TestCov:

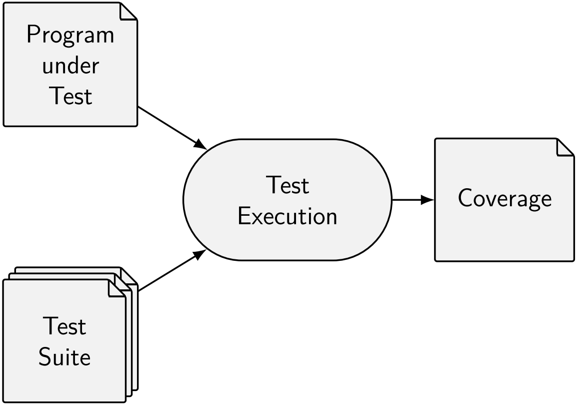
Robust Test-Suite Execution and Coverage Measurement

Thomas Lemberger
Joint work with Dirk Beyer

LMU Munich, Germany

2019-11-12, ASE 2019

Our Starting Point



- In our case: International Competition of Software Testing (Test-Comp)

The Issue

```
#include <stdio.h>
#include <unistd.h>
extern char input();

int main() {
  char x = input();
  if (x == 'a') {
    while (1) {
       fork();
    }
  } else {
    remove("important.txt");
    if (access("important.txt", F_OK) != −1) {
      return 1;
    }
  }
}
```

The Issue

```
#include <stdio.h>
#include <unistd.h>
extern char input();

int main() {
  char x = input();
  if (x == 'a') {
    while (1) {
        fork();
    }
  } else {
    remove("important.txt");
    if (access("important.txt", F_OK) != -1) {
      return 1;
    }
  }
}
```

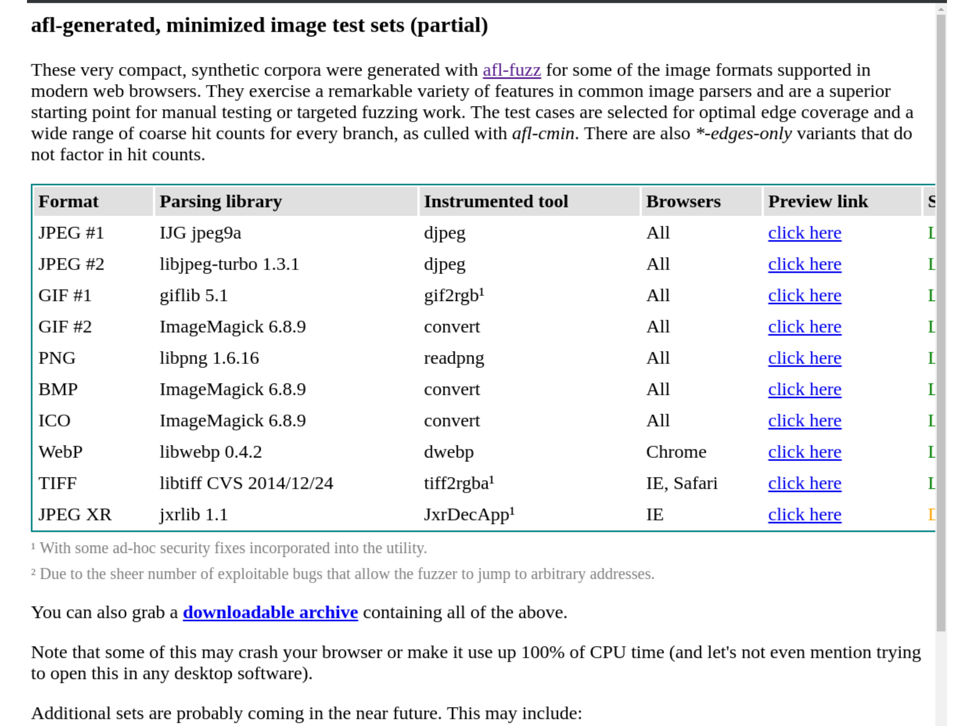
The Issue

```
#include <stdio.h>
#include <unistd.h>
extern char input();

int main() {
  char x = input();
  if (x == 'a') {
    while (1) {
        fork();
    }
  } else {
    remove("important.txt");
    if (access("important.txt", F_OK) != -1) {
      return 1;
    }
  }
}
```

The Issue

```
#include <stdio.h>
#include <unistd.h>
extern char input();

int main() {
  char x = input();
  if (x == 'a') {
    while (1) {
       fork();
    }
  } else {
    remove("important.txt");
    if (access("important.txt", F_OK) != −1) {
      return 1;
    }
  }
}
```

- Goal: Achieve 100 % branch coverage
- But: You don't want to use your system to execute a test suite that achieves that.



afl-generated, minimized image test sets (partial)

These very compact, synthetic corpora were generated with afl-fuzz for some of the image formats supported in modern web browsers. They exercise a remarkable variety of features in common image parsers and are a superior starting point for manual testing or targeted fuzzing work. The test cases are selected for optimal edge coverage and a wide range of coarse hit counts for every branch, as culled with *afl-cmin*. There are also **-edges-only* variants that do not factor in hit counts.

