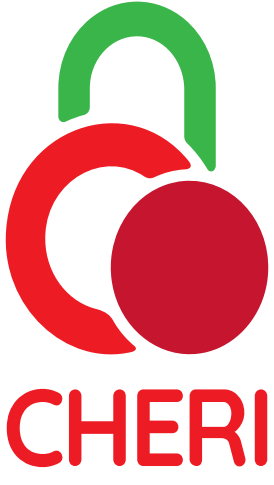
CHERI Linking through the Ages



John Baldwin / Ararat River Consulting, LLC
Jessica Clarke / University of Cambridge

CHERI Blossoms 2026

SRI UNIVERSITY OF CAMBRIDGE

Prologue

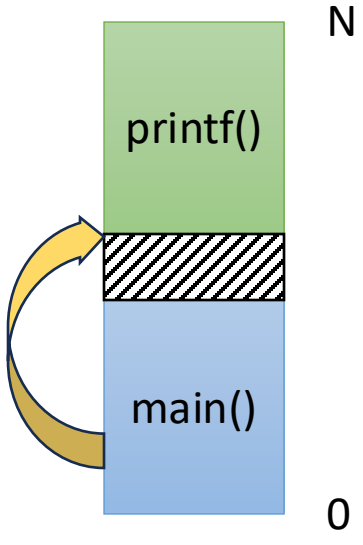
- More slides than time
- May gloss over less important bits of history
- Wanted to document this history in one place
- Slides and recording will be available if interested in more

Basic Problem of Linking

- Source code refers to functions and data objects by symbol names
- Machine code refers to functions and data by addresses (or, now, capabilities)
- How do we bridge the gap?

```
#include <stdio.h>

int
main(void)
{
        printf("hello world\n");
        return (0);
}
```



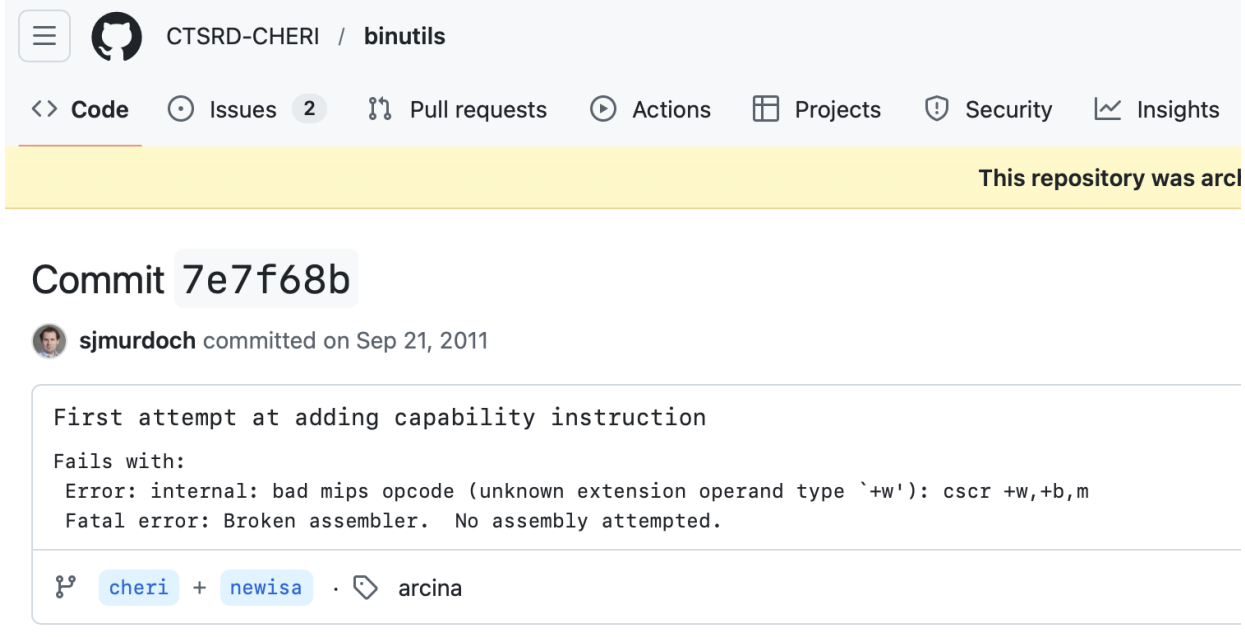
Unique Challenges of Linking for CHERI

- CHERI capabilities must be derived at runtime
  - Even position-dependent static binaries require runtime initialization
  - Runtime initialization requires additional data beyond address relocation
    - Permissions
    - Bounds

