CheriBSD
Hybrid-capability OS prototype

Robert N. M. Watson

University of Cambridge, SRI International

CHERI Microkernel Workshop – 23 April 2016
CHERI software models

More compatible  Safer

Unmodified  Hybrid  Pure-capability
All pointers are registers  Annotated and automatically selected pointers are capabilities  All code and data pointers are capabilities

- **Source and binary compatibility:** common C-language idioms, various ABIs
  - **Unmodified code:** Existing n64 code runs without modification
  - **Hybrid code:** e.g., used solely in return addresses, for annotated data/code pointers, for specific types, stack pointers, etc.; n64-interoperable.
  - **Pure-capability code:** ubiquitous data-pointer protection, strong Control Flow Integrity (CFI). Non-n64-interoperable.

- **CHERI Clang/LLVM prototype** generates code for all three
Software deployment models

Hybrid capability/MMU OSes

Hybrid MMU-capability models: protection and compartmentalization within virtual address spaces

Single-address-space systems are possible but not yet our focus
Key software hypotheses

• Viable composition of capability and MMU models for software
  • E.g., CHERI complements paged VM in practical systems
• CHERI capabilities can be usefully applied to program constructs
  • E.g., heap/stack allocations, code pointers, return addresses
  • E.g., kernel-provided memory mappings, static + run-time linking
• Strong binary and source-code compatibility; incremental deployment
  • E.g., selected libraries, applications within a larger system
• Platform for compartmentalization research
  • Libraries/applications are efficiently/easily compartmentalized
  • But also kernel code (in due course)
CheriBSD

• Based on open-source FreeBSD operating system
• “Minimalist” kernel adaptation
  • Process model, VM, debugging, signals support capabilities
  • E.g., thread state includes capability registers
  • E.g., tags preserved for swapped anonymous memory
  • Kernel actually compiled with MIPS, not CHERI, compiler
• Multiple process ABIs: hybrid MIPS and CheriABI
• Fine-grained, in-address-space compartmentalization model
  • Kernel-assisted domain transition, fault handling
  • libcheri object-capability runtime
Multiple process ABIs

- **64-bit MIPS ABI** supports highly compatible hybrid code execution, traditional pointer-based system calls
- **CheriABI** binaries/processes are pure-capability code throughout; system-call interface enforces user model
Demonstration applications

- Pure-capability libraries and applications
  - Pure-capability compilation of all key system libraries and increasing number of commands – e.g., OpenSSL, OpenSSH
  - Strong memory protection for heap, stack; control-flow integrity for minimally modified or unmodified applications
- Library compartmentalization
  - Transparent, efficient sandboxing of security-critical libraries
- tcpdump compartmentalization
  - Fine-grained: multiple domain transitions per packet
- Otherwise (essentially) unmodified userspace
Next directions

• Short-term: complete pure-capability userspace
  • CHERI-aware run-time linking, multithreading
  • Remainder of C (and C++) pure-capability userspace
  • LLDB debugger support
• Short-term: selected capability use and CFI within the kernel
  • E.g., in CheriABI, for user-originated pointers, network stack
• Longer-term: selectively compartmentalized kernel
  • CHERI-based microkernel within CheriBSD kernel
• Longer-term: non-volatile memory + capabilities
  • Semantics for tagged capabilities within filesystem objects
BACKUP SLIDES
# Kernel Changes

<table>
<thead>
<tr>
<th>Component</th>
<th>File</th>
<th>Lines +</th>
<th>Lines -</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine-dependent headers</td>
<td>19</td>
<td>1424</td>
<td>11</td>
</tr>
<tr>
<td>CHERI initialization</td>
<td>2</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>Context management</td>
<td>2</td>
<td>392</td>
<td>10</td>
</tr>
<tr>
<td>Exception handling</td>
<td>3</td>
<td>574</td>
<td>90</td>
</tr>
<tr>
<td>Memory copying</td>
<td>2</td>
<td>122</td>
<td>0</td>
</tr>
<tr>
<td>Virtual memory</td>
<td>5</td>
<td>398</td>
<td>27</td>
</tr>
<tr>
<td>Object capabilities</td>
<td>2</td>
<td>883</td>
<td>0</td>
</tr>
<tr>
<td>System calls</td>
<td>2</td>
<td>76</td>
<td>0</td>
</tr>
<tr>
<td>CheriABI</td>
<td>6</td>
<td>2855</td>
<td>0</td>
</tr>
<tr>
<td>Signal delivery</td>
<td>3</td>
<td>327</td>
<td>71</td>
</tr>
<tr>
<td>Process monitoring/debugging</td>
<td>3</td>
<td>298</td>
<td>0</td>
</tr>
<tr>
<td>Kernel debugger</td>
<td>2</td>
<td>264</td>
<td>0</td>
</tr>
</tbody>
</table>