Toward a Trustable, Self-Hosting Computer System
Gabriel L. Somlo, Ph.D.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM20-0353
Computer Systems are Everywhere

Embedded Computers with *exotic* enclosures and peripherals, e.g.:

- comms
- navigation
- artillery
A Tech Based Supply Chain Workaround?

- Trustworthiness of computer hardware is a **Big Question**
  - Microchips in particular!
- Development / Design / Production supply chains are problematic
  - Multinational corporations
  - Opaque relationships, abundance of NDAs
  - International, highly mobile workforce

- Non-destructive testing & reverse engineering of microchips is **HARD**
  - Unlike *software*
Hardware Attack Surface

• ASIC Fabrication (Malicious Foundry)
  - masks reverse engineered and modified to insert malicious behavior
    • privilege escalation CPU backdoor
    • compromised random number generator
  - problematic to test/verify after the fact!
  - mitigated by using FPGAs instead!

• Compilation (Malicious Toolchain)
  - generates malicious design from clean sources

• Design Defects (Accidentally or Intentionally Buggy HDL Sources)
  - Spectre
  - Meltdown
### Field Stripping a Computer

<table>
<thead>
<tr>
<th>Applications (incl. compiler)</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Runtime Libraries</td>
<td>(e.g., Linux, BSD, seL4)</td>
</tr>
<tr>
<td>Kernel</td>
<td>Hardware</td>
</tr>
<tr>
<td>Hypervisor (optional)</td>
<td>(e.g., x86, ARM, RISC-V)</td>
</tr>
<tr>
<td>CPU ISA &amp; I/O Registers</td>
<td></td>
</tr>
<tr>
<td>Microarchitecture</td>
<td></td>
</tr>
<tr>
<td>Register Transfer Level (RTL)</td>
<td></td>
</tr>
<tr>
<td>ASICs</td>
<td>FPGAs</td>
</tr>
</tbody>
</table>

#### Self-hosting:

- a system’s capability to produce new versions of itself, from bounded sources, without reliance on external third-party support*
- the software stack is self-hosting

* Assuming the hardware can be trusted!!!
Field Stripping a Computer

<table>
<thead>
<tr>
<th>Applications (incl. compiler)</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(e.g., Linux, BSD, seL4)</td>
</tr>
<tr>
<td>System Runtime Libraries</td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td></td>
</tr>
<tr>
<td>Hypervisor (optional)</td>
<td></td>
</tr>
<tr>
<td>CPU ISA &amp; I/O Registers</td>
<td>Hardware</td>
</tr>
<tr>
<td>Microarchitecture</td>
<td>(e.g., x86, ARM, RISC-V)</td>
</tr>
<tr>
<td>Register Transfer Level (RTL)</td>
<td></td>
</tr>
<tr>
<td>ASICs</td>
<td></td>
</tr>
<tr>
<td>FPGAs</td>
<td></td>
</tr>
</tbody>
</table>

Self-hosting:

- a system’s capability to produce new versions of itself, from bounded sources, without reliance on external third-party support*
- the software stack is self-hosting
  * Assuming the hardware can be trusted!!!

Goal: Extend self-hosting property to encompass hardware, including hardware source-language (HDL) compiler!
Hardware Development and Compilation Stages

Source Code

```module alu_mod (  // operator:  input alu_op_t op,  // operands:  input logic [31:0] a, b,  // result:  output logic [31:0] res);  
always_comb begin  // unique case (op)  
unique case (op)  
   ALU_ADD: res = a + b;  
   ALU_MUL: res = a * b;  
   ALU_XOR: res = a ^ b;  
   ALU_AND: res = a & b;  
   ALU_OR: res = a | b;  
   default: res = 32'b0;  
endcase  
end  
endmodule: alu_mod```

Elaboration

![Diagram of logic gates]  
```
module alu_mod (  // operator:  input alu_op_t op,  // operands:  input logic [31:0] a, b,  // result:  output logic [31:0] res);  
always_comb begin  // unique case (op)  
unique case (op)  
   ALU_ADD: res = a + b;  
   ALU_MUL: res = a * b;  
   ALU_XOR: res = a ^ b;  
   ALU_AND: res = a & b;  
   ALU_OR: res = a | b;  
   default: res = 32'b0;  
endcase  
end  
endmodule: alu_mod```