- CHERI capabilities are not the same size as an ELF address
  - For example, addend field in Elf_Rela cannot hold a template capability

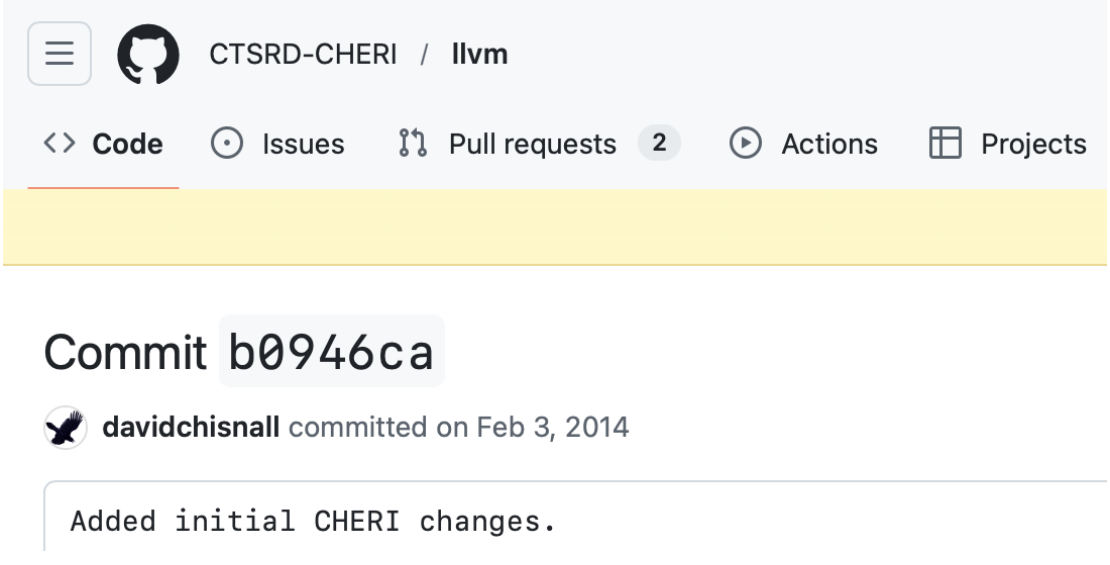
Stone Age: Assembling the First Tool

- In the beginning (October 2010) there was no toolchain
- Within a year, CHERI started to gain a GNU binutils assembler



