Beyond the PDP-11: Architectural support for a memory-safe C abstract machine

David Chisnall†, Colin Rothwell†, Brooks Davis‡, Robert N.M. Watson†, Jonathan Woodruff†, Munraj Vadera†, Simon W. Moore†, Peter G. Neumann‡, and Michael Roe†

†University of Cambridge
‡SRI International

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 and FA8750-11-C-0249. The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.
Why bring the PDP-11 into it?

- First target for C
- Flat, byte-addressable memory
- C split memory into objects purely in software
- All widely deployed C implementations follow this model
Memory safety for compartmentalisation

- Processes are isolated by hardware (MMU), but expensive
- Fine-grained compartmentalisation needs:
  - Cheap compartments
  - Fine-grained sharing
From compartments to objects

- Sharing requires pointers with enforced bounds and permissions
- Can we use this mechanism for every pointer?
The initial CHERI ISA

- All memory accesses via a capability register
- ISA allows reducing capabilities
- Tagged memory protects capabilities
Binary compatibility

More compatible

n64
Pure MIPS

Hybrid
Some pointers are capabilities

Pure-capability
All pointers are capabilities

More safe
The prototype CPU

- 64-bit MIPS-compatible ISA ($\approx$ R4000)
- CHERI ISA extensions
- Runs at 100MHz on FPGA
- Full software stack
Real world code

• A lot of C is implementation defined
• Most real C code does interesting things with pointers
• Case study: `tcpdump` does most of them (on untrusted data, running as root)

Supporting just the standard isn’t enough
Common pointer idioms

• Full list in the paper
• Around 2M lines of C code surveyed
• Thousands of instances found
• Breaking them is not acceptable!
Example: The mask idiom

// The low bit of an aligned pointer is always 0, so we can hide a flag in it
int *set_flag(int *b)
{
    return (int*)((intptr_t)b | 1);
}
Example: The mask idiom

// The low bit of an aligned pointer is always 0, so we can hide a flag in it
int *set_flag(int *b)
{
    return (int*)((intptr_t)b | 1);
}

00x1601231231
Example: Invalid Intermediates
Example: Invalid Intermediates

```
Pointer += x;
```

[Diagram showing: Pointer, Buffer, End]
Example: Invalid Intermediates

Pointer += x;
if (Pointer > End)
Example: Invalid Intermediates

```c
Pointer += x;
if (Pointer > End)
    Pointer = End - 1;
```
Capabilities

Unforgeable

Monotonic length and permissions

Grant rights

Old CHERI Capabilities:

- Base [64]
- Length [64]
- Permissions [32]  Type [24]
- Experimental [136]
Fat Pointers

Describe a point

Add metadata
Capabilities + Fat Pointers

Unforgeable

Monotonic length and permissions

Grant rights

Describe a point

Add metadata
Capabilities + Fat Pointers

Unforgeable
Describe a point
Monotonic length and permissions
Add metadata
Grant rights
### New CHERI Capabilities

- CHERI capabilities extended to include an offset field
- Checks apply only on dereference

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>64</td>
</tr>
<tr>
<td>Length</td>
<td>64</td>
</tr>
<tr>
<td>Permissions</td>
<td>32</td>
</tr>
<tr>
<td>Type</td>
<td>24</td>
</tr>
<tr>
<td>Reserved</td>
<td>8</td>
</tr>
<tr>
<td>Offset</td>
<td>64</td>
</tr>
</tbody>
</table>
It’s alive!

• Fully supports real-world C pointer use.
• Negligible overhead in tcpdump
• More performance evaluation in the paper
Conclusions

• We have shown that a capability model can provide a memory-safe C abstract machine

• This paves the way for fine-grained compartmentalisation of C programs

• Come and see us at IEEE Security and Privacy for the next part of the story!

http://chericpu.org