Reverse Engineering of Binary Device Drivers with RevNIC
by Vitaly Chipounov and George Candea

Steffen Liebergeld

TU-Berlin

11. 05. 2010
Agenda

1. Symbolic Execution
   - Introduction
   - Selective Symbolic Execution

2. RevNIC
   - Motivation
   - Implementation
   - Evaluation
   - Summary

3. Discussion
Symbolic Execution

- Provide input with symbolic instead of concrete value
  - Example: \( x = 7 \Rightarrow x = \lambda \)
- Update expressions
  - Example: \( x += 2 \Rightarrow x = \lambda + 2 \)
- Copy state on conditionals
- Split execution into different paths
- Explore both

\[
I = \text{<input>};
x = I - 2;
\text{if } (x > 0) \{ \\
  \text{  if } (x < 8) \{ \\
    \ldots \\
  \}
\}
\text{else } \{ \\
  \text{  if } (x > -12) \{ \\
    \ldots \\
  \}
\}
\]

- Constraint solver to find concrete input
- Use cases: automated software testing, malware analysis and more
Selective Symbolic Execution

- Problem: Exponential growth of execution tree (*state explosion*)
- Symbolic execution feasible only for small programs with sharp system boundaries
- But systems are complex:

![Diagram of system components]

- Solution: Restrict symbolic execution
  - *Selective Symbolic Execution*
- Enables new use cases: binary driver analysis, systems debugging ...
The Driver Problem

- Lots of devices available, but
  - Binary only drivers for the most popular platforms only
- Driver availability crucial for OS adaption
- How to get functional drivers?
  - Implement according to specification
    - Specification often unavailable: Device logic is trade secret
    - HW quirks often not included
  - Reverse engineering
    - Time consuming
    - New devices appear all the time...
  - Reuse existing drivers with compatibility layer
    - Possibly bad performance
    - Hard to get right
RevNIC - Automated Reverse Engineering of Binary Device Drivers

- Idea: Reverse engineer binary device drivers automatically
- Agenda:
  1. Exercise the driver
  2. Record driver activity
  3. Synthesize driver code
  4. Provide driver template
  5. Synthesize driver
Exercise, Record and Synthesize Driver Code

- Exercise driver
  - Initiate driver execution with concrete workload
  - Driver code run in symbolic execution
  - kernel-to-driver and HW response converted to symbolic values
  - No access to physical device needed

- Recording driver activity
  - HW IO
  - Memory accesses
  - Intermediate representation of instructions

- Synthesize driver code
  - Infer driver state machine from collected traces
  - Produce code implementing state machine
Driver Template and Driver Synthesis

- Driver template
  - Provided by OS developer
  - Boilerplate code for interaction with OS
- Synthesizing the driver
  - Paste generated code into template
  - Compile + load

![Diagram of driver template and driver synthesis process]

- Guest OS (e.g., Windows)
- RevNIC Wiretap
- Original Driver
- Activity traces
- RevNIC Code Synthesizer
- Synthetic Driver (e.g., for Linux)
- NIC Driver Template
- C code
## Evaluation

<table>
<thead>
<tr>
<th>Reverse Engineered Windows Driver</th>
<th>RevNIC Ported from Windows to ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD PCNet</td>
<td>Windows, Linux, KitOS</td>
</tr>
<tr>
<td>Realtek RTL8139</td>
<td>Windows, Linux, KitOS</td>
</tr>
<tr>
<td>SMSC 91C111</td>
<td>μC/OS-II, KitOS</td>
</tr>
<tr>
<td>Realtek RTL8029 (NE2000)</td>
<td>Windows, Linux, KitOS</td>
</tr>
</tbody>
</table>
Evaluation

<table>
<thead>
<tr>
<th>Reverse Engineered Windows Driver</th>
<th>RevNIC Ported from Windows to ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD PCNet</td>
<td>Windows, Linux, KitOS</td>
</tr>
<tr>
<td>Realtek RTL8139</td>
<td>Windows, Linux, KitOS</td>
</tr>
<tr>
<td>SMSC 91C111</td>
<td>μC/OS-II, KitOS</td>
</tr>
<tr>
<td>Realtek RTL8029 (NE2000)</td>
<td>Windows, Linux, KitOS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functionality</th>
<th>AMD PCNet</th>
<th>RTL8139</th>
<th>SMSC 91C111</th>
<th>RTL8029</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init/Shutdown</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Send/Receive</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Multicast</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Get/Set MAC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Promiscuous Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Full Duplex</td>
<td>✓</td>
<td>✓</td>
<td>✓ N/A</td>
<td>✓ N/A</td>
</tr>
<tr>
<td>DMA</td>
<td>✓</td>
<td>✓</td>
<td>✓ N/A</td>
<td>✓ N/A</td>
</tr>
<tr>
<td>Wake-on-LAN</td>
<td>N/T</td>
<td>✓</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>LED Status Display</td>
<td>N/T</td>
<td>✓</td>
<td>✓ N/T</td>
<td></td>
</tr>
</tbody>
</table>
Performance - RTL 8139 Driver

![Graph showing throughput vs UDP packet size for different scenarios]
Performance - RTL 8139 Driver

- Throughput (Mbps) vs UDP Packet Size (Bytes)
- CPU Utilization (%) vs UDP Packet Size (Bytes)
## Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>Manual (Linux)</th>
<th>RevNIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Persons</td>
<td>Span</td>
</tr>
<tr>
<td>RTL8139</td>
<td>18</td>
<td>4 years</td>
</tr>
<tr>
<td>SMSC 91C111</td>
<td>8</td>
<td>4 years</td>
</tr>
<tr>
<td>RTL8029</td>
<td>5</td>
<td>2 years</td>
</tr>
<tr>
<td>AMD PCNet</td>
<td>3</td>
<td>4 years</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Device</th>
<th>Manual (Linux)</th>
<th>RevNIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Persons</td>
<td>Span</td>
</tr>
<tr>
<td>RTL8139</td>
<td>18</td>
<td>4 years</td>
</tr>
<tr>
<td>SMSC 91C111</td>
<td>8</td>
<td>4 years</td>
</tr>
<tr>
<td>RTL8029</td>
<td>5</td>
<td>2 years</td>
</tr>
<tr>
<td>AMD PCNet</td>
<td>3</td>
<td>4 years</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target OS</th>
<th>Person-Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>5</td>
</tr>
<tr>
<td>Linux</td>
<td>3</td>
</tr>
<tr>
<td>μC/OS-II</td>
<td>1</td>
</tr>
<tr>
<td>KitOS</td>
<td>0</td>
</tr>
</tbody>
</table>
Summary

- Automated reverse engineering of binary device drivers
- Drivers for four platforms generated
- Performance is on par with original (but more CPU utilization)
Discussion

- Authors claim: Applicable to large range of hardware
- Legal status?
- Code coverage?
- What about bugs in the original driver?