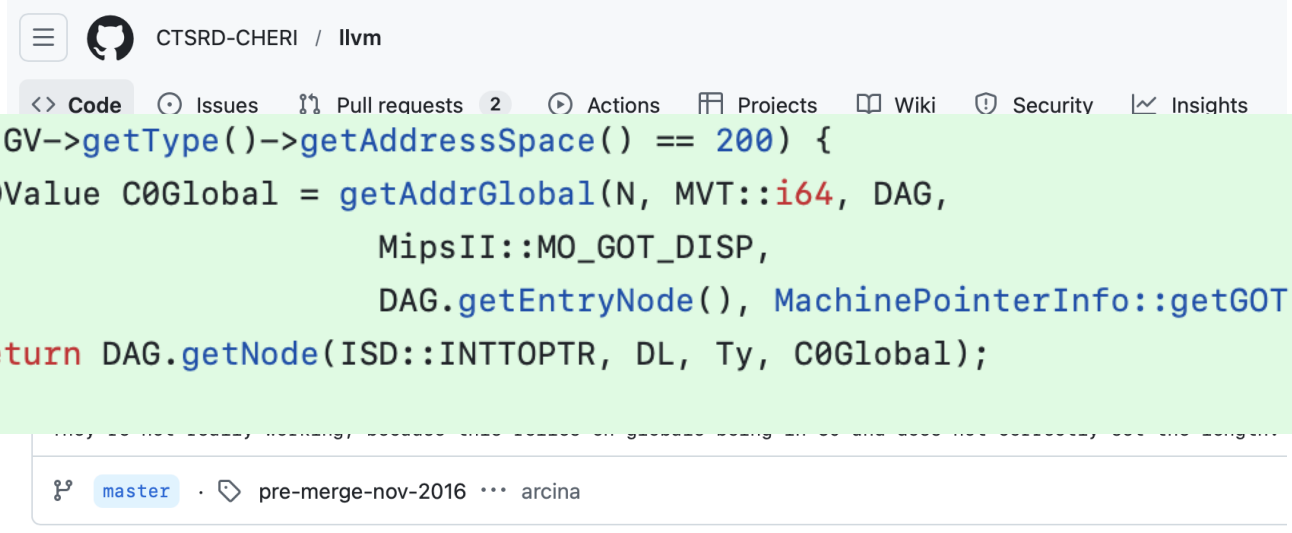
Stone Age: From Gnus to Wyverns

- Early work continued with binutils
- Fast forward a few years and CHERI LLVM appears with the start of code generation, not just an assembler



Stone Age: When All You Have Is a Compiler

- If you don't have a CHERI-aware linker, where do capabilities come from?
- In code: load address as if non-CHERI and dynamically derive from DDC (called C0 at the time), but no bounds set yet



Stone Age: When All You Have Is a Compiler

- What about global data needing initialisation?
  `int *p = &x; // where p is a global`
  - For non-CHERI, either hard-coded address in binary or initialised by run-time loader
- Compiler emits new __cap_relocs section, CHERI-unaware linker automatically concatenates like any other section
-

Stone Age: When All You Have Is a Compiler

- “Compiler emits new __cap_relocs section, CHERI-unaware linker automatically concatenates like any other section”
- … does this sound familiar to anyone?
- C

```
technos:rootfs-riscv64-purecap jrtc4% cheri-llvm llvm-readelf -WS lib/libc.so.7
There are 38 section headers, starting at offset 0x2085d0:

Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .dynsym           DYNSYM          0000000000000350 000350 013aa0 18   A  7   1  8
  [ 2] .gnu.version      VERSYM          0000000000013df0 013df0 001a38 02   A  1   0  2
  [ 3] .gnu.version_d    VERDEF          0000000000015828 015828 000134 00   A  7  11  4
  [ 4] .gnu.version_r    VERNEED         000000000001595c 01595c 000020 00   A  7   1  4
  [ 5] .gnu.hash         GNU_HASH        0000000000015980 015980 006174 00   A  1   0  8
  [ 6] .hash             HASH            000000000001baf4 01baf4 0068e8 04   A  1   0  4
  [ 7] .dynstr           STRTAB          00000000000223dc 0223dc 00a8fa 00   A  0   0  1
  [ 8] .rela.dyn         RELA            000000000002ccd8 02ccd8 0010b0 18   A  1   0  8
  [ 9] .rela.plt         RELA            000000000002dd88 02dd88 0055b0 18  AI  1  16  8
  [10] __cap_relocs      PROGBITS        0000000000033338 033338 03e828 28   A  0   0  8
```

Stone Age: When All You Have Is a Compiler

CheriBSD 6abb68c6f00a:lib/csu/mips/crt1_c.c

```
struct capreloc
{
        uint64_t capability_location;
        uint64_t object;
        uint64_t offset;
        uint64_t size;
        uint64_t permissions;
};
__attribute__((weak))
extern struct capreloc
__start___cap_relocs;
__attribute__((weak))
extern struct capreloc
__stop___cap_relocs;
```

s/cheri_init_globals.h

```
__SIZE_TYPE__ capability_location;
__SIZE_TYPE__ object;
__SIZE_TYPE__ permissions;
__attribute__((weak)) extern struct
capreloc __start___cap_relocs[];
__attribute__((weak)) extern struct
capreloc __stop___cap_relocs[];
```

