Securing Hardware Platforms Against Malicious Circuits Through Static Analysis

Matthew Hicks - University of Illinois
Samuel T. King - University of Illinois
Milo M. K. Martin - University of Pennsylvania
Jonathan M. Smith - University of Pennsylvania
Building Secure Systems

- We make assumptions when designing secure systems
- Break secure system, break assumptions
  - E.g., look for crypto keys in memory
- People assume hardware is correct

- What can we do if we can’t make this assumption?
Why Hardware?

- Key hardware properties
  - Complex
  - Expensive
  - Static
  - Base of the system
The Threat
Dead Circuit Identification (DCI)

- Goal: Help designers by highlighting all potentially malicious circuits automatically
Dead Circuit Identification (DCI)

- Goal: Help designers by highlighting all potentially malicious circuits automatically
- Intuition: Attacker is motivated to avoid impacting functionality during testing
  - Otherwise they would be caught
Dead Circuit Identification (DCI)

- Goal: Help designers by highlighting all potentially malicious circuits automatically
- Intuition: Attacker is motivated to avoid impacting functionality during testing
  - Otherwise they would be caught

- Detect all circuits where the output value is identical to the input value for all tests
  - Internal circuits don’t impact functionality
Simple Example
Simple Example
Simple Example

Uncontaminated Circuit

Malicious Circuit

MUX

priv

out

escalate_priv

1
What the numbers say... so far

<table>
<thead>
<tr>
<th>Foothold</th>
<th>Num. nodes in data-flow graph</th>
<th>False negatives</th>
<th>False positives</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non-foothold</td>
<td>Foothold</td>
<td></td>
</tr>
<tr>
<td>Supervisor transition</td>
<td>532</td>
<td>3</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Shadow mode</td>
<td>532</td>
<td>4</td>
<td>0 (0%)</td>
</tr>
</tbody>
</table>
Questions
It Can Happen

IF ( r.d.inst ( conv_integer ( r.d.set ) ) = X"80082000" ) THEN
  hackStateM1 <= '1';
ELSE
  hackStateM1 <= '0';
END IF;

IF ( r.d.inst ( conv_integer ( r.d.set ) ) = X"80102000" ) THEN
  r.w.s.s <= hackStateM1 OR rin.w.s.s;
ELSE
  r.w.s.s <= rin.w.s.s;
END IF;
Won’t there be a lot of pairs

- Number of pairs is $N^2$ in the levels of logic
Will I have to re-run my tests

- Dataflow pairs analysis is additive
- New tests can be added and tested independently of old tests
What’s the relationship between pairs remaining and code coverage

- Each uncovered branch is a missed opportunity for a unique assignment to a wire
Complex Example

\[
k \leftarrow \text{good} \quad \text{when} \quad (C_1 = '0') \quad \text{else} \quad \text{bad};
\]

\[
l \leftarrow \text{good} \quad \text{when} \quad (C_2 = '0') \quad \text{else} \quad \text{bad};
\]

\[
\text{out} \leftarrow m \quad \text{when} \quad (C_1 = '0') \quad \text{else} \quad n;
\]

\[
\text{process}(\text{clk}) \begin{align*}
&\text{begin} \\
&\quad \text{if} (\text{rising}_\text{edge}(\text{clk})) \text{then} \\
&\quad\quad m \leftarrow k; \\
&\quad\quad n \leftarrow l; \\
&\quad \text{end if}; \\
&\text{end process;}
\end{align*}
\]
Complex Example

Test cases:
- C1
- C2

Pairs:
- (good, k, 0)
- (good, m, 1)
- (good, out, 1)
- (bad, k, 0)
- (bad, m, 1)
- (bad, out, 1)
- (k, m, 1)
- (k, out, 1)
- (m, out, 0)
- (good, l, 0)
- (good, n, 1)
- (bad, l, 0)
- (bad, n, 1)
- (l, n, 1)
- (l, out, 1)
- (n, out, 0)
Complex Example

Pairs:
- (good, k, 0)
- (good, m, 1)
- (good, out, 1)
- (bad, k, 0)
- (bad, m, 1)
- (bad, out, 1)
- (k, m, 1)
- (k, out, 1)
- (m, out, 0)
- (good, l, 0)
- (good, n, 1)
- (bad, l, 0)
- (bad, n, 1)
- (l, n, 1)
- (l, out, 1)
- (n, out, 0)

Test cases:
<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Complex Example

Test cases:

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Pairs:

- (good, k, 0)
- (good, m, 1)
- (good, out, 1)
- (bad, k, 0)
- (bad, m, 1)
- (bad, out, 1)
- (k, m, 1)
- (k, out, 1)
- (m, out, 0)
- (good, l, 0)
- (good, n, 1)
- (bad, l, 0)
- (bad, n, 1)
- (l, n, 1)
- (l, out, 1)
- (n, out, 0)

k = good
l = bad
m = good
n = good
out = good
Complex Example

Pairs:
(good, k, 0)
(good, m, 1)
(good, out, 1)
(bad, k, 0)
(bad, m, 1)
(bad, out, 1)
(k, m, 1)
(k, out, 1)
(m, out, 0)
(good, l, 0)
(good, n, 1)
(bad, l, 0)
(bad, n, 1)
(l, n, 1)
(l, out, 1)
(n, out, 0)

Test cases:
<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>

k = bad
l = good
m = good
n = bad
out = good
Complex Example

Pairs:
- (good, k, 0)
- (good, m, 1)
- (good, out, 1)
- (bad, k, 0)
- (bad, m, 1)
- (bad, out, 1)
- (k, m, 1)
- (k, out, 1)
- (m, out, 0)
- (good, l, 0)
- (good, n, 1)
- (bad, l, 0)
- (bad, n, 1)
- (l, n, 1)
- (l, out, 1)
- (n, out, 0)

Test cases:
- C1  C2
- 0   0
- 0   1
- 1   0
- 0   0
- ... ...
- 1   0

k = good
l = good
m = bad
n = good
out = good
Complex Example

Test cases:
- C1: 0 0
- C2: 0 1
- C1: 1 0
- C2: 0 0

Pairs:
- (good, k, 0)
- (good, m, 1)
- (good, out, 1)
- (bad, k, 0)
- (bad, m, 1)
- (bad, out, 1)
- (k, m, 1)
- (k, out, 1)
- (m, out, 0)
- (good, l, 0)
- (good, n, 1)
- (bad, l, 0)
- (bad, n, 1)
- (l, n, 1)
- (l, out, 1)
- (n, out, 0)
Problems

- Undefined state
  - Low visibility test cases

- Control information
  - Implementation dependent
Hardware Attack Properties

- **Small**
  - Avoids visual, and side-channel analysis

- **Powerful**
  - Can contaminate the entire system

- **Difficult to stop**
  - Ingrained deep within the system, potentially impacting common functionality
What Happens?