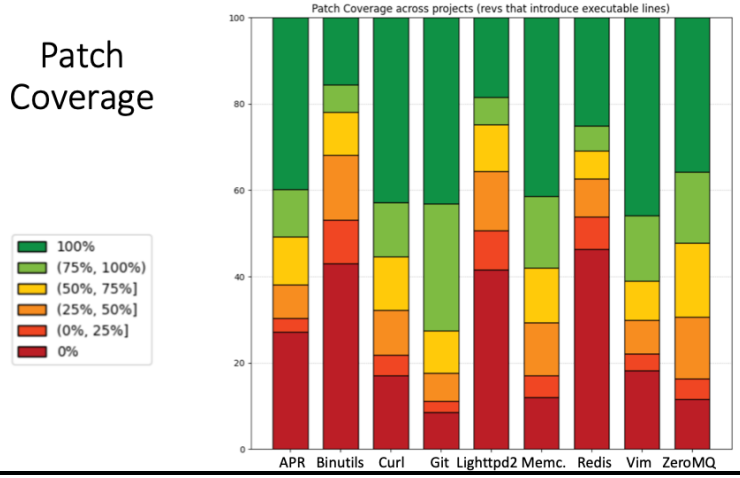
Imperial Global Singapore
Singapore, 28 November 2024

Evolving Software

- Code changes are poorly validated and often introduce bugs & vulnerabilities
- Some with catastrophic impact

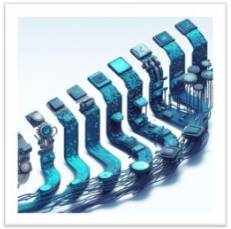
Heartbleed (2014) **Shellshock (2014)** **Stagefright (2016)** **Crowdstrike (2024)**

Patch Coverage



Testing Evolving Software

Reuse testing results of previous versions



Direct testing effort toward the changes

PaZZER = Patch + Fuzzer

- Designed to be practical for short CI/CD runs
- Aims to find a sweet spot between time spent in distance computation and effectiveness
- Relies on less precise but quick distance estimates (using only the call graph)
- Computes distances incrementally (LPA*, Anytime-D*)

IMPERIAL

Google

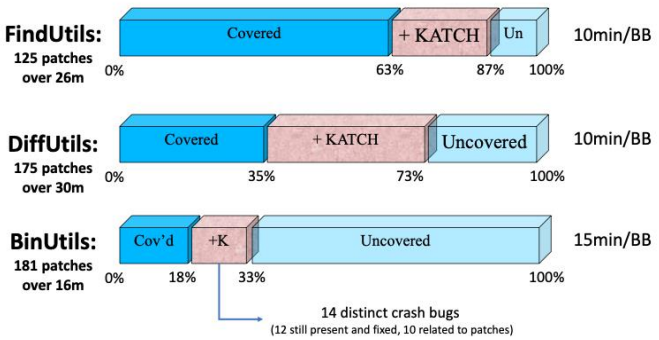
KLEE

https://klee-se.org/
https://github.com/klee/

Popular dynamic symbolic executor primarily developed and maintained at Imperial
Works at the LLVM level: C (full support), C++, Rust

Active user and developer base:
- 100+ contributors to KLEE and its subprojects
- 400+ mailing list subscribers
- 600+ forks
- 2500+ stars
- 400+ participants across the first four KLEE workshops

KATCH Patch Testing



Preliminary Experience: Reproducing Security Vulnerabilities

Goal: given vulnerable location, generate an input that triggers the vulnerability
- Time limit: 24 hours

Benchmark: GNU libtasn1
- ASN.1 protocol used in many networking and cryptographic applications, such as for public key certificates and e-mail

Fuzz Driver Generation via Program Analysis & LLMs

```
...
key.mv_size = sizeof(int);
key.mv_data = &key_data;
data.mv_size = strlen(expected_data);
data.mv_data = expected_data;
E(mdb_put(txn, dbi, &key, &data, 0));
E(mdb_txn_commit(txn));

// Begin a new transaction for reading
E(mdb_txn_begin(env, NULL, MDB_RDONLY, &txn));
```

```
// Perform the database get operation
rc = mdb_get(txn, dbi, &key, &data);
if (rc == MDB_NOTFOUND) {
  printf("Key not found.\n");
} else {
  CHECK(rc == MDB_SUCCESS, "mdb_get");
  CHECK(data.mv_size ==
strlen(expected_data), "Data size mismatch");
  CHECK(strncmp((char *)data.mv_data,
        expected_data, data.mv_size)
        == 0, "Data content mismatch");
...
```