Initialised as normal 64-bit addresses

Compile-time constant

Aspirational for now

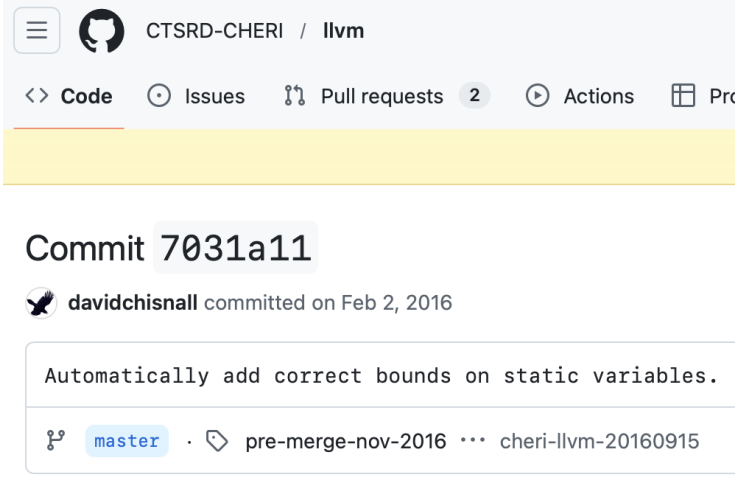
† 9e82d2969a62 as of 23rd March 2026

Stone Age: Boundary Issues

- __cap_relocs included “size” field; what sets it?
- Filled by new “capsizefix” tool, post-link step, matching entries to ELF symbols by address
  - ELF symbols include “st_size” field, copied to corresponding entry

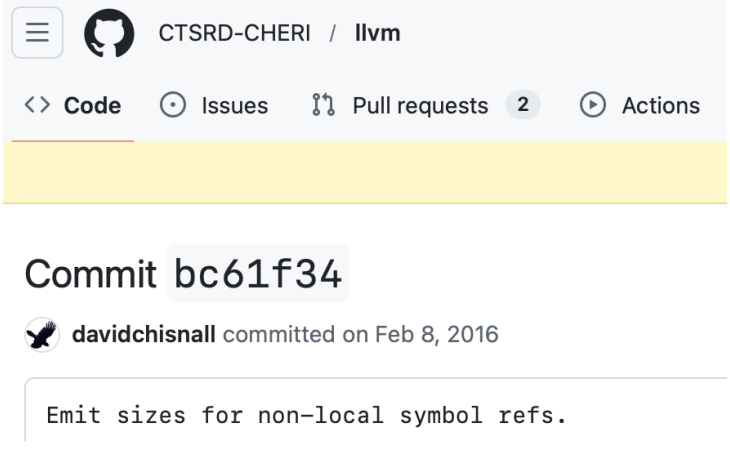
Stone Age: Boundary Issues

- What about for our “derive dynamically from DDC” case?
- Static variables’ sizes known at compile time



Stone Age: Boundary Issues

- Still no CHERI-aware linker… but what if we don't know what 'x' is until link time?
- When compiling file defining '`T x`' now automatically emit new '`const size_t .size.x = sizeof(x)`' (in assembly)
- When deriving capability for 'x', can now load value in '.size.x' and use as input to CSetBounds



Stone Age: Boundary Issues

```
int *
f(void)
{
        return (&x);
}
```

1. Load address of x from GOT via DDC
2. Load address of .size.x from GOT via DDC
3. Load from .size.x via DDC
4. Convert address of x to capability using DDC
5. CSetBounds using loaded size

Five instructions, three loads, dependency chains, all every time f is run… and DDC is sitting around the entire time with whole-address-space bounds

Stone Age: Boundary Issues

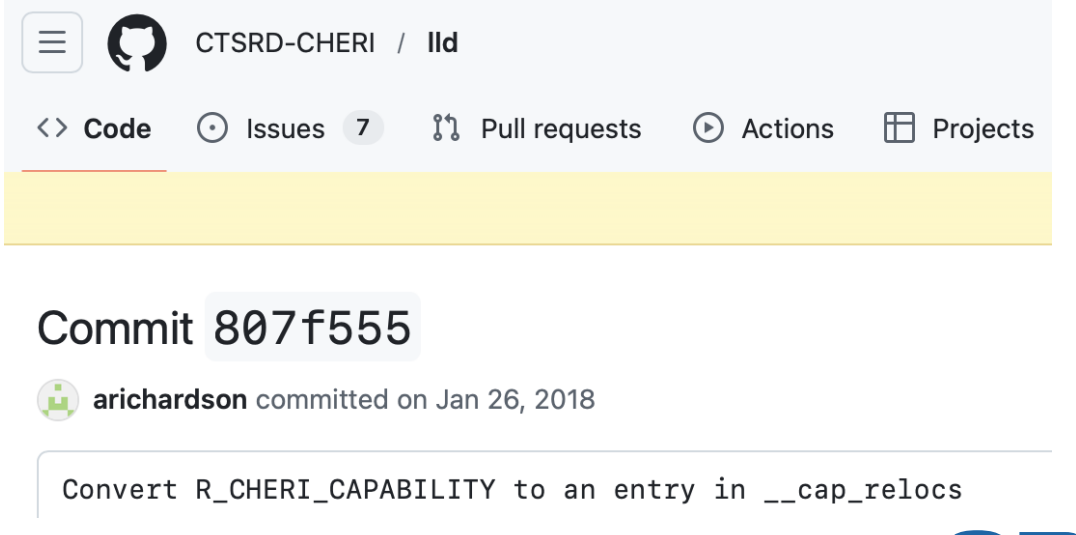
- What about dynamic linking?
- So long as x is preempted if and only if .size.x is, derivation in code works
- For __cap_relocs though… good luck?
  - capability_location (where the capability lives) and object (where the base should point to) relocated via normal ELF relocations
    - Two step relocation process – “relocate the relocation itself” 🤯
  - offset always a compile-time constant
  - permissions (at this point, capsizefix uses symbol type to write “data or function” to field)? size? Definitions may live in another library…

