MorFuzz: Fuzzing Processor via Runtime Instruction Morphing enhanced Synchronizable Co-simulation

Jinyan Xu¹, Yiyuan Liu¹, Sirui He², Haoran Lin¹, Yajin Zhou¹, and Cong Wang²

¹ Zhejiang University
² City University of Hong Kong

August, 2023
Motivation

- Even the most advanced commercial processor is not perfect

<table>
<thead>
<tr>
<th>Product</th>
<th>Last Update</th>
<th>#Errata</th>
<th>#Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel 13th Generation</td>
<td>2023</td>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>Intel 12th Generation</td>
<td>2023</td>
<td>56</td>
<td>10</td>
</tr>
<tr>
<td>Intel 11th Generation</td>
<td>2023</td>
<td>36</td>
<td>10</td>
</tr>
<tr>
<td>Intel 10th Generation</td>
<td>2023</td>
<td>85+145</td>
<td>27+12</td>
</tr>
<tr>
<td>AMD EPYC 9004 Series</td>
<td>2023</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>AMD EPYC 7002 Series</td>
<td>2023</td>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td>AMD EPYC 7003 Series</td>
<td>2022</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>AMD 1st/2nd Ryzen Series</td>
<td>2019</td>
<td>45/60</td>
<td>0</td>
</tr>
</tbody>
</table>
Motivation

• Even the most advanced commercial processor is not perfect

<table>
<thead>
<tr>
<th>Product</th>
<th>Last Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel 13th Generation</td>
<td>2023</td>
</tr>
<tr>
<td>Intel 12th Generation</td>
<td>2023</td>
</tr>
<tr>
<td>Intel 11th Generation</td>
<td>2023</td>
</tr>
<tr>
<td>Intel 10th Generation</td>
<td>2023</td>
</tr>
<tr>
<td>AMD EPYC 9004 Series</td>
<td>2023</td>
</tr>
<tr>
<td><strong>AMD EPYC 7002 Series</strong></td>
<td><strong>2023</strong></td>
</tr>
<tr>
<td>AMD EPYC 7003 Series</td>
<td>2022</td>
</tr>
<tr>
<td>AMD 1st/2nd Ryzen Series</td>
<td>2019</td>
</tr>
</tbody>
</table>
Motivation

Even the most advanced commercial processor is not perfect. The Pentium FDIV bug is a hardware bug affecting the floating-point unit (FPU) of the early Intel Pentium processors. Because of the bug, the processor would return incorrect binary floating point results when dividing certain pairs of high-precision numbers. The bug was discovered in 1994 by Thomas R. Nicey, a professor of mathematics at Lynchburg College.[1] Missing values in a lookup table used by the FPU's floating-point division algorithm led to calculations acquiring small errors. While these errors would in most use-cases only occur rarely and result in small deviations from the correct output values, in certain circumstances these errors can occur frequently and lead to more significant deviations.[2]

The severity of the FDIV bug is debated. Though rarely encountered by most users (Byte magazine estimated that 1 in 9 billion floating point divisions with random parameters would produce inaccurate results),[3] both the flaw and Intel's initial handling of the matter were heavily criticized by the tech community.

---

Table: Product Last Update #Errata #Fixed

<table>
<thead>
<tr>
<th>Product</th>
<th>Year</th>
<th>#Errata</th>
<th>#Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel 13th Generation</td>
<td>2023</td>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>Intel 12th Generation</td>
<td>2023</td>
<td>56</td>
<td>10</td>
</tr>
<tr>
<td>Intel 11th Generation</td>
<td>2023</td>
<td>36</td>
<td>10</td>
</tr>
<tr>
<td>Intel 10th Generation</td>
<td>2023</td>
<td>85+145</td>
<td>27+12</td>
</tr>
<tr>
<td>AMD EPYC 9004 Series</td>
<td>2023</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>AMD EPYC 7002 Series</td>
<td>2023</td>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td>AMD EPYC 7003 Series</td>
<td>2022</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>AMD 1st/2nd Ryzen Series</td>
<td>2019</td>
<td>45/60</td>
<td>0</td>
</tr>
</tbody>
</table>
Processor Fuzzing

Input Generation  Hardware Simulation  State Verification
Processor Fuzzing

```
"10111111
00000101
01110011
01110...
"
```

