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

Program under Test

Test Execution

Test Suite

Coverage

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

```c
#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

```c
#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.
The Issue

```c
#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

```c
#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.](#)
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.

<table>
<thead>
<tr>
<th>Format</th>
<th>Parsing library</th>
<th>Instrumented tool</th>
<th>Browsers</th>
<th>Preview link</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG #1</td>
<td>IJG jpeg9a</td>
<td>djjpeg</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>JPEG #2</td>
<td>libjpeg-turbo 1.3.1</td>
<td>djjpeg</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>GIF #1</td>
<td>giflib 5.1</td>
<td>gif2rgb¹</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>GIF #2</td>
<td>ImageMagick 6.8.9</td>
<td>convert</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>PNG</td>
<td>libpng 1.6.16</td>
<td>readpng</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>BMP</td>
<td>ImageMagick 6.8.9</td>
<td>convert</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>ICO</td>
<td>ImageMagick 6.8.9</td>
<td>convert</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>WebP</td>
<td>libwebp 0.4.2</td>
<td>dwebp</td>
<td>Chrome</td>
<td>click here</td>
</tr>
<tr>
<td>TIFF</td>
<td>libtiff CVS 2014/12/24</td>
<td>tiff2rgbα</td>
<td>IE, Safari</td>
<td>click here</td>
</tr>
<tr>
<td>JPEG XR</td>
<td>jxrlib 1.1</td>
<td>JxrDecApp¹</td>
<td>IE</td>
<td>click here</td>
</tr>
</tbody>
</table>

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:
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.

<table>
<thead>
<tr>
<th>Format</th>
<th>Parsing library</th>
<th>Instrumented tool</th>
<th>Browsers</th>
<th>Preview link</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPEG #1</td>
<td>IJG jpeg9a</td>
<td>djjpeg</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>JPEG #2</td>
<td>libjpeg-turbo 1.3.1</td>
<td>djjpeg</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>GIF #1</td>
<td>giflib 5.1</td>
<td>gif2rgb¹</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>ICO</td>
<td>ImageMagick 6.9.9</td>
<td>convert</td>
<td>All</td>
<td>click here</td>
</tr>
<tr>
<td>WebP</td>
<td>libwebp 0.4.2</td>
<td>dwebp</td>
<td>Chrome</td>
<td>click here</td>
</tr>
<tr>
<td>TIFF</td>
<td>libtiff CVS 2014/12/24</td>
<td>tiff2rgba¹</td>
<td>IE, Safari</td>
<td>click here</td>
</tr>
<tr>
<td>JPEG XR</td>
<td>jxrlib 1.1</td>
<td>JxrDecApp¹</td>
<td>IE</td>
<td>click here</td>
</tr>
</tbody>
</table>

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

- **Accumulated coverage**
- **Individual coverage**

The graph shows the condition coverage (%) for different numbers of tests executed. The accumulated coverage line starts at 60% and reaches 90% after 6 tests, while the individual coverage bars show the coverage for each test.
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

TestCov

Program under Test

Coverage Criterion

Test Suite

Executable

Coverage Statistics

Reduced Test Suite

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!

Thomas Lemberger
LMU Munich, Germany
**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


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
<?xml version="1.0"?>
<!DOCTYPE test-metadata PUBLIC "+//IDN sosy-lab.org//DTD test-format metadata 1.1//EN" "https://sosy-lab.org/test-format/test-metadata-1.1.dtd">
<test-metadata>
  <sourcecodelang>C</sourcecodelang>
  <producer>Testsuite Validator v2.0</producer>
  <specification>CHECK(FQL(cover EDGES(@CONDITIONEDGE)))</specification>
  <programfile>example.c</programfile>
  <programhash>eeecda9cbf27c43c9017fa00dd900c19a5ec18d46303f59a6e0357db78</programhash>
  <entryfunction>main</entryfunction>
  <architecture>32bit</architecture>
  <inputtestsuitefile>original-suite.zip</inputtestsuitefile>
  <inputtestsuitehash>11911d658dcf8501390bf0faa96eb193b11bb1</inputtestsuitehash>
  <creationtime>2019-06-19T14:17:34Z</creationtime>
</test-metadata>
<?xml version="1.0"?>
<!DOCTYPE testcase PUBLIC "+//IDN sosy-lab.org//DTD test-format testcase 1.1//EN" "https://sosy-lab.org/test-format/testcase-1.1.dtd">
<testcase>
  <input>'b'</input>
  <input>10</input>
  <input>0x0f</input>
</testcase>