Bronze Age: Sinking capsizefix into LLD

- First step to having a CHERI-aware linker: replace the separate capsizefix with integrated handling in LLD

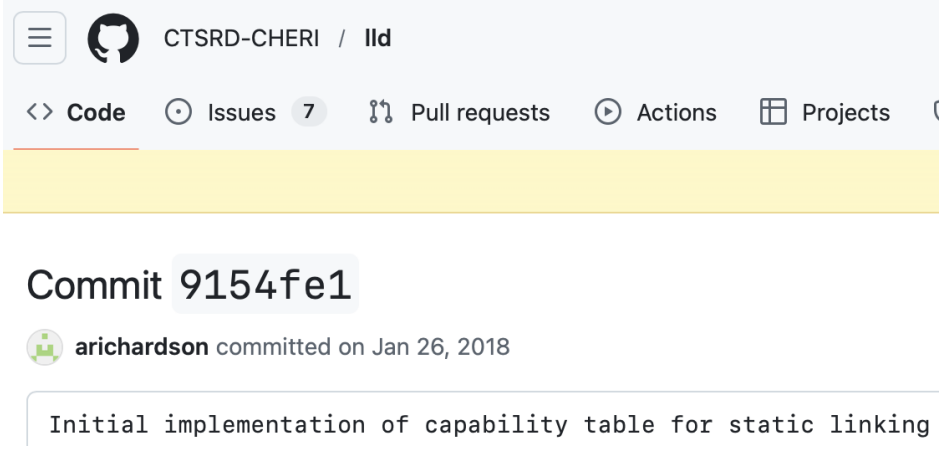
Bronze Age: No Going Back

- Now we have a CHERI-aware linker, __cap_relocs is unhelpful for it; have to match entries back to ELF relocations for individual fields
- Compiler now emits new “normal” ELF relocation, leaves linker to generate __cap_relocs for runtime



Bronze Age: Have You GOT a Table?

- With a CHERI linker, we can now build a capability version of the normal GOT
  - Runtime initialises it via __cap_relocs at program startup
  - Getting '&x' is now a single (capability) load from '.captable' section, no more dynamic derivation from DDC

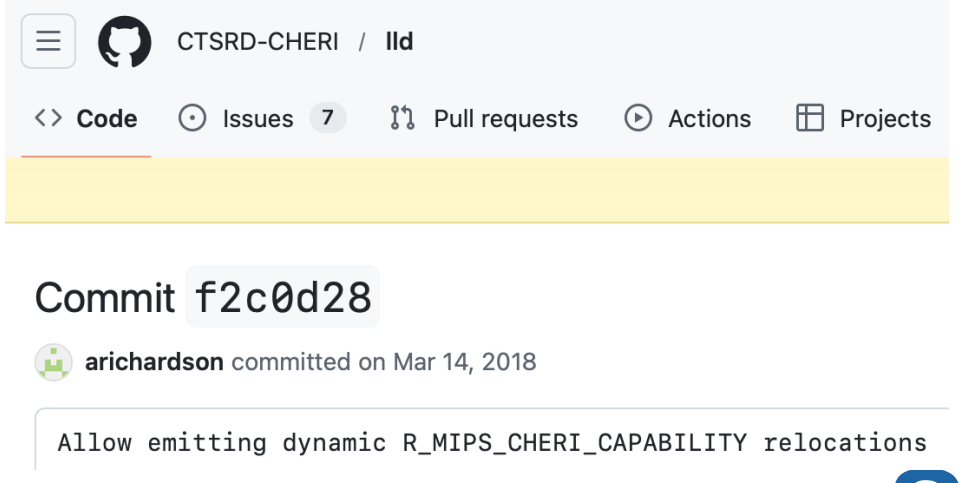


Bronze Age: Bounding Ahead

- Unlike capsizefix, linker sees libraries being linked against
  - Can look up sizes and types for caprelocs against external symbols; dynamic linking now less broken

