Poster: *TheHuzz*: Instruction Fuzzing of Processors Using Golden-Reference Models for Finding Software-Exploitable Vulnerabilities

Rahul Kande[†], Addison Crump[†], Garrett Persyn[†], Patrick Jauernig[*], Ahmad-Reza Sadeghi[*], Aakash Tyagi[†], and Jeyavijayan Rajendran[†]
[†]Texas A&M University, USA, [*]Technische Universität Darmstadt, Germany
[†]{rahulkande, addisoncrump, gpersyn, tyagi, jv.rajendran}@tamu.edu,
[*]{patrick.jauernig, ahmad.sadeghi}@trust.tu-darmstadt.de

Abstract

The increasing complexity of modern processors poses many challenges to existing hardware verification tools and methodologies for detecting security-critical bugs. Recent attacks on processors have shown the fatal consequences of uncovering and exploiting hardware vulnerabilities.

Fuzzing has emerged as a promising technique for detecting software vulnerabilities. Recently, a few hardware fuzzing techniques have been proposed. However, they suffer from several limitations, including non-applicability to commonly-used hardware description languages (HDLs) like Verilog and VHDL, the need for significant human intervention, and inability to capture many intrinsic hardware behaviors, such as signal transitions and floating wires.

In this paper, we present the design and implementation of a novel hardware fuzzer, *TheHuzz*, that overcomes the aforementioned limitations and significantly improves the state of the art. We analyze the intrinsic behaviors of hardware designs in HDLs and then measure the coverage metrics that model such behaviors. *TheHuzz* generates assembly-level instructions to increase the desired coverage values, thereby finding many hardware bugs exploitable from software. We evaluate *TheHuzz* on four popular open-source processors and achieve 1.98× and 3.33× the speed compared to the industry-standard random regression approach and the state-of-the-art hardware fuzzer, DifuzzRTL, respectively. Using *TheHuzz*, we detected 11 bugs in these processors, including 8 new bugs, and we demonstrate exploits using the detected bugs. We also show that *TheHuzz* overcomes the limitations of formal verification tools from the semiconductor industry by comparing its findings to those discovered by the Cadence JasperGold tool.

I. Main Content

This research [1] is recently published in USENIX Security 2022. The original abstract and author list are shown above. We post the paper link with the conference version[1].

References

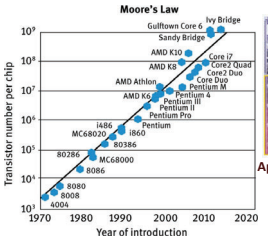
[1] R. Kande, A. Crump, G. Persyn, P. Jauernig, A.-R. Sadeghi, A. Tyagi, and J. Rajendran, "TheHuzz: Instruction Fuzzing of Processors Using Golden-Reference Models for Finding Software-Exploitable Vulnerabilities," *USENIX Security Symposium*, pp. 3219–3236, 2022.

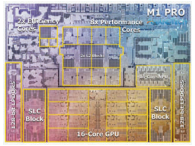
[1] https://www.usenix.org/conference/usenixsecurity22/presentation/kande

Poster: *TheHuzz*: Instruction Fuzzing of Processors Using Golden-Reference Models for Finding Software-Exploitable Vulnerabilities

Rahul Kande†, Addison Crump†, Garrett Persyn†, Patrick Jauernig*, Ahmad-Reza Sadeghi*, Aakash Tyagi†, Jeyavijayan Rajendran†
†Texas A&M University, College Station, USA, *Technische Universität Darmstadt, Germany.
{rahulkande, addisoncrump, gpersyn, tyagi, jv.rajendran}@tamu.edu, {patrick.jauernig, ahmad.Sadeghi}@trust.tu-darmstadt.de

Motivation





Apple's new chip: 33.7bn transistors

Size and complexity

Efficiency of verification tools

Arm CPUs impacted by rare side-channel attack

Apple iOS 14.7.1: Urgent Update Aims To Fix Critical iPhone Problem

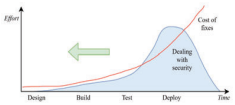
Samsung, Android Phones Exposed to Hackers Due to Qualcomm Chip Bugs: Updates, Fixes and More

SPECTRE MELTDOWN FORESHADOW

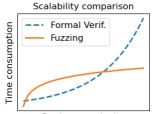
Limitations of Existing Tools

Technique	Fast	Coverage	Scalable	Automated
Manual inspection	✗	✗	✗	✗
Formal verification	✗	✓	✗	✗
Regression	✗	✗	✓	✓

