Hardware is the New Software: Finding Exploitable Bugs in Hardware Designs

Hardware Security @ UNC

Cynthia Sturton
Meltdown
Almost every modern processor is affected

Spectre
Almost every modern processor is affected

Foreshadow
Can expose the cryptographic keys that protect the integrity of SGX enclaves
[WARNING] Intel Skylake/Kaby Lake processors: broken hyper-threading

- To: debian-user@lists.debian.org, debian-devel@lists.debian.org
- Subject: [WARNING] Intel Skylake/Kaby Lake processors: broken hyper-threading
- From: Henrique de Moraes Holschuh <mh@debian.org>
- Date: Sun, 25 Jun 2017 09:19:36 -0300

The Cyrix 6x86 Coma Bug

The Pentium F00F Bug
by Robert R. Collins

Hypervisor headaches: Hosts hosed by x86 exception bugs

Microsoft, Xen, KVM et al need patches

By Richard Chirgwin 13 Nov 2015 at 04:56

Various hypervisors and operating systems are scrambling to patch
The Pentium F00F Bug
by Robert R. Collins

The Cyrix 6x86 Coma Bug

Hypervisor headaches: Hosts hosed by x86 exception bugs
Microsoft, Xen, KVM et al need patches
By Richard Chirgwin 13 Nov 2015 at 04:56

C. Sturton
Software Security

- Buffer overflow
- Integer overflow
- Format string
- SQL injection
- Directory crawling
- Cross-site scripting
- Cross-site request forgery
Software Security

- Buffer overflow
- Integer overflow
- Format string
- SQL injection
- Directory crawling
- Cross-site scripting
- Cross-site request forgery

→ stack smashing
→ heap overflow
→ return to libC
→ return oriented programming
→ jump oriented programming
fuzzing
program analysis
secure languages
Hardware Security

- Secure languages
- Manual review
Hardware Security

- Side channels
- Transient faults

→ Extract private keys
How can we identify vulnerabilities and their exploits in hardware designs?
How can we identify vulnerabilities and their exploits in hardware designs?
How can we identify vulnerabilities and their exploits in hardware designs?
How can we identify vulnerabilities and their exploits in hardware designs?
// Internal wires and regs

wire [3y-1:0] from_rfa;
wire [3v-1:0] from_rvb;
wire [3w-1:0] rf_ostra;
wire [3z-1:0] rf_addew;
wire [3u-1:0] rf推开;
wire rf_we;
wire rpr_valid;
wire rfr_csa;
wire rfr_cea;
wire rfr_en;
wire rf_we lawful;
wire rf_we allow;

// Logic to restore output on HFA after debug unit has read out via SCR if.
// Problem here that the incorrect output would be on HFA after debug unit
// had read out - this is bad if that output is relied upon by someone
// steps for next instruction. We simply save the last address for rf_a and
// and re-read it whenever the SCR select goes low, so we must remember
// the last address and generate a signal for failing edge of SCR ca.
// -- Sturton

// Detect falling edge of SCR select
reg rpr_dnc;
wire rpr_csa_fe;
// Track RF A's address each time it's enabled
reg [3y-1:0] addr_a last;
always @(posedge rpr)
if (rf_xno & |rpr_csa_fe | (du_read & rpr_cs))
  addr_a last <= addr;
always @(posedge rfr)
  rpr_csa_fe <= rpr_csa & du_read;
  assign rpr_csa_fe = rpr_csa & du_read;

// SCR access is valid when rpr_csa is asserted and
// SCR address matches SCR addresses:
assign rpr_valid = rpr_csa & (scr_addr[10:1] == 'GH1000_SCR_RF);

// SCR data output is always from RF A
assign rpr_d_o = from_rfa;

// Operand A comes from RF or from saved A register
assign data_a = from_rfa;

// Operand B comes from RF or from saved B register
assign data_b = from_rvb;
Vulnerabilities: An Analysis of Exploitable Bugs

[ASPLOS 2015]
Under a highly specific and detailed set of internal timing conditions, the processor core may incorrectly fetch instructions ...