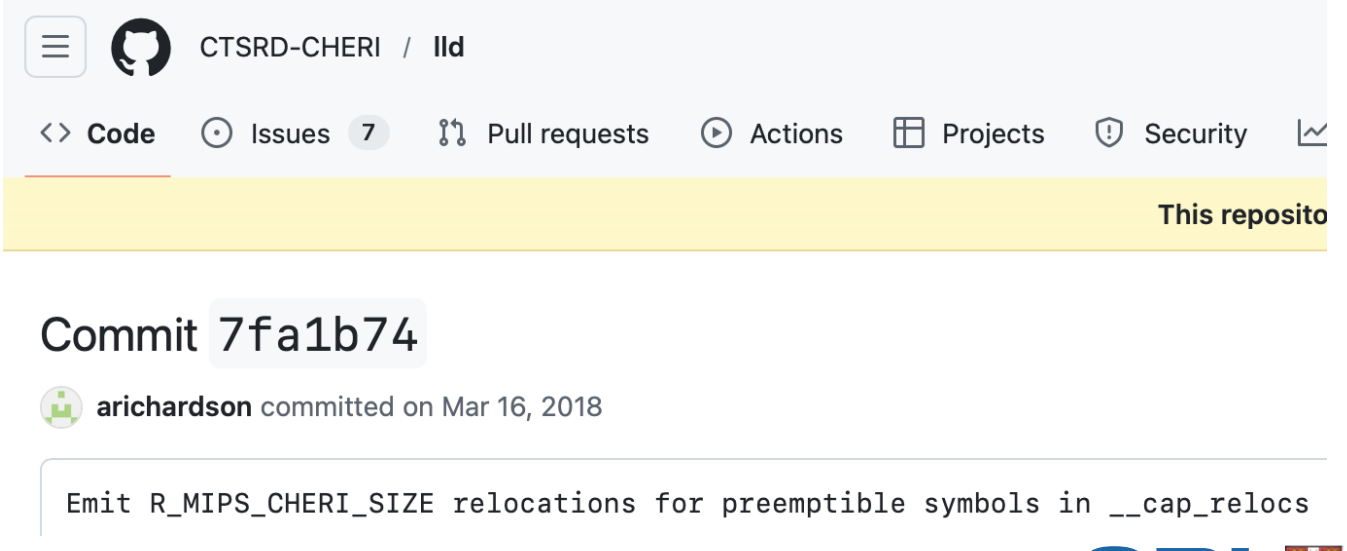
Bronze Age: A Symbolic Move

- __cap_relocs not designed for dynamic relocations (i.e. referring to symbols resolved at load/run time)
- ELF relocation for “object” (base) already has symbol in it
- Can replace it all with a single new ELF relocation; runtime can read size / permissions when looking up symbol



Bronze Age: Putting Lipstick on a Pig

- At the same time, __cap_relocs extended with new relocation to fill in “size” field at run time when needed as a stop-gap
  - Easier to add a new integer relocation than teach run-time loader to handle capabilities in normal ELF relocation handling



8 years in

CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment

Brooks Davis*
brooks.davis@sri.com

Robert N. M. Watson†
robert.watson@cl.cam.ac.uk

Alexander Richardson†
alexander.richardson@cl.cam.ac.uk

Peter G. Neumann*
peter.neumann@sri.com

Simon W. Moore†
simon.moore@cl.cam.ac.uk

John Baldwin‡
john@araratriver.co

David Chisnall§

Jessica Clarke†

Na[illegible]hanie[illegible] Vesle[illegible] Fila[illegible]do†

This is the ABI used in our ASPLOS 2019 paper

CHERI

SRI UNIVERSITY OF CAMBRIDGE

Bronze Age: Relatively Constant Progress

- "`const int x, *p = &x;`" gave a read/write capability; relied on MMU to stop writes
  - __cap_relocs permissions field extended to include "constant" flag

