
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
Motivation

- iOS 14.7.1: Apple Issues Urgent iPhone Update With Important Security Fixes
  - Kate O'Flaherty, Senior Contributor @ Straight Talking Cyber
  - Jul 26, 2021, 02:15pm EDT

- Samsung, Android Phones Exposed to Hackers Due to Qualcomm Chip Bugs: Updates, Fixes and More
  - technewsgrace del valle
  - Feb 7, 2021 09:59 AM EDT

- Arm CPUs impacted by rare side-channel attack
  - Arm issues guidance to developers to mitigate new "straight-line speculation" attack.
  - Written by Catalin Cimpanu on June 9, 2020

- 63% of organizations face security breaches due to hardware vulnerabilities

---

https://www.zdnet.com/article/arm-cpus-impacted-by-rare-side-channel-attack/
https://www.techrepublic.com/article/63-of-organizations-face-security-breaches-due-to-hardware-vulnerabilities/
Motivation

• 113 new H/W CWEs since 2020 by MITRE

https://www.zdnet.com/article/arm-cpus-impacted-by-rare-side-channel-attack/
https://www.techrepublic.com/article/63-of-organizations-face-security-breaches-due-to-hardware-vulnerabilities/

63% of organizations face security breaches due to hardware vulnerabilities

iOS 14.7.1: Apple Issues Urgent iPhone Update With Important Security Fixes
Kate O'Flaherty Senior Contributor @ Straight Talking Cyber
Jul 26, 2021, 02:15pm EDT

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

Arm CPUs impacted by rare side-channel attack
Arm issues guidance to developers to mitigate new "straight-line speculation" attack.
Written by Catalin Cimpanu on June 9, 2020
<table>
<thead>
<tr>
<th>Technique</th>
<th>Fast</th>
<th>Coverage</th>
<th>Scalable</th>
<th>Automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual inspection</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Formal verification[^1]</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Regression testing</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Hardware Fuzzing</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Case Study: Ariane Cache Controller
Case Study: Ariane Cache Controller
Case Study: Ariane Cache Controller

- **MUX ③** to flush the cache
- **MUX ②** to protect debug reads using password
- **Combinational logic ①** to generate valid signal
Case Study: Ariane Cache Controller

**Bug in MUX ③:** Cache can be flushed even when it is not enabled

**Bug in MUX ②:** Debug password protection is bypassed

**Bug in Combinational logic ①:** Cache is activated even when the en_i signal is not enabled
Case Study: Ariane Cache Controller

Bug in MUX ③:
Cache can be flushed even when it is not enabled.

Bug in MUX ②:
Debug password protection is bypassed.

Bug in Combinational logic ①:
Cache is activated even when the en_i signal is not enabled.

Cover different combinations of values for input signals of select logic.

Cover different combinations of values for input signals of select logic.

Cover different combinations of values for input signals of combinational logic.
Existing Hardware Fuzzers

RFUZZ [2]

- Novelty: first hardware fuzzer
- Can fuzz any hardware design
- Covers select signals of MUXs coded as control logic
Existing Hardware Fuzzers

RFUZZ [2]

- Novelty: first hardware fuzzer
- Covers select signals of MUXs
described as control logic
- Doesn’t detect MUX
- Doesn’t cover activity in combinational logic and flip-flops
- Computationally expensive
- Does not scale to large designs
Existing Hardware Fuzzers

RFUZZ [2]

- Novelty: first hardware fuzzer
- Covers select signals of MUXs coded as control logic
- Doesn’t detect MUX
- Doesn’t cover activity in combinational logic and flip-flops
- Computationally expensive
- Does not scale to large designs

MUX as control logic

```plaintext
if (s)  
  out <= a;
else  
  out <= b;
```

MUX as combinational logic

```plaintext
assign out => (s&a) | (!s&b)
```
Existing Hardware Fuzzers

RFUZZ [2]

- Novelty: first hardware fuzzer
- Can fuzz any hardware design
- Covers select signals of MUXs
- Doesn't detect MUX
- Doesn't cover activity in combinational logic and flip-flops
- Computationally expensive
- Does not scale to large designs

MUX as control logic

```plaintext
if (s)
  out <= a;
else
  out <= b;
```

MUX as combinational logic

```plaintext
assign out => (s&a) | (!s&b)
```
Existing Hardware Fuzzers

**RFUZZ [2]**

- Novelties: first hardware fuzzer
- Can fuzz any hardware design
- Covers select signals of MUXs coded as control logic