Seed corpus

Mutator

Input

Mutator

Coverage

Host executable binary

Hardware Simulation

Comparator

State Verification

```
slli a2,a2,0x4
li     a0,0
liui   a2,0xef1
...
...
...
```

```
0612
4501
```

```
elf/binseed
```

Processor Fuzzing

- **Processor RTL code** → **RTL simulator**
- **Seed corpus** → **Mutator** → **Coverage**
- **Input Generation**
  - Seed: "10111111 00110101 01110010 01110..."
  - Assembly:
    - `slli a2,a2,0x4`
    - `or a1,a1,a2`
    - `lui a2,0xef1`
  - Output: `0612 8dd1`
- **Hardware Simulation**
- **State Verification**
- **Comparator**
  - **S_{DUT}** → **S_{ref}**
  - **Bug**
Processor Fuzzing

Input Generation: Processor RTL code -> RTL simulator

Hardware Simulation: Input -> Mutator -> Host executable binary

State Verification: Coverage Instrument

Input: 0612 8dd1

elf/bin -> DUT Software Model

Coverage
Processor Fuzzing

Input Generation

Seed corpus

Processor RTL code

RTL simulator

Mutator

Host executable binary

Coverage

Input

DUT Side

slli a2,a2,0x4
or a1,a1,a2
lui a2,0xef1

RegFile

0x0123456789abcdef
0x0fffffff

State_{DUT}

Ref Model Side

slli a2,a2,0x4
or a1,a1,a2
lui a2,0xef1

RegFile

0x0123456789abcdef
0x0fffffff

State_{REF}

StateDUT

StateREF

Comparator

Bug

RegFile

0x0123456789abcdef
0x0fffffff

Coverage Instrument

ISA simulator

S_{ref}

S_{DUT}
Challenges of Processor Fuzzing

DifuzzRTL:

```
1 start:
   call init_regs
   call init_page_table

2 l1:
   lui x4, 0x40052
   addi x4, x4, -768
   lw x2, 0(x4)

3 l2:
   la x2, l86
   jalr x20, 0(x2)

# ...

4 l86:
   csrrw x6, satp, x5

5 l87:
   blt x25, x6, exit
   # ...

6 exit:
   call signature
```

```
start:
   call init_regs
   call init_page_table

l1:  
   lui x4, 0x40052
   addi x4, x4, -768
   lw x2, 0(x4)

l2:  
   la x2, l86
   jalr x20, 0(x2)

# ...

l86: 
   csrrw x6, satp, x5

l87:  
   blt x25, x6, exit
   # ...

exit: 
   call signature
```
Challenges of Processor Fuzzing

1. Complex Input Grammar

- Read the base address from x4
- Calculate the effective address by adding x4 to the offset
- Load different length of value from the effective address
- Save the value from memory to x2

C1: Processor State

\[ x_4 \in \{\text{memory range}\} \]

C2: Instruction Field

- \( lb, lh, lw, ld \)
- \( lbu, lhu, lwu \), \textbf{Reserved}

C3: Program Semantic

\[ (x_4 + \text{offset}) \% 4 = 0 \]
Challenges of Processor Fuzzing

2. Deceptive Mutation Guidance

Generated Instructions ≠ Executed Instructions

Unexecuted valuable mutations will be discarded
Challenges of Processor Fuzzing

3. Model Implementation Differences

CSR satp has **WARL** (Write Any Values, Reads Legal Values) fields

Divergent control flows make subsequent execution meaningless
Mutating those instructions that are going to be executed

- all mutations are executed, yielding effective coverage
- use runtime context to simplify input generation
Stimulus Template Generation

DifuzzRTL:

```plaintext
11:
   lui x4, 0x40052
   addi x4, x4, -768
   lw x2, 0(x4)
```

MorFuzz:

```plaintext
fuzztext_ls_27:
   # magic inst
   ld x1, RDM_DATA_ADDR(x0)
   # template inst
   lh x??, ??(x??)
```

Magic Instruction
- load a random value with desired type into target register
- processor state mutation primitive

Template Instruction
- blank instruction with dummy fields
- instruction field & program semantic mutation primitive
Runtime Instruction Morphing

<table>
<thead>
<tr>
<th>imm</th>
<th>rs1</th>
<th>funct3</th>
<th>rd</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000100</td>
<td>00001</td>
<td>010</td>
<td>00010</td>
<td>0000011</td>
</tr>
</tbody>
</table>