Our Solution: HARDWARE FUZZING



Shift left in bug detection



Fast Coverage Scalable Automated

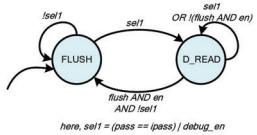
Specification comparison

Logic aware

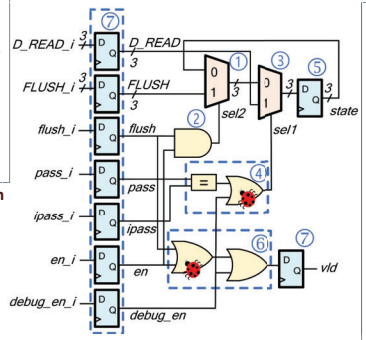
Case Study: Hardware Coverage

```
// combinational logic for vld register
vld := debug_en |(flush |en) // Bug b2

// select signal for mux
val sel1 = Wire(Bool())
sel1 :=((pass===ipass)|debug_en) // Bug b1

// flush logic
val state_f = Wire(UInt(3.W))
when (flush & en){
   state_f := FLUSH
} otherwise {
   state_f := state
}

state :=(!sel1 &state_f) |(sel1 & D_READ)
```

Chisel code of the case study design



Finite state machine (FSM) of the case study design

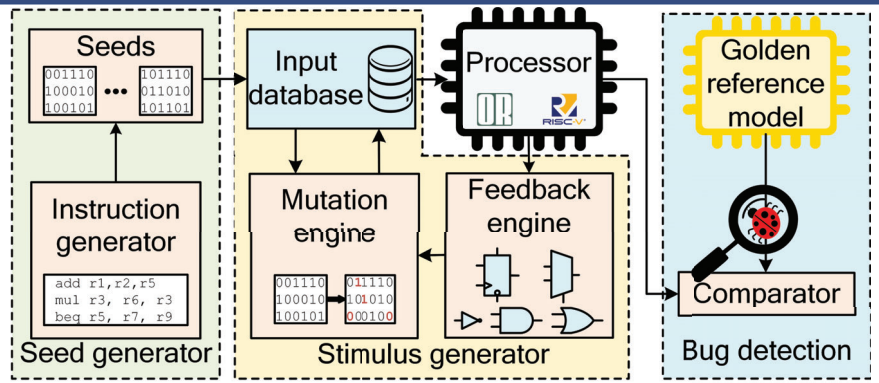


Hardware design for the case study

Hardware Coverage Types

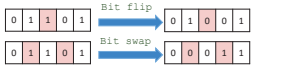
- **Branch coverage**
  - Select signal values of MUX ①
- **Condition coverage**
  - Input signal values of condition ②
- **Expression coverage**
  - Input signal values of the logic ③, ⑥
- **FSM coverage**
  - State transitions of the 3-bit reg ⑤
- **Toggle coverage**
  - Bit-toggle of the flip-flops ⑤, ⑦
- **Statement coverage**
  - Each line in the HDL source code

TheHuzz[1]: Overview




```
int main(void)
{
  asm volatile ( "SRLIW x16,x29,1" );
  asm volatile ( "MUL x3,x21,x7" );
  asm volatile ( "SRL x21,x15,x17" );
  asm volatile ( "SLTI x3,x1,-12" );
  asm volatile ( "ADDI x2,x25,146" );
  • • •
  return 0;
}
```

Seed input: C program



Mutation techniques to create new inputs

```
@00000000 00000093 00000113 00000193 00000213
@00000001 00000293 00000313 00000393 00000413
• • •
@0000025b 00000013 00000013 00000013 00000013
@0000025c 00000013 00000013 001ed81b 027a81b3
@0000025d 0117dab3 84198003 4197de1b ff40a193
• • •
@00000297 ffffee7c fffff1cc 00000000 00000000
```

Binary executable

```
CO:       1379 [1] pc=[00000000800025c8] W[r16=0000000000000000][1]
R[r29=0000000000000000] R[r 0=0000000000000000] inst=[001ed81b]
DASM(001ed81b)
C0:       1380 [1] pc=[00000000800025cc] W[r 3=0000000000000000][0]
R[r21=0000000000000000] R[r 7=0000000000000000] inst=[027a81b3]
DASM(027a81b3)
• • •
```