Potential effect: **unpredictable system behavior**
AMD Processors 2007–2013

301 errata
AMD Processors 2007–2013

28 security critical
Classifying Exploitable Bugs

- Exception Related
- Incorrect Results
- Memory Access
- Incorrect Instruction
- Register Related
Manually Writing Security Properties
Writing Security Properties

Specification Documents (14):
Writing Security Properties

Specification Documents (14):

φ: Processor mode changes from low privilege to high privilege only by an exception or a reset.
Writing Security Properties

Specification Documents (14):

AMD Errata (3):
Writing Security Properties

Specification Documents (14): φ φ φ φ φ φ φ φ φ φ φ φ φ φ

AMD Errata (3): φ φ φ

φ: When a register changes, it must be specified as the target of the instruction.
Writing Security Properties

Specification Documents (14):

AMD Errata (3):

Initial Evaluation (1):
Writing Security Properties

Specification Documents (14):

AMD Errata (3):

Initial Evaluation (1):

φ: The instruction does not change in the pipeline.
Security Property Specification
Traces

- $p_1$
- $p_2$
- ...

Patterns

Miner

Properties

- $\varphi_1$
- $\varphi_2$
- ...
- $\varphi_n$

C. Sturton
**ctrl:**

```
0 0 1 0 1 0 1 0 0
```

**Traces**

```
r[0] ^ r[1] ^ ... ^ r[n]
```

**Pattern**

The Traces are fed into a **Miner** which outputs a property in the form of:

```
```
Example of Exploitable Bug

Example of Exploitable Bug

```plaintext
(a < b);
```
Traces \xrightarrow{SCI Finder} \text{Security Errata}

- $\varphi_1$
- $\varphi_2$
- ... 
- $\varphi_n$
1. Known bugs → Demonstrate exploit → Security properties
1. Known bugs → Demonstrate exploit → Security properties
2. Machine learning → Additional security properties
SCIFinder

Traces

Miner [Daikon]

Bug Classification [Manual]

Security Property Identification

Initial Security Properties

Logistic Regression

Security Properties

Processor Errata

Properties

Security Bugs

C. Sturton
SCIFinder

Miner [Daikon]

Bug Classification [Manual]

Security Property Identification

Security Bugs

Initial Security Properties

Logistic Regression

Properties

Traces

Processor Errata

Security Properties

C. Sturton
SCIFinder

Miner [Daikon] → Properties

Security Property Identification → Initial Security Properties

Security Bugs → Logistic Regression

Traces

Bug Classification [Manual] → Security

Processor Errata

Properties

Security Properties

C. Sturton
Evaluation

- How well does SCIFinder identify security properties?
- Will the generated properties find security vulnerabilities?
OR1200

- 26GB trace data
- 17 programs
- full instruction coverage

SCI Finder

\( \varphi_1 \)

\( \varphi_2 \)
OR1200

- 26GB trace data
- 17 programs
- full instruction coverage

17 bugs
- 3 RISC architectures
- processor core

C. Sturton
OR1200
- 26GB trace data
- 17 programs
- full instruction coverage

17 bugs
- 3 RISC architectures
- processor core

87 properties
- 54 bug driven
- 33 model driven

SCI Finder

\( \varphi_1 \)
\( \varphi_2 \)
Properties

Bug Driven

- 54 properties from 17 bugs
- 47% false discovery rate
Properties

Bug Driven

- 54 properties from 17 bugs
- 47% false discovery rate
Properties

Bug Driven

- 54 properties from 17 bugs
- 47% false discovery rate

Model Driven

- 33 additional properties
- 27% false discovery rate
Comparison to State of the Art

- found in step 1: 11
- found in step 2: 8
- property not mined: 2
- not recognized as security critical: 1

