CHERI
A Hybrid Capability-System Architecture for Fine-grained Memory Protection and Compartmentalization


University of Cambridge, SRI International

May 2015
Application compartmentalization mitigates vulnerabilities by decomposing applications into isolated compartments delegated limited rights.
Compartimentalization vision
• A single application has many possible compartmentalizations
  • Each trade off security against performance and programming complexity
• But the process model is problematic:
  • Limited simultaneous-process scalability
  • Multi-address-space programming model
The process-model consensus

- Coarse-grained process isolation
  - Inter-program robustness
  - Multi-user access control
- Memory Management Unit (MMU)
  - Page tables control per-process virtual-to-physical mappings
  - Translation Look-aside Buffer (TLB)
- Bridged via Inter-Process Communication (IPC) and other kernel services (e.g., filesystem)
- Inefficient and inadequate foundation for granular memory protection
- Inefficient and hard-to-program foundation for granular software compartmentalization
If you could revise the fundamental principles of computer system design to improve security…

...what would you change?
CHERI capability model

- **ISCA 2014**: Fine-grained, in-address-space memory protection
  - **Capabilities** replace pointers for data references
  - **Data-pointer integrity, control-flow integrity, bounds checking**
  - **Hybrid model** lives side-by-side with a conventional MMU
- **ASPLOS 2015**: Explore and refine C-language compatibility
  - Converge **fat-pointer** and **capability** models
  - Develop **binary-compatibility models**
- **Oakland 2015**: Software compartmentalization with CHERI
  - **Object-capability model** implemented over hardware capabilities
  - Efficient, in-address-space **software-defined domain transition**
CHERI MEMORY PROTECTION
Revisiting RISC in an age of risk

• Fine-grained, capability-based in-address-space protection
  • Deconflate virtualization and protection – retain MMU
  • Implement spatial protection (e.g., bounds checking)
  • Foundation for temporal protection (e.g., GC)
  • Foundation for fine-grained compartmentalization

• A RISC approach
  • Instructions are for compilers
  • Unprivileged fast paths, software slow paths
  • Prototype on 64-bit MIPS, but more broadly applicable
## 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 these models: pick two!**
Capabilities are fat pointers with strong integrity properties

- Guarded manipulation enforced monotonic rights decrease
- Tags protection integrity; can’t dereference invalid capability
- Tagged memory maintains integrity in RAM
Capability extensions to pipeline

- Capability coprocessor provides capability registers, instructions
- Interposes on legacy MIPS load/store instructions, instruction fetch
- Processing ‘before’ MMU makes capabilities address-space relative
- Tag controller associates tags with in-memory capabilities
- Under the hood: memory partitioned, with a region holding all tags
Our focus: hybridizing conventional virtual-memory security and in-address-space capability systems

Single-address-space systems are also possible on CHERI
C-language memory protection

- Compiler implements C pointers in terms of capabilities
- Strong pointer integrity and use limits
  - Tag detects any pointer corruption
  - Bounds checking, with subsetting
  - Permissions constrain use (e.g., read-only, W^X)
- Protects data and control flow integrity
  - Out-of-bounds data access eliminated
  - Data-pointer confusion eliminated
- Support for accurate garbage collection
Binary compatibility

More compatible

N64
Pure MIPS

Hybrid
Some pointers are capabilities

Pure-capability
All pointers are capabilities

Safer

• MIPS code lives side-by-side with CHERI code

• Incremental adoption options – e.g., shifting to capabilities for return addresses, just stack pointers, etc.
CHERI OS considerations

- Prototyped using the FreeBSD operating system (+/- 4 KLoC)
  - Changes for user memory protection, compartmentalization
- Process model extended for tagged capabilities
  - Register-file setup and maintenance (exec, switch, thread create)
  - Virtual-memory support for physical tags
  - Signal-handling, debugging extensions
- Fine-grained, in-address-space object-capability security model
  - Kernel CCall/CReturn exception handlers
  - System calls blocked from non-system classes
  - Userspace compartmentalization runtime
CHERI
COMPARTMENTALIZATION
CheriBSD object capabilities

- In-process **object-capability model**
- Per-thread capability register file describes its **protection domain**
- **Domain transition** within threads via register-file transformation
- Object capabilities support strong **encapsulation, mutual distrust**
- **libcheri** implements **classes, objects**
- **CCall/CReturn** exception handlers unseal capabilities
- Capabilities passed via call, return
- **Trusted stack** provides reliable software-defined return, recovery
Supporting object capabilities

- **Sealed bit** prevents modification, dereferencing: *encapsulation*
- **Object types** atomically link *code, data* pairs to create *objects*
- **Object invocation** mechanism *unseals* capability pairs
  - Userspace TCB manages object-type namespace
Object-capability call/return

- Initial registers after execve() grant ambient authority
- Synchronous function-like call fits current application/library programming styles
- CCall/CReturn ABI clears unused registers to prevent data or capability leakage
- Only authorized system classes can make system calls
- Constant overhead to function-call cost
Application implications

**Pros**

- Single address-space programming model
- Referential integrity matches programmer model
- Only modest work to insert protection-domain boundaries
- Objects permit mutual distrust
- Constant (low) overhead relative to function calls even with large memory flows

**Cons**

- Still have to reason about the security properties
- Shared memory is more subtle than copy semantics
- Capability overhead in data cache is real and measurable
- ABI subtleties between MIPS and CHERI compiled code
- Lower overhead raises further cache side-channel concerns
CHERI
COMPARTMENTALIZATION
PERFORMANCE
CHERI hardware/software prototypes

- Bluespec FPGA prototype
  - 64-bit MIPS + CHERI ISA
  - Pipelined, L1/L2 caches, MMU
  - Synthesizes at ~100MHz
  - Realistic (modest) illustration of what could be accomplished in silicon designs

- Capability-aware software
  - CheriBSD OS
  - CHERI clang/LLVM compiler
  - Adapted applications

- Open-source release
Updated ISCA/ASPLOS results looking at the cache-footprint question?