Bronze Age: Relatively Constant Progress

- R_MIPS_CHERI_CAPABILITY now used for symbolic relocations
- __cap_relocs now only used for local (“non-preemptible”) symbols
  - size / permissions now always constants, no relocations needed
  - capability_location and object still need to be absolute addresses – ELF relocations emitted in position-independent binaries to add load address, the same for both fields, and for every relocation
- Runtime can instead implicitly add this load address

9 years in

UKRI Digital Security by Design: A £190M research programme around Arm's Morello – an experimental ARMv8-A CPU, SoC, and board with CHERI support

2019-10-18 Open-source security, Operating systems, Processors, Programming languages, Uncategorized Arm, CHERI, Morello Robert N. M. Watson

Commit 61c84c16 authored 5 Aug 2020 by **Silviu Baranga**

Add support for Morello to clang, llvm, lld, lldb, libcxx,

```
libcxxabi and compiler-rt.
```

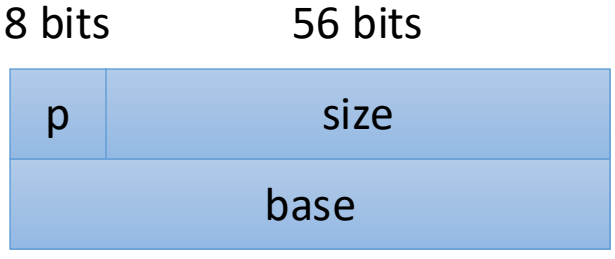
- 1385 files changed, 94515 insertions(+), 5722 deletions(-)
- No public history to chronicle…

Iron Age: Pitting Morello Against CHERI-MIPS

- Repurposes existing GOT sections (including PLT GOTs)
  - Replaces .captable
- Lazy function symbol resolution via PLT stubs
  - Same linkage model as conventional architectures
  - MIPS / CHERI-MIPS had simplistic function call sequence: “load function pointer and perform indirect jump” (ignoring R_MIPS[_CHERI_CAPTAB_INDEX]_CALL* and -mno-abicalls…)
- New ELF relative relocation format for references to local symbols
  - Replaces __cap_relocs (but also supports __cap_relocs… still lives on)
- Support for narrower PCC bounds on function symbols
- Padding added to sections referenced by start/stop symbols

Iron Age: A Relative Improvement

- Fragment format packs base, size, and permissions into target capability, and offset stored in Elf_Rela's addend
- Used by new dynamic relocations R_MORELLO_[I]RELATIVE for references to local symbols
  - Similar to R_*_RELATIVE relocations for non-CHERI
- Also used for initial relocation of jump slots in PLT GOT



Iron Age: Leaps and Bounds

- PCC used to access GOTs, jump to functions and PLT entries, read from constant pools, etc.; bounds must include all these
- Static linker determines which sections can be accessed via PCC
  - Aligns first output section
  - Pads last output section
- Encoded code/function capabilities as offsets relative to PCC bounds in relative relocations
  - That is, base is first PCC output section, not start of function (and size similarly end of last PCC output section, not size of function)
- Does not communicate PCC bounds to runtime loader for use when deriving capabilities for exported functions
  - Infers appropriate bounds as before, which may differ from static linker's view


RISC-V

Industrial Age: Maturation and C18n Prep

- CHERI-RISC-V uses GOT instead of .captable (adapted from Morello LLVM)
  - Includes separate PLT GOTs and lazy function symbol resolution
- Distinguishing Function Pointers from Code Pointers
- Support for IFUNCs in caprelocs
- Bounds on individual TLS variables via TGOT
- PCC bounds on function symbols (adapted from Morello LLVM)

