CHERI
A Hybrid Capability-System Architecture for Scalable Software Compartmentalization

Robert N.M. Watson*, Jonathan Woodruff*, Peter G. Neumann†, Simon W. Moore*,
Jonathan Anderson‡, David Chisnall*, Nirav Dave†, Brooks Davis†, Khilan Gudka*, Ben Laurie§,
Steven J. Murdoch¶, Robert Norton*, Michael Roe*, Stacey Son, and Munraj Vadera*

*University of Cambridge, †SRI International, ‡Memorial University,
§Google UK Ltd, ¶University College London

IEEE Symposium on Security and Privacy
18 May 2015
Application compartmentalization mitigates vulnerabilities by decomposing applications into isolated compartments delegated **limited rights**.
• Many possible compartmentalizations:
  • Trade off security, complexity, performance

• But the process model is problematic:
  • Virtual addressing scales poorly due to page tables, Translation Look-aside Buffer (TLB)
  • Multiple address spaces and Inter-Process Communication (IPC) are hard to program

• Quite poor for library compartmentalization due to memory-centered APIs (e.g, zlib)
CHERI capability model

- **ISCA 2014**: Fine-grained, in-address-space memory protection via a capability model
  - **Capabilities** replace pointers for data references
  - **Capability registers** and **tagged memory** enforce strong pointer and control-flow integrity, bounds checking
  - Hybrid model composes naturally with an MMU
- **ASPLOS 2015**: Compiler support for capabilities
  - Converge **fat-pointer** and **capability** models
  - C pointers compiled into capabilities with various ABIs
- **Can we build efficient compartmentalization over CHERI memory protection?**
## Virtual memory vs. capabilities

<table>
<thead>
<tr>
<th></th>
<th>Virtual Memory</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Protects</strong></td>
<td>Virtual addresses and pages</td>
<td>References (pointers) to C code, data structures</td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td>MMU, TLB</td>
<td>Capability registers, tagged memory</td>
</tr>
<tr>
<td><strong>Costs</strong></td>
<td>TLB, page tables, lookups, shootdowns</td>
<td>Per-pointer overhead, context switching</td>
</tr>
<tr>
<td><strong>Compartment scalability</strong></td>
<td>Tens to hundreds</td>
<td>Thousands or more</td>
</tr>
<tr>
<td><strong>Domain crossing</strong></td>
<td>IPC</td>
<td>Function calls</td>
</tr>
<tr>
<td><strong>Optimization goals</strong></td>
<td>Isolation, full virtualization</td>
<td>Memory sharing, frequent domain transitions</td>
</tr>
</tbody>
</table>

CHERI hybridizes the models: pick two!
Hybrid capability/MMU OSes

Legacy application + capability libraries

Pure-capability application

Capability-based OS with legacy libraries

CHERI CPU

zlib
libssl

class1
libssl

class2

Address-space executive

libssl
zlib

class1

Address-space executive

Virtual address spaces

Single address space

Virtual address spaces

Single address space
CHERI capabilities

- **Sealed bit** prevents further modification
- **Object types** atomically link code, data capabilities
- **CCall/CReturn** instructions provide hardware-assisted, software-defined domain transitions

Virtual address space
CheriBSD object capabilities

- In-process object-capability model
- libcheri loads and links classes, instantiates objects
- Per-thread capability register file describes its protection domain
- Domain transition within threads via register-file transformation
- CCall/CRReturn exception handlers unseal capabilities, allow delegation
- Trusted stack provides reliable software-defined return, recovery
- Many other software-defined models possible; e.g., asynchronous closures
Object-capability call/return

- Initial registers after execve() grant ambient authority
- Synchronous function-like call eases application/library adaptation
- CCall/CReturn ABI clears unused registers to prevent leakage
- Only authorized system classes can make system calls
- Constant overhead to function-call cost
CHERI hardware/software prototypes

- Bluespec FPGA prototype
  - 64-bit MIPS + CHERI ISA
  - Pipelined, L1/L2 caches, MMU
  - Synthesizes at ~100MHz
- Capability-aware software
  - CheriBSD OS
  - CHERI clang/LLVM compiler
  - Adapted applications
- Open-source release
# Application implications

<table>
<thead>
<tr>
<th><strong>Pros</strong></th>
<th><strong>Cons</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Single address-space</td>
<td>Still have to reason about the security properties</td>
</tr>
<tr>
<td>programming model</td>
<td></td>
</tr>
<tr>
<td>Referential integrity matches</td>
<td>Shared memory is more subtle than copy semantics</td>
</tr>
<tr>
<td>programmer model</td>
<td></td>
</tr>
<tr>
<td>Modest work to insert</td>
<td>Capability overhead in data</td>
</tr>
<tr>
<td>protection-domain boundaries</td>
<td>cache is real and measurable</td>
</tr>
<tr>
<td>Objects permit mutual distrust</td>
<td>ABI subtleties between MIPS and CHERI compiled code</td>
</tr>
<tr>
<td>Constant (low) overhead</td>
<td>Lower overhead raises further cache side-channel concerns</td>
</tr>
<tr>
<td>relative to function calls</td>
<td></td>
</tr>
<tr>
<td>even with large memory flows</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

• Hybrid object-capability model over memory capabilities
  • Software-defined, fine-grained, in-address-space compartmentalization
  • Cleanly extends the MMU-based process model
  • Targets C-language userspace TCBs
  • Non-IPC model supports library compartmentalization
  • Orders of magnitude more efficient compartmentalization than conventional designs
• Open-source reference implementation, ISA specification:
  [link]
    http://www.cheri-cpu.org/