Include 128-bit simulation
Domain-switching overhead

The graph shows the cycles overhead for different communication methods as a function of payload/bytes. The methods include:

1: socket
2: pipe
3: shmem+pipe
4: shmem+sem
5: pthread+sem
6: invoke
7: func

The x-axis represents the payload in bytes, while the y-axis represents the cycles overhead.
Library compartmentalization

Fig. 12. Compression time for gzip with various sandboxing mechanisms.

Creation and destruction to instantiate and destroy compartments; (2) the amortized and indirect costs of additional virtual address spaces, such as TLB contention and kernel-data-structure footprint on the cache; and (3) the cost of protection-domain switching and implied copying (or MMU operations) to pass data between compartments using IPC.

In some cases, these overheads are negligible or amortized. For example, Capsicum allows processes to acquire access to capabilities while operating with ambient authority, and then to enter capability mode without creating an additional process. For simple program structures, such as in tcpdump, Capsicum sandboxing adds only a few extra system calls to limit future.

Fig. 13. Sandbox creation overhead: time taken to compress varying number of files of size 500,000 bytes with different zlib implementations.

Fig. 14. gif2png with unmodified and CHERI zlib implementations.
ONGOING REFINEMENT TO THE CHERI MODEL
Hardware and software refinement

• Architecture performance / cost
  • Re-converge register files
  • 128-bit compressed capabilities
  • Explore and evaluate application to a non-MIPS ISAs

• Software model and compatibility
  • Compiler and ABI refinement
  • Software compartmentalization models
  • Hybrid system call layer
  • Complete capability-ABI FreeBSD version
  • Reliable and accurate garbage collection
Capability compression

- Exchange precision for reduced register size, cache footprint
- Floating-point style representation
- Unlike prior schemes, support pointer “out of bounds” for C
- Important: retain monotonicity for safe delegation!
- Great care must be taken with security—performance trade-offs

```
mask = 0xfffffffffffffffff << exp
top = (pointer + (toTop << exp)) & mask
base = (pointer + (toBase << exp)) & mask
```

Virtual address space

128-bit capability

- 1-bit tag
- 1-bit tag

- perms (23 bits)
- exp (6)
- toBase
- toTop
- s

pointer (64 bits)
Papers and reports


Conclusions

• Hybrid capability model for fine-grained, in-address-space memory protection
  • Supplements current MMU
  • Targets C-language TCBs
• Object-capability model for granular, in-address-space compartmentalization
  • Software-defined domain-transition model
  • Reference-oriented programmer model restored
  • Efficient non-IPC communication with protection
• Open-source reference implementation, ISA specification:
  http://www.cheri-cpu.org/
BACKUP SLIDES
Domain-switch pseudocode

CCall

CReturn

TODO
## CCale - cycle-by-cycle analysis
(Including user and kernel space)

<table>
<thead>
<tr>
<th>Module</th>
<th>Phase</th>
<th>#instr.</th>
<th>#cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caller</td>
<td>Setup call, clear unused argument registers</td>
<td>22 *</td>
<td>42 *</td>
</tr>
<tr>
<td>libcheri</td>
<td>Save callee-save regs, push call frame</td>
<td>30 **</td>
<td>34 **</td>
</tr>
<tr>
<td>Kernel</td>
<td>Receive trap</td>
<td>13</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Validate CCale args</td>
<td>79 * **</td>
<td>79 * **</td>
</tr>
<tr>
<td></td>
<td>Push trusted stack, unseal Call args.</td>
<td>31</td>
<td>41 **</td>
</tr>
<tr>
<td></td>
<td>Clear non-argument registers</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Exit kernel</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Sandbox</td>
<td>Set up sandbox</td>
<td>33</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td></td>
<td>219</td>
<td>299</td>
</tr>
</tbody>
</table>

* Further opportunities for hardware optimization  
** Will further reduce with smaller registers, converged register file
CReturn cycle-by-cycle analysis
(Including user and kernel space)

<table>
<thead>
<tr>
<th>Module</th>
<th>Phase</th>
<th>#instr.</th>
<th>#cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandbox</td>
<td>Exit sandbox</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>Kernel</td>
<td>Receive trap</td>
<td>13</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>Validate return capability</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Pop trusted stack</td>
<td>26</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>Clear non-return registers</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Exit kernel</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>libcheri</td>
<td>Pop call frame, restore regs.</td>
<td>28</td>
<td>52</td>
</tr>
<tr>
<td>Caller</td>
<td>Back in caller</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

* Further opportunities for hardware optimization
** Will further reduce with smaller registers, converged register file