Synthesis, Optimization

![Diagram of circuit board]  
```
module alu_mod (  // operator:  input alu_op_t op,  // operands:  input logic [31:0] a, b,  // result:  output logic [31:0] res);  
always_comb begin  // unique case (op)  
unique case (op)  
   ALU_ADD: res = a + b;  
   ALU_MUL: res = a * b;  
   ALU_XOR: res = a ^ b;  
   ALU_AND: res = a & b;  
   ALU_OR: res = a | b;  
   default: res = 32'b0;  
endcase  
end  
endmodule: alu_mod```

Technology Mapping, Place-and-Route

![Diagram of circuit board]  
```
module alu_mod (  // operator:  input alu_op_t op,  // operands:  input logic [31:0] a, b,  // result:  output logic [31:0] res);  
always_comb begin  // unique case (op)  
unique case (op)  
   ALU_ADD: res = a + b;  
   ALU_MUL: res = a * b;  
   ALU_XOR: res = a ^ b;  
   ALU_AND: res = a & b;  
   ALU_OR: res = a | b;  
   default: res = 32'b0;  
endcase  
end  
endmodule: alu_mod```

mask (ASIC)  or  bitstream (FPGA)
ASICS vs. FPGAs

- Application Specific Integrated Circuits
  - dedicated, optimized etched silicon
    - photolithographic masks
  - “hard” IP cores

- Field Programmable Gate Arrays
  - grid: programmable blocks, interconnect
    - bitstream
  - “soft” IP cores
Anchoring Trust for Fielded Systems

System Sources (blueprint, design, etc.)

Build Tool

Fielded System
Anchoring Trust for Fielded Systems

System Sources (blueprint, design, etc.)

Builder Sources

Build Tool

Fielded System
Anchorng Trust for Fielded Systems

System Sources (blueprint, design, etc.)

Builder Sources

C Compiler

Build Tool

Fielded System
Anchoring Trust for Fielded Systems

System Sources (blueprint, design, etc.)

Builder Sources

C Compiler

Build Tool

Fielded System

Builder Sources

C Compiler

Build Tool

Fielded System
Bootstrapping a Trustworthy RISC-V Cleanroom System

**Host** (x86/Linux):

- Use DDC to verify we have a clean C (cross-)compiler
- Build clean HDL compiler toolchain, for both x86 and rv64
- Cross-compile target rv64 OS (kernel, libraries, utilities)
- Build rv64 SoC FPGA bitstream, from HDL sources

**Target** (rv64/Linux):

- Boot up FPGA-based rv64 computer into cross-compiled OS
  - rv64/Linux system is *self-hosting* from this point forward!
- Natively rebuild FPGA bitstream, kernel, libraries, and applications
  - we now have a trustworthy cleanroom
  - guaranteed to “honestly” compile any imported sources (HDL and/or software)!
List of Ingredients

Physical Hardware: FPGA development board (based on Lattice ECP5 series chip):

- Versa-5G or TrellisBoard

Free/Open HDL toolchain (Verilog-to-bitstream):

- Yosys (compiler), Project Trellis (bitstream utilities), NextPNR (place-and-route tool)

Free/Open RISC-V 64-bit CPU:

- Rocket Chip

Free/Open system-on-chip (SoC) environment (e.g., system bus, peripherals):

- LiteX

Free/Open software stack (e.g., Linux kernel, glibc runtime, GCC compiler):

- Fedora-riscv64
LiteX + Rocket 64-bit FPGA-based Linux Computer

- Memory (DRAM) Controller
- CPU (Rocket Chip)
- Peripheral Bus (MMIO)
- UART
- Ethernet (Eth.)
- µSD