Hardware simulation trace output

```
core   0: 0x0000000000001000 (0x00000297) auipc   t0, 0x0
3 0x0000000000001000 (0x00000297) x 5 0x0000000000001000
core   0: 0x0000000000001004 (0x02028593) addi    a1, t0, 32
3 0x0000000000001004 (0x02028593) x11 0x0000000000001020
Mismatches ignored: {'2':4, '2_3':67,
'6':8}
core   0: 0x0000000000001008 (0xf1402573) csrr    a0, mhartid
• • •
```

Golden model emulation trace output

```
file,incr_cov,cov,line,branch,cond,fsm,tgl
0,161027,161027,7117,8589,4982,28,140311
1,25041,186068,7151,9291,5656,36,163934
2,21353,207421,7180,9555,5895,44,184747
3,22044,229465,7245,9848,6130,47,206195
4,10606,240071,7260,10016,6284,53,216458
5,6242,246313,7263,10086,6351,53,222560
6,3952,250265,7283,10144,6394,59,226385
• • •
```

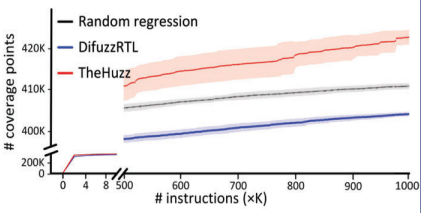
Coverage output

```
No errors found in 0_comp_out.log
--------------------------------
mismtaches ignored: ['2_2']
No errors found in 1_comp_out.log
--------------------------------
[2_comp_out.log][87][ERROR: **PCs
Mismatch**]
• • •
--------------------------------
Total files = 2257, Files with errors = 1
Mismatches ignored: {'2':4, '2_3':67,
'6':8}
```

Bug detection output

Results

- **Fuzzed 4 different real-world open-source processors from RISC-V and OpenRISC ISAs**
  - RISC-V Rocket core[2] & Ariane[3]
  - OR MOR1KX[4] & OR1200[5]
- **Faster coverage than the state-of-the-art**



- **Detected 11 bugs including 8 new bugs**
  - Filed CVE's for 5 bugs

Ariane bug: Cannot recognize all types of fence instruction

MOR1KX bugs: Incorrect access permissions for control status registers EPCR and EEAR

OR1200 bug: Incorrect implementation of overflow bit

- **Developed two exploits**
  - Privilege escalation using MOR1KX bugs
  - Arbitrary code execution using Ariane bugs

Future Work

- While TheHuzz is faster than traditional verification techniques and capable of detecting new vulnerabilities, there is still scope for improvement in terms of the fuzzing speed, achieving close to 100% coverage, and fuzzing different types of hardware designs.
- Thus our future focus in on the following:
  - **FPGA fuzzing:** Emulate hardware on FPGA to improve the speed of fuzzing.
  - **Formal fuzzing:** Leverage capabilities of formal tools to improve the coverage achieved by TheHuzz.
  - **Fuzzing non-processor designs:** Extend TheHuzz to fuzz non-processor designs such as cryptographic accelerators.

Contact Information

RAHUL KANDE
Ph.D. in Computer Engineering
Texas A&M University
rahulkande@tamu.edu
https://www.rahulkande.com/
Ph: +1 979-739-8914
SETH research lab: https://seth.engr.tamu.edu/

SCAN ME

Published at USENIX Security Symposium, 2022

References

[1] R. Kande, A. Crump, G. Persyn, P. Jauernig, A.-R. Sadeghi, A. Tyagi, and J. Rajendran, TheHuzz: Instruction Fuzzing of Processors Using Golden-Reference Models for Finding Software-Exploitable Vulnerabilities, USENIX Security Symposium'22
[2] https://github.com/chipsalliance/rocket-chip
[3] https://github.com/openhwgroup/cva6
[4] https://github.com/openrisc/mor1kx
[5] https://github.com/openrisc/or1200

Acknowledgements

Our research work was partially funded by the US Office of Naval Research, by Intel's Scalable Assurance Program, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), and by the German Federal Ministry of Education and Research and the Hessian State Ministry for Higher Education, Research and the Arts within ATHENE.

We thank Kevin Laeufer (UC Berkeley), Jaewon Hur (Seoul National University), and TAMU HRPC for their support. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government.