C. Sturton
Comparison to State of the Art

11
8
2
1
3

New properties
Finding and Exploiting Property Violations

[FMS 2018, MICRO 2018]
Problem Statement

Given \( \varphi \) and a processor design

- Can we find a violation of \( \varphi \) ?
- How do we reach the violating state?
- Can the violating state be exploited?
Existing Tools

Simulation Based Testing

```c
// Output logic
// Depending on the state
// output is determined from:
// valid
// change the quarter bit
// end
if (present_state == sig4)
  angle += 1
else if (present_state == sig3)
  angle += 4
else if (present_state == sig2)
  angle += 8
else if (present_state == sig1)
  angle += 16
else
  angle = f(0)
end
```
Existing Tools

Simulation Based Testing

Model Checking
Problem Statement

Given $\varphi$ and a processor design

- Can we find a violation of $\varphi$?
- How do we reach the violating state?
- Can the violating state be exploited?

<table>
<thead>
<tr>
<th>testing</th>
<th>model checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Symbolic Execution

```
if (reset)
    count = 0;
else
    count = count + 1;

if (count > 3)
    ERROR;
```

<table>
<thead>
<tr>
<th>symbolic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset := ( r_0 )</td>
</tr>
<tr>
<td>count := ( c_0 )</td>
</tr>
</tbody>
</table>
Symbolic Execution

```c
if (reset)
    count = 0;
else
    count = count + 1;
if (count > 3)
    ERROR;
```

<table>
<thead>
<tr>
<th>path condition</th>
<th>symbolic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>reset := (r_0), count := (c_0)</td>
</tr>
</tbody>
</table>
Symbolic Execution

if (reset)
    count = 0;
else
    count = count + 1;

if (count > 3)
    ERROR;

<table>
<thead>
<tr>
<th>path condition</th>
<th>symbolic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>reset := ( r_0 )</td>
</tr>
<tr>
<td></td>
<td>count := ( c_0 )</td>
</tr>
</tbody>
</table>
Symbolic Execution

if (reset)
    count = 0;
else
    count = count + 1;

if (count > 3)
    ERROR;

path condition
True
symbolic state
reset := r_0
count := c_0

r_0 = 0
Symbolic Execution

```c
if (reset)
    count = 0;
else
    count = count + 1;

if (count > 3)
    ERROR;
```

<table>
<thead>
<tr>
<th>path condition</th>
<th>symbolic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>reset := r₀</td>
</tr>
<tr>
<td></td>
<td>count := c₀</td>
</tr>
<tr>
<td>r₀ = 0</td>
<td>reset := r₀</td>
</tr>
<tr>
<td></td>
<td>count := c₀ + 1</td>
</tr>
</tbody>
</table>
Symbolic Execution

if (reset)
  count = 0;
else
  count = count+1;

if (count > 3)
  ERROR;

<table>
<thead>
<tr>
<th>path condition</th>
<th>symbolic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>reset := ( r_0 )</td>
</tr>
<tr>
<td></td>
<td>count := ( c_0 )</td>
</tr>
<tr>
<td>( r_0 = 0 )</td>
<td>reset := ( r_0 )</td>
</tr>
<tr>
<td>( c_0 + 1 &gt; 3 )</td>
<td>count := ( c_0 + 1 )</td>
</tr>
</tbody>
</table>

C. Sturton
Symbolic Execution

```c
if (reset)
    count = 0;
else
    count = count+1;
if (count > 3)
    ERROR;
```

Path condition | Symbolic state
--- | ---
True | reset := r₀
| count := c₀
r₀ = 1 | reset := r₀
| count := 0
r₀ = 1
| 0 > 3

C. Sturton
Symbolic Execution of a Hardware Design
Symbolic Execution of a Hardware Design

Realizable processor states
Symbolic Execution of a Hardware Design

one clock cycle
Symbolic Execution of a Hardware Design

one clock cycle
Backward Search

\[ \text{instruction } i_n \]
Backward Search

\[
\text{instruction } i_{n-1}
\]
Backward Search

\[ \text{instruction } i_{n-2} \]
Backward Search

trigger:

\[ i_0, i_1, \ldots, i_{n-2}, i_{n-1}, i_n \]

instruction \( i_{n-2} \)
If a sequence of inputs is returned, it will take the processor from the initial state to an error state.
Requirements

processor’s initial state $\models$ path condition
Requirements

...