Format	Parsing library	Instrumented tool	Browsers	Preview link	S
JPEG #1	IJG jpeg9a	djpeg	All	click here	L
JPEG #2	libjpeg-turbo 1.3.1	djpeg	All	click here	L
GIF #1	giflib 5.1	gif2rgb[1]	All	click here	L
GIF #2	ImageMagick 6.8.9	convert	All	click here	L
PNG	libpng 1.6.16	readpng	All	click here	L
BMP	ImageMagick 6.8.9	convert	All	click here	L
ICO	ImageMagick 6.8.9	convert	All	click here	L
WebP	libwebp 0.4.2	dwebp	Chrome	click here	L
TIFF	libtiff CVS 2014/12/24	tiff2rgba[1]	IE, Safari	click here	L
JPEG XR	jxrlib 1.1	JxrDecApp[1]	IE	click here	I

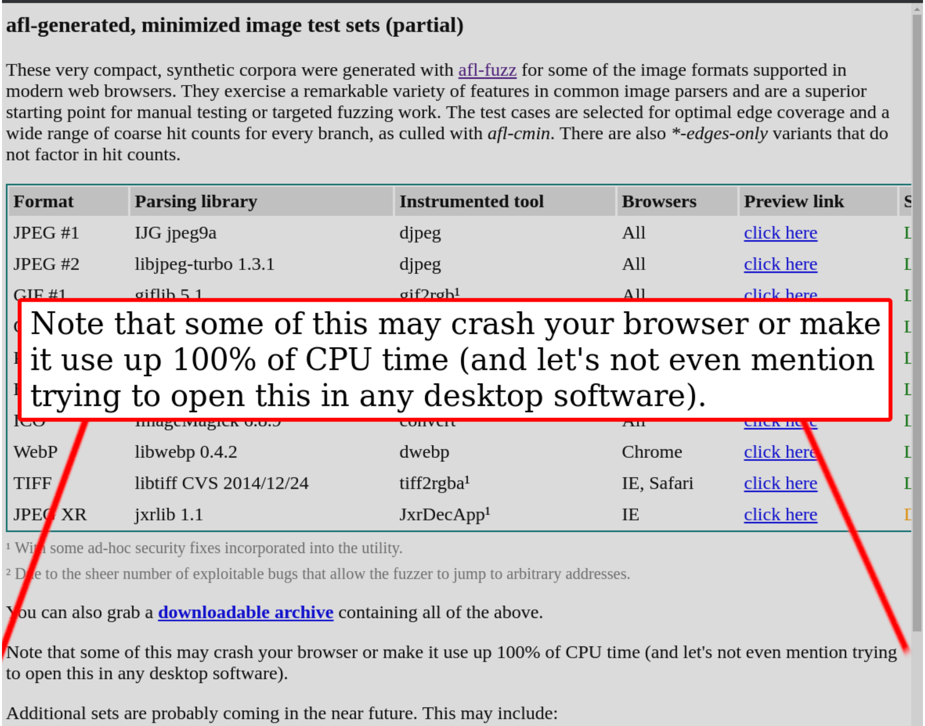
[1] With some ad-hoc security fixes incorporated into the utility.

[2] Due to the sheer number of exploitable bugs that allow the fuzzer to jump to arbitrary addresses.

You can also grab a **downloadable archive** containing all of the above.

Note that some of this may crash your browser or make it use up 100% of CPU time (and let's not even mention trying to open this in any desktop software).

Additional sets are probably coming in the near future. This may include:



afl-generated, minimized image test sets (partial)

These very compact, synthetic corpora were generated with afl-fuzz for some of the image formats supported in modern web browsers. They exercise a remarkable variety of features in common image parsers and are a superior starting point for manual testing or targeted fuzzing work. The test cases are selected for optimal edge coverage and a wide range of coarse hit counts for every branch, as culled with *afl-cmin*. There are also **-edges-only* variants that do not factor in hit counts.