LiteX
## Benchmarks

<table>
<thead>
<tr>
<th>CPU</th>
<th>MHz</th>
<th>CoreMark</th>
<th>Linpack (KFLOPS)</th>
<th>P5-90</th>
<th>K6-223</th>
<th>nbench</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Single</td>
<td>Double</td>
<td>Int</td>
<td>Float</td>
<td>Mem</td>
</tr>
<tr>
<td>P5</td>
<td>90</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.00</td>
<td>1.00</td>
<td>-</td>
</tr>
<tr>
<td>K6</td>
<td>233</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.00</td>
</tr>
<tr>
<td>Xeon</td>
<td>2400</td>
<td>12489.07</td>
<td>1679090</td>
<td>1618198</td>
<td>109.59</td>
<td>112.60</td>
<td>34.36</td>
</tr>
<tr>
<td>rv64gc</td>
<td></td>
<td>1468.86</td>
<td>21520</td>
<td>20964</td>
<td>13.38</td>
<td>1.67</td>
<td>2.80</td>
</tr>
<tr>
<td>u54mc</td>
<td>1400</td>
<td>2079.59</td>
<td>112832</td>
<td>88496</td>
<td>18.21</td>
<td>12.97</td>
<td>3.81</td>
</tr>
<tr>
<td>P5</td>
<td>133</td>
<td>282.63</td>
<td>13227</td>
<td>8923</td>
<td>1.77</td>
<td>0.90</td>
<td>0.35</td>
</tr>
<tr>
<td>Rocket</td>
<td>65</td>
<td>47.45</td>
<td>48</td>
<td>31</td>
<td>0.31</td>
<td>0.003</td>
<td>0.077</td>
</tr>
<tr>
<td>Rocket</td>
<td>60</td>
<td>103.89</td>
<td>84</td>
<td>79</td>
<td>0.47</td>
<td>0.003</td>
<td>0.11</td>
</tr>
<tr>
<td>Rocket</td>
<td>50</td>
<td>103.58</td>
<td>5709</td>
<td>4492</td>
<td>0.92</td>
<td>0.67</td>
<td>0.19</td>
</tr>
</tbody>
</table>
## Next Steps

<table>
<thead>
<tr>
<th>NEAR</th>
<th>MID</th>
<th>FAR</th>
</tr>
</thead>
</table>
| **Performance Optimizations**<br>• Early prototype HDL is a target-rich environment for further performance improvements, e.g.,:  
• 64bit AXI system bus  
• separate RAM and MMIO data paths | **Formal Analysis & Verification**<br>• Starting from a bounded set of sources, 100% as trustworthy as the fielded system.  
• Goal: measure actual ability to trust the system by conducting source code analysis! | **Hardware Assurance BCPs**<br>• Supply chain complexity mitigated by hardware openness, auditability |
In Conclusion…

• **Side-stepping** supply chain questions re. hardware assurance

• **FPGAs** mitigate against malicious foundry (silicon) backdoors

• **Field Stripping** computers (from complete sources) to determine trustability of:
  
  - build tools
  
  - fielded end-product systems
Demo: Linux on Rocket+LiteX (on ECP5 FPGA)
http://www.contrib.andrew.cmu.edu/~somlo/BTCP
C Compiler vs. “Trusting Trust”: Problem & Workaround

• **self-propagating C compiler hack** (Ken Thompson)
  - malicious compiler inserts Trojan during compilation of a *victim program*
    • clean source → malicious binary
      • including *compiler’s own sources!*
      • compiler source hack *no longer needed* after 1st iteration!

• David A. Wheeler’s defense: **Diverse Double Compilation**
  - suspect compiler A: sources $S_A$, binary $B_A$
  - trusted compiler T: binary $B_T$

  \[
  S_A \rightarrow B_A \rightarrow X \quad \quad \quad \quad \quad S_A \rightarrow B_T \rightarrow Y
  \]
  • $X$ and $Y$ are functionally identical, but different binaries

  \[
  S_A \rightarrow X \rightarrow X_1 \quad \quad \quad \quad \quad S_A \rightarrow Y \rightarrow Y_1
  \]
  • $X_1$ and $Y_1$ must be identical binaries (since $X$, $Y$ were functionally identical)!
Related Topics

Diminishing distinction between civilian and military/industrial security posture:
• Bruce Schneier blog post: https://www.lawfareblog.com/myth-consumer-security
• Ability to source trustworthy microchips drowned out by consumer market
• https://youtu.be/1uCy-T22el8?t=132

Right To Repair:
• automobiles, electronics, agricultural machinery
• issues of ownership, control, trust: all aspects of security