Symbolic Execution of Virtual Devices

Kai Cong
Portland State University
Virtual Devices to the Rescue

Virtual Devices can enable early driver development.
Is it possible to bring more benefits with virtual devices to help HW/SW development and validation???
static uint32_t
e1000_mmio_readl(void *opaque, uint64 addr)
{
    E1000State *s = opaque;
    unsigned int index = (addr & 0x1ffff) >> 2;
    if (macreg_readops[index])
    {
        return macreg_readops[index](s, index);
    }
    return 0;
}

......
Symbolic Execution of Virtual Devices

- Execute virtual devices symbolically
- Enumerate as many paths as possible
- Generate and replay concrete test cases

Symbolic input

A concrete test case
Evaluation

- Applied to five QEMU virtual devices
- Most popular network adapters

<table>
<thead>
<tr>
<th>Virtual Device</th>
<th>Vendor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1000</td>
<td>Intel</td>
<td>Pro/1000 Gigabit Ethernet Adapter</td>
</tr>
<tr>
<td>EEPro100</td>
<td>Intel</td>
<td>Pro/100 Ethernet Adapter</td>
</tr>
<tr>
<td>PCNet</td>
<td>AMD</td>
<td>PCNet32 10/100 Ethernet Adapter</td>
</tr>
<tr>
<td>RTL8139</td>
<td>Realtek</td>
<td>PCI Fast Ethernet Adapter</td>
</tr>
<tr>
<td>Tg3</td>
<td>Broadcom</td>
<td>BCM57xx-based Gigabit Ethernet Adapter</td>
</tr>
</tbody>
</table>
**Evaluation**

- Experiment setup: 8-core i7 CPU, 8 GB of RAM, and Ubuntu Linux 64-bit
- Five configurations with different loop bounds and time bounds

<table>
<thead>
<tr>
<th>Device</th>
<th>Paths</th>
<th>Memory (MB)</th>
<th>Paths</th>
<th>Memory (MB)</th>
<th>Paths</th>
<th>Memory (MB)</th>
<th>Paths</th>
<th>Memory (MB)</th>
<th>Paths</th>
<th>Memory (MB)</th>
<th>Paths</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1000</td>
<td>289</td>
<td>401</td>
<td>440</td>
<td>721</td>
<td>671</td>
<td>1614</td>
<td>406</td>
<td>803</td>
<td>505</td>
<td>1622</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EEPro100</td>
<td>183</td>
<td>131</td>
<td>499</td>
<td>254</td>
<td>1539</td>
<td>508</td>
<td>468</td>
<td>238</td>
<td>1005</td>
<td>483</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTL8139</td>
<td>371</td>
<td>66</td>
<td>402</td>
<td>131</td>
<td>408</td>
<td>238</td>
<td>404</td>
<td>131</td>
<td>414</td>
<td>262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCNet</td>
<td>279</td>
<td>74</td>
<td>424</td>
<td>139</td>
<td>646</td>
<td>262</td>
<td>417</td>
<td>139</td>
<td>601</td>
<td>262</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tg3</td>
<td>252</td>
<td>811</td>
<td>391</td>
<td>1196</td>
<td>556</td>
<td>4104</td>
<td>398</td>
<td>1581</td>
<td>569</td>
<td>3162</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Concrete execution

Symbolic execution

Conclusions and Future Work

• Presented an approach to symbolic execution of virtual devices, central to achieving observability and traceability.

• Application example
  – Concolic Test Generation for Post-silicon Validation

• Future work
  – Algorithms for setting loop bounds adaptively
  – Utilization of symbolic execution of virtual devices in run-time fault injection and test coverage computation
Thanks!