RIFF: Reduced Instruction Footprint for Coverage-Guided Fuzzing

Mingzhe Wang, Jie Liang, Chijin Zhou, Yu Jiang, Rui Wang, Chengnian Sun, and Jiaguang Sun

Tsinghua University

2 Capital Normal University

3 Waterloo University
Coverage is Important for Guided Fuzzing
Coverage Pipeline in Fuzzers

Coverage Pipeline

Corpus

\x89PNG

Select

\x89PNG

Mutate

\x89PNG\x0d
\xffPNG

New Inputs

Coverage Pipeline

Save Interesting

Coverage Statistics

Run & Collect

\x89P
Example: Coverage Collection in AFL

```c
#include <afl_log.h>

void foo();
void bar();

int main() {
    afl_maybe_log(0x52a7);
    if (...) {
        afl_maybe_log(0x236e);
        foo();
    } else {
        afl_maybe_log(0xf084);
        bar();
    }
    afl_maybe_log(0x7981);
}
```

![Diagram showing coverage collection in AFL](image)
Example: Coverage Analysis in AFL

Coverage

Fuzzer

1. Read  2. Write  Back  3. Read

Counter 01 0A 00 00 00 11 3C 00

Bitmap 01 10 00 00 00 20 40 00

Classify

Bitmap 01 10 00 00 00 20 40 00

Scan

Virgin FFF 00 00 00 00 20 40 00

Unknown Program States

3. Read
Overhead in Coverage Collection

```
lea -0x98(%rsp),%rsp
mov %rdx,(%rsp)
mov %rcx,0x8(%rsp)
mov %rax,0x10(%rsp)
mov $0x0ca5,%rcx
callq __afl_maybe_log
lahf
seto %al
mov __afl_area_ptr(%rip),%rdx
test %rdx,%rdx
je near __afl_setup
xor __afl_prev_loc(%rip),%rcx
xor __rcx,__afl_prev_loc(%rip)
shrq __afl_prev_loc(%rip)
incb (%rdx,%rcx,1)
add $0x7f,%al
sahf
retq
mov 0x10(%rsp),%rax
mov 0x8(%rsp),%rcx
mov (%rsp),%rdx
lea 0x98(%rsp),%rsp

mov __afl_prev_loc,%rax
movslq %fs:,%rcx
lea __afl_area_ptr(%rip),%rdx
mov (%rdx),%rdx
xor $0x6956,%rcx
addb $0x1,(%rdx,%rcx,1)
movl $0x3abf,%fs:,%rax
```

Method  | Duration  | Instructions | L1-I      | L1-D      | µops
---      | ---       | ---          | ---       | ---       | ---
afl-clang | 3.50x     | 4.26x        | 102.36x   | 5.16x     | 4.72x
afl-fuzzbench | 2.45x     | 2.83x        | 19.88x    | 2.53x     | 2.14x
afl-clang-fast | 1.69x     | 1.79x        | 33.58x    | 2.88x     | 2.11x
Overhead in Coverage Analysis

Table 4: Number of Processed Counters and Executions

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Useless</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>65,536</td>
<td>64,664.37</td>
<td>98.67%</td>
</tr>
<tr>
<td>Execution</td>
<td>67,696</td>
<td>67,694</td>
<td>99.997%</td>
</tr>
</tbody>
</table>
RIFF: Overview and Insights

Target Program
Move run-time computation to compile-time if possible

Fuzzer
Add hot-path processing logic specially tuned for simple cases
Single-Instruction Instrumentation: Problem of Block Coverage

Block coverage is intuitive but *incomplete*: multiple edge counts map to one block count.
Single-Instruction Instrumentation: Problem of Block Coverage

Block coverage is intuitive but *complex*: requires extra computation at fuzzer’s side.
Single-Instruction Instrumentation: Simplified Algorithm

for each potential control transfer $E$ in program $P$:

if $E$ is direct control transfer:

if basic block of $E.source$ must transfer to basic block of $E.target$:

    InstrumentBlock(basic block of $E.source$)

else if basic block of $E.target$ must transfer from basic block of $E.source$:

    InstrumentBlock(basic block of $E.target$)

else:

    InstrumentBlock(CreateDummyBlock($E$))

else:

    (Handle indirect control transfer, see the next slide)

# Single-instruction instrumentation
incb  $INDEX(%rip)   # fe 05 ?? ?? ?? ?? ??
Single-Instruction Instrumentation: Simplified Algorithm

for each potential control transfer $E$ in program $P$:

if $E$ is direct control transfer:
    (See the previous slide)
else:
    InstrumentBefore($E$.source, SetSourceID)
    InstrumentAfter($E$.target, LogEdgeTransfer)
Hot-Path Vectorized Analysis

Newly-Introduced Hot Path (99.997%)

Original Algorithm Cold Path (0.003%)

Stage 0: Vectorized Scan

Stage 1: Masked Compare

Stage 2: Infrequent Update
Hot-Path Vectorized Analysis

Stage 0: optimized for useless counters
Hot-Path Vectorized Analysis

Stage 1: optimized for useless executions
Evaluation: Overall Speedup in Fuzzing

Figure 7: Normalized execution time required by RIFF to reach the same coverage as AFL and MOpt. The X axis is programs, the Y axis is the ratio between the execution times required for reaching the same coverage. A bar below the red line indicates a speed-up.
Improved Performance Brought by Speedup

Figure 8: Normalized performance metrics for RIFF-based fuzzers after 24 hours of fuzzing. X axis is programs, Y axis is the normalized performance metric (ratio between RIFF and standard fuzzer). Bars higher than 1 (red line) indicate better performance.
Speedup in Coverage Collection and Analysis

Figure 10: Normalized execution duration of fuzzed programs: time to execute 1000 on fixed inputs normalized to the time of uninstrumented programs. Lower bars indicate better performance.

Figure 11: Coverage processing time (normalized against the baseline algorithm). Lower bars indicate better performance.
Summary

Observation
1. Coverage collection and analysis significantly affect the speed of fuzzing.
2. We break down the cost of instrumentation and analysis code.

Design
1. Accelerate coverage collection with single instruction instrumentation.
2. Accelerate coverage analysis with hot-path vectorization.

Implementation
1. Adapt RIFF to popular fuzzers, including AFL and M0pt.
2. Integrated into AFL++.
Thank You

Q & A