Field Level Mutation
- identify instruction format
- mutate instruction opcode field

Semantic Level Mutation
- select register with desired type
- calculate the other fields
Commit Stage
- DUT commits control flow info
- REF execute one step
- **cross-check control flow info**
  - match, continue
  - mismatch, report as bug
- REF commits reference write-back data

Judge Stage
- DUT finally commits write-back data
- **cross-check wdata**
  - match, continue
  - mismatch, analysis committed info
  - sync DUT state to REF if permitted
  - otherwise report as bug
## Bug Detected

MorFuzz detected 19 bugs including 17 new bugs, 13 CVEs

<table>
<thead>
<tr>
<th>Processor</th>
<th>Bug Description</th>
<th>CVE/Issue ID</th>
<th>CWE</th>
<th>New Bug</th>
<th>Confirmed</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rocket</td>
<td><strong>B1:</strong> Treat aes64ksli with rcon greater than 0xA as valid</td>
<td>CVE-2022-34632</td>
<td>CWE-327</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B2:</strong> Error in condition of the rocc_illegal signal</td>
<td>Issue #2980</td>
<td>CWE-1281</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B3:</strong> The vsstatus.xs is writable</td>
<td>CVE-2022-34627</td>
<td>CWE-732</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BOOM</td>
<td><strong>B4:</strong> Incorrect exception type when a PMA violation</td>
<td>CVE-2022-34636</td>
<td>CWE-1202</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B5:</strong> Incorrect exception type when a PMP violation</td>
<td>CVE-2022-34641</td>
<td>CWE-1198</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B6:</strong> Floating-point instruction with invalid rm field does not raise exception</td>
<td>Issue #458</td>
<td>CWE-391</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B7:</strong> Floating-point instruction with invalid frn does not raise exception</td>
<td>Issue #492</td>
<td>CWE-391</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CVA6</td>
<td><strong>B8:</strong> Crafted or incorrectly formatted sfence.vma instructions are executed</td>
<td>CVE-2022-34633</td>
<td>CWE-1242</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B9:</strong> Crafted or incorrectly formatted dret instructions are executed</td>
<td>CVE-2022-34634</td>
<td>CWE-1242</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B10:</strong> Non-standard fence instructions are treated as illegal</td>
<td>CVE-2022-34639</td>
<td>CWE-1209</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B11:</strong> The mstatus.sd field does not update immediately</td>
<td>CVE-2022-34635</td>
<td>CWE-1199</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B12:</strong> The value of mtval/stval after ecall/ebreak is incorrect</td>
<td>CVE-2022-34640</td>
<td>CWE-755</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B13:</strong> Incorrect exception type when a PMA violation</td>
<td>CVE-2022-34636</td>
<td>CWE-1202</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B14:</strong> Incorrect exception type when a PMP violation</td>
<td>CVE-2022-34641</td>
<td>CWE-1198</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B15:</strong> Incorrect exception type when accessing an illegal virtual address</td>
<td>CVE-2022-34637</td>
<td>CWE-754</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B16:</strong> Improper physical PC truncate</td>
<td>Issue #901</td>
<td>CWE-222</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B17:</strong> Incorrect lr exception type</td>
<td>CVE-2022-37182</td>
<td>CWE-754</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Spike</td>
<td><strong>B18:</strong> The component mcontrol.action contains the incorrect mask</td>
<td>CVE-2022-34642</td>
<td>CWE-787</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td><strong>B19:</strong> Incorrect exception priority when accessing memory</td>
<td>CVE-2022-34643</td>
<td>CWE-754</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Coverage

4.4× than DifuzzRTL, 3.1× than riscv-torture, 1.6× than riscv-dv

~30 min
MorFuzz is a novel Processor Fuzzer

- detect architecture functional bugs automatically
  - instruction morphing effectively guides fuzzing
  - state synchronization eliminates false positive
- thorough evaluation on real world processors
  - faster & higher coverage than SOTA
  - detected 19 bugs with 13 CVEs assigned
- battle-hardened and open-source
  - deployed in a undergraduate CPU design course (50+ students)
  - 1st Place in HACK@DAC 2023
  - [https://github.com/sycuricon/MorFuzz](https://github.com/sycuricon/MorFuzz)
Thank You!

Jinyan Xu
phantom@zju.edu.cn