**DifuzzRTL [3]**

- Novelties: Resolved scalability issue of RFUZZ
- Covers registers driving select logic of MUXs coded as control logic
  \[ \Rightarrow \text{covers bug in } 3 \]

- Doesn't detect MUX ②
- Doesn't cover activity in combinational logic and flip-flops
- Computationally expensive
- Does not scale to large designs

\{flush, en\} == \{00, 01, 10, 11\}
Existing Hardware Fuzzers

**RFUZZ [2]**
- sel2 == \{0,1\}

**DifuzzRTL [3]**
- \{\text{flush, en}\} == \{00,01,10,11\}

**Novelty:** first hardware fuzzer
- Doesn’t detect MUX
- Doesn’t cover activity in combinational logic and flip-flops
- Computationally expensive
- Does not scale to large designs

**Novelty:** Resolved scalability issue of RFUZZ
- Covers registers driving select logic of MUXs coded as control logic
- Doesn’t detect MUX
- Doesn’t cover activity in combinational logic and flip-flops
- Bug comparison at end of program
Existing Hardware Fuzzers

HyperFuzzing [4]

\[
\begin{align*}
\psi & ::= \forall \pi. \psi | \varphi \\
\varphi & ::= \text{AP}_{\pi_1, \ldots, \pi_k} | \neg \varphi | \varphi \land \varphi | \varphi \lor \varphi \\
& | Y \varphi | O \varphi | H \varphi | \varphi S \varphi
\end{align*}
\]

- Novelty: New semantics for SoC security properties
- Fuzzer accelerates property checking
Existing Hardware Fuzzers

HyperFuzzing [4]

\[ \psi ::= \forall \pi. \psi \mid \varphi \]
\[ \varphi ::= \text{AP}_{\pi_1, \ldots, \pi_k} \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid Y \varphi \mid O \varphi \mid H \varphi \mid \varphi S \varphi \]

- Novelty: New semantics for SoC security properties
- Fuzzer accelerates property checking
- Not applicable to general hardware like FSMs or combinational logic
- Need to write security properties
- Supports Verilator simulator only
Existing Hardware Fuzzers

HyperFuzzing [4]

- Novelty: converts HW to SW for fuzzing
- Existing software fuzzers can be integrated to fuzz hardware

+ Novelty: New semantics for SoC security properties
+ Fuzzer accelerates property checking
+ Not applicable to general hardware like FSMs or combinational logic
+ Need to write security properties
+ Supports Verilator simulator only

Trippel et al. [5]

+ Novelty: converts HW to SW for fuzzing
+ Existing software fuzzers can be integrated to fuzz hardware
Existing Hardware Fuzzers

HyperFuzzing [4]

- Novelty: converts HW to SW for fuzzing
- Existing software fuzzers can be integrated to fuzz hardware
- Supports Verilator like simulator only
- Does not support all Verilog constructs like latches, floating wires

Modified AFL fuzzer → Property checker

Trippel et al. [5]

- Novelty: converts HW to SW for fuzzing
  Existing software fuzzers can be h...
- Supports Verilator like simulator only
- Does not support all Verilog constructs like latches, floating wires

RTL code

ψ ::= ∀π. ψ | φ
φ ::= AP_{π_1,...,π_k} | ¬φ | φ ∧ φ | φ ∨ φ
| Y φ | O φ | H φ | φ S φ
## Summary of Existing Techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Hardware Components Covered</th>
<th>Scalability (Largest Design’s LOC)</th>
<th>Applicability</th>
<th>Simulator</th>
<th># Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFUZZ [2]</td>
<td>Select signals of some MUXs</td>
<td>5-stage Sodor core (4,088)</td>
<td>Any RTL</td>
<td>Any</td>
<td>0</td>
</tr>
<tr>
<td>DifuzzRTL [3]</td>
<td>Registers driving select signals of some MUXs</td>
<td>Boom (12,956 (Scala))</td>
<td>Processors</td>
<td>Any</td>
<td>16</td>
</tr>
<tr>
<td>HyperFuzzing [4]</td>
<td>Inserted properties</td>
<td>SHA crypto engine (1,196)</td>
<td>SoCs</td>
<td>Verilator</td>
<td>0</td>
</tr>
</tbody>
</table>
Coverage Metrics of TheHuzz
Coverage Metrics of *TheHuzz*

- **Statement**: All statements in RTL code
- **Branch**: Control signals (sel1, sel3 of 3, 1)
- **Toggle**: 0 → 1/1 → 0 transitions of flip-flops
- **FSM**: States & state transitions of FSM 5
- **Condition**: Control path combinational logic (AND gate in 2)
- **Expression**: Data path combinational logic (gates in 4 & 6)
Design of *TheHuzz*