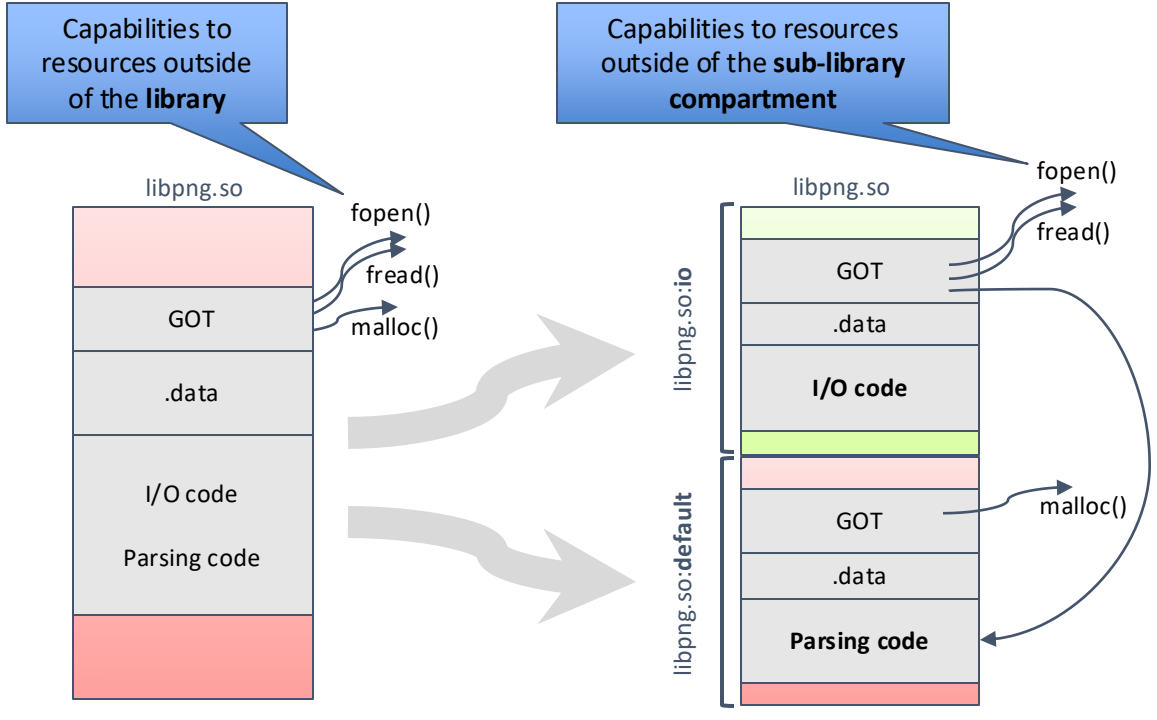
Industrial Age: PCC Bounds

- Tracks PC relative ELF relocations to determine additional sections that should be covered by bounds
  - Handles .data.rel.ro special case in Morello LLVM
- Uses an explicit output section for PCC padding instead of appending zeroes to last covered output section
- Sorts output sections to group all PCC sections together to narrow bounds
- New PT_CHERI_PCC segment (ELF program header) describes address range of PCC bounds

Atomic Age: Sub-Library Compartments

- CheriABI provides isolation between shared libraries
- Ongoing library c18n research strengthens that isolation
- But we want to decompose libraries and/or executables into smaller boxes
- We could split shared libraries into multiple shared libraries…
- … but refactoring source code and build systems is invasive

Atomic Age: Splitting the library



Sub-library compartments are contiguous code, data, and linkage that can access only authorized resources:

- New ELF metadata specifies compartment details including name and **Program-Counter Capability (PCC)** bounds
- External functions and globals are reached via per-compartment **Global Offset Tables (GOTs)**, reached via its PCC

No changes to source code, just a policy file added to LDFLAGS

CHERI

SRI UNIVERSITY OF CAMBRIDGE

Atomic Age: Inter-Compartment Accesses

- Cross-compartment function calls cannot use relative branches due to disjoint PCC bounds
  - Also need GOT indirection to permit runtime interposition (trampolines)
- IPLT stubs created for cross-compartment calls along with GOT entries reusing support for non-preemptible IFUNCs
- Each ELF relocation is verified against an ACL specified in the policy
  - ACL violations result in a link-time error
- ACL can also constrain permissions
  - Read-only capability when accessing a writable symbol

Atomic Age: Compartment ELF Structures

- New string table for c18n-related strings including compartment names
  - .c18nstrtab section
  - DT_C18N_STRTAB and DT_C18N_STRTABSZ
- New PT_C18N_NAME segment describes address range
  - p_vaddr and p_memsz describe virtual address range
  - p_paddr holds offset in .c18nstrtab of compartment name
  - Multiple segments can reference the same compartment name

Future Work

- Implement CHERI relative relocations using a template capability instead of a fragment format
  - Template would be used along with CBuildCap to derive runtime capability
  - Proposed CRelocate instruction would optimize this sequence

- Merge Morello support into CHERI LLVM

Questions?