Overcoming an **UNTRUSTED COMPUTING BASE:**
*Detecting and Removing Malicious Hardware Automatically*

Matthew Hicks
Murph Finnicum
Samuel T. King
University of Illinois Urbana-Champaign

Milo M. K. Martin
Jonathan M. Smith
University of Pennsylvania

Hardware is too important to trust blindly

Hardware is as complex as software

Hardware complexity equals hardware vulnerability

Hardware must be defended against malicious designers

BlueChip looks for malicious insertions at design time and prevents them from affecting the system during runtime
BlueChip is both hardware and software, design time and run time

<table>
<thead>
<tr>
<th>Software system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handle missing hardware - BlueChip software</td>
</tr>
<tr>
<td>Handle malicious circuits - BlueChip hardware</td>
</tr>
<tr>
<td>Identify malicious circuits - UCI</td>
</tr>
<tr>
<td>Original Design</td>
</tr>
</tbody>
</table>

Design

Implementation
Experiments
Conclusion
Questions

BlueChip helps managers increase trust, without requiring them to know more

UCI highlights potentially malicious circuits automatically

Attackers must avoid impacting functionality during testing

UCI detects all circuits where the output value is identical to the input value, for all test cases
Data-flow triples generation

start at output signals

recurse_tuples

for each item in the parents list

generate a tuple

for each driver

add self to temp parents list

if driver behind a flip-flop

increase delay in temp parents list

recurse on child

Triples:
(good, m, 0)

(good, out, 0)

(bad, m, 0)

(good, n, 0)

(bad, n, 0)

(n, out, 0)
UCI Analysis

for each test case
for each clock cycle
for each dataflow triple remaining
if target != driver(delay)
remove triple
BlueChip is both hardware and software, design time and run time.

**Design Time**
- Circuit designed
- Attack inserted
- Suspicious circuits identified and removed

**Run Time**
- Hardware triggers emulation software

BlueChip hardware alerts software when it attempts to use removed circuits.
BlueChip software emulates the behavior of removed hardware
1. Receive BlueChip exception
2. Load state of processor
3. Fetch trapping instruction
4. Decode trapping instruction
5. Execute trapping instruction in emulator
6. Store updated state to hardware
7. Return from trap

BlueChip does **NOT** emulate the removed hardware

BlueChip **DOES** emulate the behavior of the hardware spec at a higher level of abstraction

- **Undefined state**
  - Low visibility test cases
  - Architecturally undefined state
- **Malicious test cases**
  - ISA emulator also vettes test cases
- **Control information**
  - Implementation dependent

BlueChip isn’t effective in certain situations
**Experiments**

**BlueChip successfully prevents malicious hardware**

<table>
<thead>
<tr>
<th>Attack</th>
<th>Prevent</th>
<th>Recover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privilege Escalation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory Redistribution</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Shadow Mode</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**BlueChip handles UCI false positives**

```
<table>
<thead>
<tr>
<th>Attack</th>
<th>Normalized Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>0.12</td>
</tr>
<tr>
<td>make</td>
<td>1.0</td>
</tr>
<tr>
<td>jpeg</td>
<td>0.8</td>
</tr>
<tr>
<td>Privilege</td>
<td>1.0</td>
</tr>
<tr>
<td>Memory</td>
<td>1.0</td>
</tr>
</tbody>
</table>
```

**BlueChip has a low overhead**

```
<table>
<thead>
<tr>
<th>Attack</th>
<th>Normalized Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>wget</td>
<td>0.12</td>
</tr>
<tr>
<td>make</td>
<td>1.0</td>
</tr>
<tr>
<td>jpeg</td>
<td>0.8</td>
</tr>
<tr>
<td>Privilege</td>
<td>1.0</td>
</tr>
<tr>
<td>Memory</td>
<td>1.0</td>
</tr>
</tbody>
</table>
```
BlueChip allows flexible handling of untrusted hardware

UCI isn’t as complex as it seems

Code coverage is deficient in both time and space
Hardware attacks can be trivial to implement, but hard to detect

```assembly
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;
```

Hardware attacks can be trivial to implement, but hard to detect.

Sometimes BlueChip software must emulate around instructions

```assembly
// Load regs[r3] and regs[r4] in l3 and l4
LD [I4-2], I5
AND I3, 0xffff, I3
SRL I5, 16, I5
SLL I5, 16, I5
OR I5, I3, I3
ST I3, [I4-2]
```

Sometimes BlueChip software must emulate around instructions

```assembly
SUB g0, 1, I5
XOR I3, I5, I3
OR r3, r4, r3
XOR I4, I5, I4
NAND I3, I4, I3
// Store I3 into regs[r3]
```

Sometimes BlueChip software must emulate around instructions

```assembly
OR r3, r4, r3
XOR I4, I5, I4
NAND I3, I4, I3
// Store I3 into regs[r3]
```

Sometimes BlueChip software must emulate around instructions

```assembly
// Load regs[r3] and regs[r4] in l3 and l4
LD [I4-2], I5
AND I3, 0xffff, I3
SRL I5, 16, I5
SLL I5, 16, I5
OR l5, l3, l3
ST l3, [l4-2]
```

Sometimes BlueChip software fails to make forward progress

```assembly
// Load regs[r3] and regs[r4] in l3 and l4
LD [I4-2], I5
AND I3, 0xffff, I3
SRL I5, 16, I5
SLL I5, 16, I5
OR l5, l3, l3
ST l3, [l4-2]
```

Sometimes BlueChip software fails to make forward progress

```assembly
// Load regs[r3] and regs[r4] in l3 and l4
LD [I4-2], I5
AND I3, 0xffff, I3
SRL I5, 16, I5
SLL I5, 16, I5
OR l5, l3, l3
ST l3, [l4-2]
```

Sometimes BlueChip software fails to make forward progress

```assembly
// Load regs[r3] and regs[r4] in l3 and l4
LD [I4-2], I5
AND I3, 0xffff, I3
SRL I5, 16, I5
SLL I5, 16, I5
OR l5, l3, l3
ST l3, [l4-2]
```

What happens when the attack triggers on

0x40005555 <= address >> 0x4000aaaa

When r4 = 0x4005ccee

What happens when the attack triggers on

0x40005555 <= address >> 0x4000aaaa

When r4 = 0x4005ccee

Assumes r4 is not word aligned

Sometimes BlueChip software fails to make forward progress

```assembly
STH r3, [r4]
```

Sometimes BlueChip software fails to make forward progress

```assembly
STH r3, [r4]
```

Sometimes BlueChip software fails to make forward progress

```assembly
STH r3, [r4]
```

Sometimes BlueChip software fails to make forward progress

```assembly
STH r3, [r4]
```

Sometimes BlueChip software failures to make forward progress

```assembly
STH r3, [r4]
```

Sometimes BlueChip software failures to make forward progress

```assembly
STH r3, [r4]
```

Sometimes BlueChip software failures to make forward progress

```assembly
STH r3, [r4]
```

Sometimes BlueChip software failures to make forward progress

```assembly
STH r3, [r4]
```