Program-Structure-Guided Approximation of Large Fault Spaces

Oskar Pusz, Daniel Kiechle, Christian Dietrich, Daniel Lohmann
Systems Research and Architecture Group (SRA)
Leibniz University Hanover, Germany

03.12.2019
Motivation

- Devices with high responsibility need to be reliable
- Shrinking transistors make hardware more vulnerable
- Quantifying software safeness in presence of transient hardware errors: Simulation-based fault injection
- Problem: Fault injection campaign run time
Fault Space Equivalence Sets

- Location [bits]
- Time [instructions]

- Green: Expected Behavior
- Red: Unexpected Behavior
- Blank: Fault
Approach in a nutshell

Dataflow

Ignored Dataflow
Fault Space Equivalence Sets

Def/Use-Pruning

9 Injections

Write
Read
Injection
Fault Space Regions

Region $i$

$b_i$

$b_{i+1}$

location [bits]

time [instructions]

2 Injections

Inner ES

Outer ES

25x

10

15
Realization of Program Structure

Static Basic Blocks

Dynamic Basic Blocks

trace + objdump

Program-Structure-Guided Approximation of Large Fault Spaces
Evaluation

- Fault injection framework **FAIL** is extended and emulates on the IA-32 simulator Bochs

- Used hardware:
  - Def/Use + FSR calculation: Intel i5-7400 @ 3Ghz (4 cores)
  - FI Campaign: 17 Intel Xeon @ 2.67 Ghz (12 cores each)

- Dimensions of the evaluation
  - Seven selected Programs of the automotive and security branch of the MiBench benchmark suite
  - Two instantiations of the FSRs: basic block and call regions
  - Three fault models: memory, GP registers, combined
Evaluation - Reduction

- Number of injections after the precise and complete Def/Use pruning is the baseline

- FSR calculation took end-to-end no more than 4.1 seconds

<table>
<thead>
<tr>
<th>fault model</th>
<th>Memory</th>
<th>Register</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>#injections</td>
<td>BBR[%]</td>
<td>−9.79</td>
<td>−83.73</td>
</tr>
<tr>
<td>CR [%]</td>
<td>−38.03</td>
<td>−95.44</td>
<td>−83.87</td>
</tr>
</tbody>
</table>
Evaluation - Reduction

Memory Fault Space

Fault Injection Reduction [%]

- Bitcount: 14.7, BBR: 24.8
- QSort: 15.5, CR: 44.3
- Blowfish Dec: 1.5, BBR: 25.2
- Blowfish Enc: 0.9, CR: 25.4
- Rijndael Dec: 35.1, BBR: 42.6
- Rijndael Enc: 34.6, CR: 41.4
- SHA1: 22.7, BBR: 93.0

Deviation for BBR below 0.2%
Evaluation - Reduction

Register Fault Space

<table>
<thead>
<tr>
<th></th>
<th>BBR</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitcount</td>
<td>72,18%</td>
<td>82,32%</td>
</tr>
<tr>
<td>QSort</td>
<td>73,96%</td>
<td>95,79%</td>
</tr>
<tr>
<td>Blowfish Dec</td>
<td>86,6%</td>
<td>97,82%</td>
</tr>
<tr>
<td>Blowfish Enc</td>
<td>89,43%</td>
<td>97,8%</td>
</tr>
<tr>
<td>Rijndael Dec</td>
<td>92,21%</td>
<td>97,89%</td>
</tr>
<tr>
<td>Rijndael Enc</td>
<td>92,46%</td>
<td>98,04%</td>
</tr>
<tr>
<td>SHA1</td>
<td>81,87%</td>
<td>99,62%</td>
</tr>
</tbody>
</table>

High Reduction → High Deviation
BBR up to 41.3 %, CR up to 108.8 %
Evaluation - Reduction

Combined Fault Space

Deviation BBR 0.2% - 2.7%
Evaluation - Locality

Dimensions of the evaluation
- The seven programs from MiBench
- For every error class: Silent Data Corruption, Benign, Timeout, Trap, Write Text Segment

Comparison of two error-rate vectors
- Precise vector from Def/Use
- Approximated one from BBRs
- Geometric mean over the class deviations
Evaluation - Locality

BBR Locality (all error classes) per Benchmark

Deviation [%]

BC BFD BFE QSORT RDD RDE SHA

0 20 40 60 80 100 120
Evaluation - Locality

![BBR Locality per Error Class](image)

- Silent Data Corruption
- Benign
- Timeout
- Trap
- Write Text Segment

Deviation [%]
Evaluation - Locality

BBRs keep locality of the results for SDCs in combined FM

Memory fault space: Median 0%, 75-quantile mostly 0%

Register fault space: As expected, bad locality of the results
Conclusion

What we did
- We extract the program structure from a program trace
- The extracted structure leads to fault space regions
- Dataflows which cross regions borders will be injected

We could reduce the number of required faults
- Combined fault space -76% with 2.7% deviation
- Even more precise for the memory fault space
- Locality of the results kept regarding silent data corruption

Check it out: FAIL* - Fault Injection Leveraged (https://github.com/danceos/fail)