symbolic state ⊆ assertion failure

C. Sturton
Requirements
Requirements

symbolic state of leaf $j$ ⊆ path condition of leaf $j + 1$
Making it Work

1. Internal and input signals are symbolic
Making it Work

2. Is this an initial state?
Making it Work

3. How much does this differ from an initial state?
3. Are we in a loop?
Making it Work

4. Have we exceeded the bound?
```vhdl
// Output Logic
// Depending on the state
// output signal has a different
// value.
always @ (posedge clk)
begin
  if (present_state == sig1)
    signal = '0'b1000;
  else if (present_state == sig2)
    signal = '0'b1100;
  else if (present_state == sig3)
    signal = '0'b1010;
  else if (present_state == sig4)
    signal = '0'b1011;
  else
    signal = '0'b1101;
end
```
Coppelia

- Transcompiler [Verilator]
- Symbolic Execution w/Recursive Strategy
- Triggering Instructions
- Exploit C Program
- Payload
- Program Stub Generation [Manual]

C CPU Design

Security Property

Exploit C C Code

Property \( \varphi \)
Coppelia

Exploit C Program

Payload

Program Stub Generation [Manual]

Triggering Instructions

Symbolic Execution w/Recursive Strategy

C Code

Transcompiler [Verilator]

CPU Design

Security Property

\( \varphi \)

C. Sturton
Coppelia

- Transcompiler [Verilator]
- Symbolic Execution w/Recursive Strategy
- Exploit C Program
- Payload
- Program Stub Generation [Manual]
- Triggering Instructions
- Security Property
- CPU Design
- Security Property

C. Sturton
Optimizations

- Explore only legal instructions
- Alternate depth-first and breadth-first searching
- Cone of Influence analysis for slicing
Evaluating Optimizations

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>DFS + BFS</th>
<th>Cone of Influence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average CPU time to find bug</td>
<td>&gt; 19h</td>
<td>&gt; 1.2h</td>
<td>4m 12s</td>
</tr>
</tbody>
</table>

- Considered only bugs triggerable with a single instruction
Evaluating Coppelia

- Does Coppelia find bugs and generate their exploits?
- Will our approach find new bugs?
Processor

- OR1200
- 31 known bugs
Processor
- OR1200
- 31 known bugs

Coppelia

35 properties
- SPECS
- SecurityCheckers
- SCIFinder
Processor
- OR1200
- 31 known bugs

35 properties
- SPECS
- SecurityCheckers
- SCIFinder

Coppelia
29 exploits
Finding Bugs (ground truth: 31)

- Coppelia: 29 bugs
  - Not replayable: 18
  - Replayable: 11

- Cadence IFV: 18 bugs
  - Not replayable: 10
  - Replayable: 10

- EBMC: 16 bugs
  - Not replayable: 9
  - Replayable: 7

C. Sturton
Processor
- Mor1kx
- PULPino

Coppelia
Processor
- Mor1kx
- PULPino

Properties
- SPECS
- SecurityCheckers
- SCIFinder

Coppelia
Processor
- Mor1kx
- PULPino

Properties
- SPECS
- SecurityCheckers
- SCIFinder

Coppelia

4 new bugs
Finding New Bugs

1. Mor1kx-Espresso
   new design

2. PULPino-RI5CY
   new architecture!
Security validation of hardware designs can be done algorithmically.
Our Products So Far

- **SCIFinder** to produce security critical properties
- **Coppelia** to find and generate exploits for property violations
- **Security properties** for RISC processor designs
Thank you

Rui Zhang, Calvin Deutschbein, Natalie Stanley, Chris Griggs, Andrew Chi, Ryan Huang, Alyssa Byrnes, Matthew Hicks, Jonathan M. Smith, Sam T. King.