Format	Parsing library	Instrumented tool	Browsers	Preview link	S
JPEG #1	IJG jpeg9a	djpeg	All	click here	L
JPEG #2	libjpeg-turbo 1.3.1	djpeg	All	click here	L
GIF #1	giflib 5.1	gif2rgb[1]	All	click here	L
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	L
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	L
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	L
ICO	ImageMagick 6.8.9	convert	All	click here	L
WebP	libwebp 0.4.2	dwebp	Chrome	click here	L
TIFF	libtiff CVS 2014/12/24	tiff2rgba[1]	IE, Safari	click here	L
JPEG XR	jxrlib 1.1	JxrDecApp[1]	IE	click here	L

Note that some of this may crash your browser or make it use up 100% of CPU time (and let's not even mention trying to open this in any desktop software).

[1] With some ad-hoc security fixes incorporated into the utility.

[2] Due to the sheer number of exploitable bugs that allow the fuzzer to jump to arbitrary addresses.

You can also grab a **downloadable archive** containing all of the above.

Note that some of this may crash your browser or make it use up 100% of CPU time (and let's not even mention trying to open this in any desktop software).

Additional sets are probably coming in the near future. This may include:

Existing Solutions to Robust Execution

- Virtual Machines
- Containerization (Docker etc.)

⇒ Potentially large overhead

⇒ Manual setup

⇒ Setups consist of multiple tools

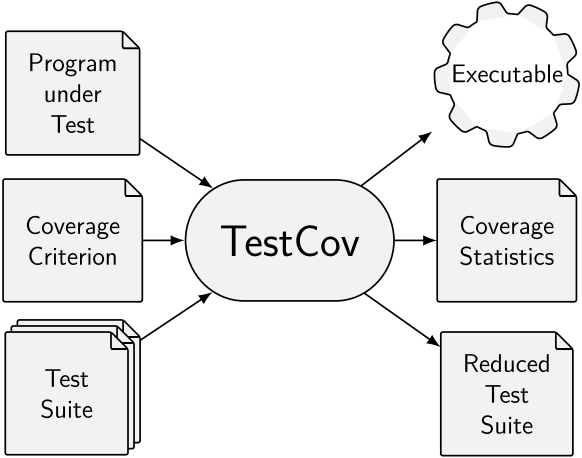
⇒ Require superuser privileges

Our Solution

- Test isolation through Linux kernel features
- Coherent, single tool (for C programs)

Our Solution

- Test isolation through Linux kernel features
- Coherent, single tool (for C programs)



Robust Test Execution

- Malicious influences:
  - Resource exhaustion
  - File system modifications
  - Dependencies between tests

⇒ Isolate each individual run

- Technology:
  - Control Groups (CGroups)
  - Containers
- Both provided by BenchExec

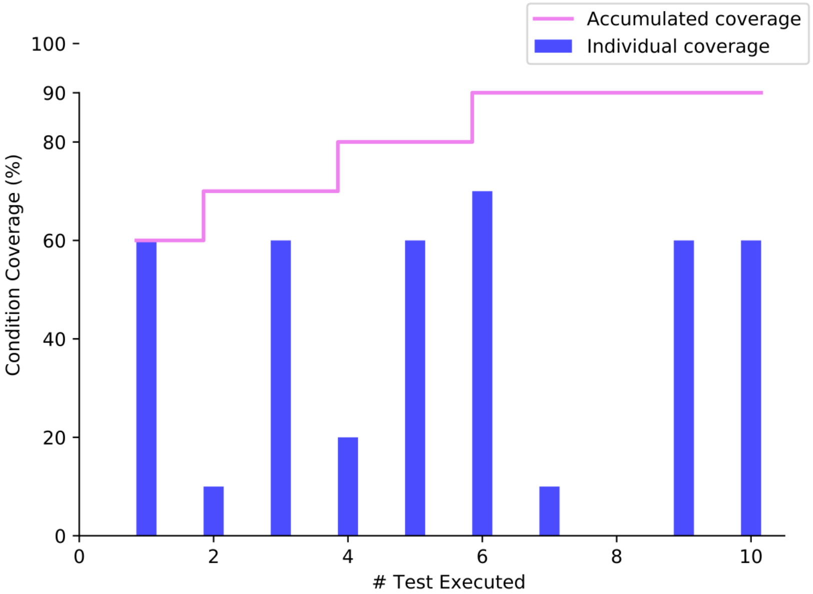
  https://github.com/sosy-lab/benchexec/