- Seeds
- Instruction generator
  - add r1, r2, r5
  - mul r3, r6, r3
  - beq r5, r7, r9
- Input database
- Processor
- Mutation engine
- Feedback engine
- Golden reference model
- Comparator
- Bug detection
Design of TheHuzz

1. Initial set of inputs (seeds) are randomly generated.
2. Inputs are program files with assembly instructions.
3. Target processor is simulated using inputs from database.
4. A GRM is also simulated with the same input.
5. Coverage data is analyzed to determine if new logic is explored.
6. Current inputs are mutated to generate new inputs.
7. Output of target and GRM are compared to detect bugs.

---

**Seeds**

```
001110 101110
100010 011010
100101 011011
```

**Instruction generator**

```
add r1, r2, r5
mul r3, r6, r3
beq r5, r7, r9
```

**Input database**

**Mutation engine**

```
001110 011100
100010 101010
100101 101101
```

**Processor**

**Feedback engine**

**Comparator**

**Golden reference model**

**Bug detection**
<table>
<thead>
<tr>
<th>Processor</th>
<th>Bug Description</th>
<th>CVE/CWE</th>
<th>Location</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ariane (cva6)[6]</td>
<td>Incorrect implementation of logic to detect the FENCE.I instruction.</td>
<td>CWE-440</td>
<td>Decoder</td>
<td>Branch</td>
</tr>
<tr>
<td>RISC-V [8]</td>
<td>Failure to detect cache coherency violation</td>
<td>CWE-1202</td>
<td>Cache controller</td>
<td>FSM</td>
</tr>
<tr>
<td>2.07 ×10⁴ LOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mor1kx [7]</td>
<td>Read/write access check not implemented for privileged reg.</td>
<td>CVE-2021-41614</td>
<td>Register file</td>
<td>Condition</td>
</tr>
<tr>
<td>OpenRISC [9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.21 ×10⁴ LOC</td>
<td>Incomplete implementation of EEAR register write logic</td>
<td>CVE-2021-41613</td>
<td>Register file</td>
<td>Condition</td>
</tr>
<tr>
<td>or1200 [7]</td>
<td>Incomplete update logic of overflow bit for MSB/MAC instrs.</td>
<td>CVE-2021-40506</td>
<td>ALU</td>
<td>Toggle</td>
</tr>
<tr>
<td>OpenRISC [9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.16 ×10⁴ LOC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor</td>
<td>Bug Description</td>
<td>CVE/CWE</td>
<td>Location</td>
<td>Coverage</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Ariane (cva6)[6]</td>
<td>Incorrect implementation of logic to detect the FENCE.I instruction.</td>
<td></td>
<td></td>
<td>2.07 ×10^4 LOC</td>
</tr>
<tr>
<td>RISC-V [8]</td>
<td>Failure to detect cache coherency violation</td>
<td>CWE-440</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mor1kx [7]</td>
<td>Read/write access check not implemented for privileged reg.</td>
<td>CVE-2021-41614</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenRISC [9]</td>
<td>Incomplete implementation of EEAR register write logic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or1200 [7]</td>
<td>Incomplete update logic of overflow bit for MSB/MAC instrs.</td>
<td>CVE-2021-40506</td>
<td>ALU</td>
<td>Toggle</td>
</tr>
<tr>
<td>OpenRISC [9]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TheHuzz detected 11 bugs including 8 new bugs, 5 CVEs**

**Exploit 1:** Arbitrary Code Execution on Ariane

**Exploit 2:** Privilege Escalation on mor1kx
Coverage Results

- **Rocket core:** RISC-V, 32-bit, 5-stage pipelined, 6.65 x 10^4 coverage points
- **1.98x** and **3.33x** the speed of random regression & DifuzzRTL
Conclusion

• Our hardware fuzzer, *TheHuzz* is
  • **Compatible**: Chisel/.v/.vhdl, any commercial hardware simulator
  • **Automated**: Design agnostic
  • **Practical**: Simple to run (50+ students trained)
  • **Efficient**: Detected 11 bugs, higher coverage than existing techniques

• We demonstrated the security impact of bugs through two exploits

• Future work
  • Extend TheHuzz to support FPGA emulation
  • Fuzzing non-processor designs
  • Fuzzing parametric properties of hardware
  • Fuzzing to detect side-channel vulnerabilities