Coverage Measurements

- Measurement through `lcov` and `llvm-cov` or `gcov`
  - Provide line- and condition-coverage
  - Unfitting definition of branch-coverage
- Branch coverage manually computed through program instrumentation

- Produced data:
  - Test success
  - Individual test coverage
  - Accumulated test coverage (after each execution)
  - Individual resource measurements
  - `.csv` table, `.json` data, `.svg` plot

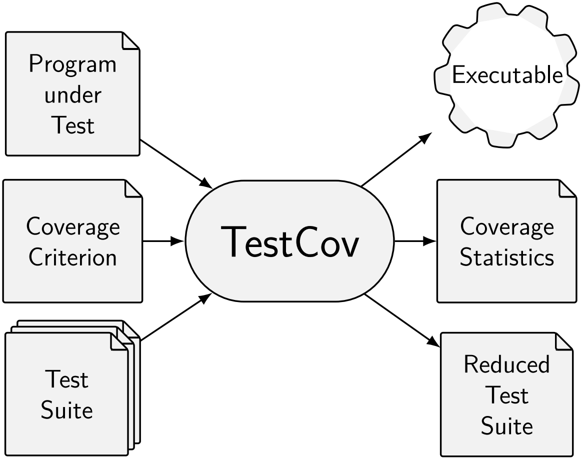
Coverage Plot



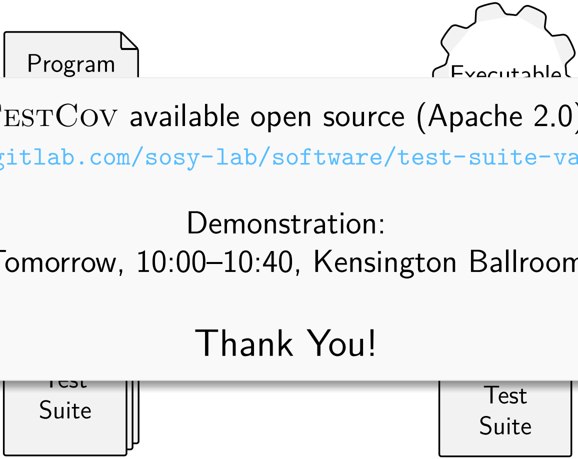
Test-Suite Reduction

- Goal: Create test suite with same coverage as input test suite, but less tests
- Strategies in `TestCov`:
  - Simple, accumulative order-based approach
  - Similarity-based approach
- Extensible through strategy pattern

Conclusion


Program under Test
Coverage Criterion
Test Suite
TestCov
Executable
Coverage Statistics
Reduced Test Suite

Conclusion



TestCov available open source (Apache 2.0):

https://gitlab.com/sosy-lab/software/test-suite-validator/

Demonstration:
Tomorrow, 10:00–10:40, Kensington Ballroom

Thank You!

References

[1] D. Beyer and T. Lemberger.
TestCov: Robust test-suite execution and coverage measurement.
In *Proc. ASE*. IEEE, 2019.

[2] D. Beyer, S. Löwe, and P. Wendler.
Reliable benchmarking: Requirements and solutions.
Int. J. Softw. Tools Technol. Transfer, 21(1):1–29, 2019.

Test-Suite Format

- XML-based
- Two components:
  1. `metadata.xml`
  2. one XML-file per test case
     - Sequence of test inputs
- Handled as zip archive

Metadata

```
<?xml version="1.0"?>
<!DOCTYPE test-metadata PUBLIC "+//IDN sosy-lab.org//DTD test-format te
<test-metadata>
  <sourcecodelang>C</sourcecodelang>
  <producer>Testsuite Validator v2.0</producer>
  < specification >CHECK(FQL(cover EDGES(@CONDITIONEDGE)))</specificatio
  <programfile>example.c</programfile>
  <programhash>eeecda9cbf27c43c9017fa00dd900c19a5ec18d46303f59a6e0357db78
  <entryfunction>main</entryfunction>
  <architecture>32bit</architecture>
  < inputtestsuitefile >original-suite. zip</ inputtestsuitefile >
  <inputtestsuitehash>11911d658dcfbf8501390bf0faa96eb193b11bb1</inputtestsui
  <creationtime>2019-06-19T14:17:34Z</creationtime>
</test-metadata>
```

Test Case

```
<?xml version="1.0"?>
<!DOCTYPE testcase PUBLIC "+//IDN sosy−lab.org//DTD test−format testcase
<testcase>
  <input>'b'</input>
  <input>10</input>
  <input>0x0f</input>
</testcase>
```