CHERI+MTE Non-Orthogonal Composition

2024/06

Wes Filardo, Microsoft Azure

CHERI Temporal Safety Implementations

Technology	Architectural Basis	Permits UAF?	Invariants
Never free memory	Nothing special	"No"	All objects created immortal
CheriOS (extended MIPS64)	Capability load instruction rejects single region of VA	Yes; caps to free AS can persist arbitrarily	Impossible to hold pointer into designated VA region
Cornucopia (MIPS64, RV64, & Morello)	Track & trap on cap flow per PTE, on *stores*	Yes; caps to free AS exist until *end* of next epoch	Any cap in need of revocation is on a page flagged "dirty"
Cornucopia Reloaded (RV64 & Morello)	Track & trap on cap flow per PTE, on stores & *loads*	Yes; caps to free AS exist until *start* of next epoch	All valid caps point to objects live as of epoch start
CHERIoT (extended RV32)	Load-capability *instruction* probes quarantine	*No*	*Impossible to hold pointer to free AS*.
Cornucopia Reloaded with MTE	Extant per-PTE track/trap; MTE & new MTE *authority*	*No; MTE recolor prevents access to AS until revoked*	All valid caps of right MTE color point to live objects

2

Let's quickly survey the emerging landscape of technologies and implementations of temporal safety atop CHERI.

1. One approach, the simplest, is to never free memory. It's pretty good: no architectural requirements, it does not permit UAF (or UAR), and it's very easy to explain.

2. The earliest I know of is the implementation in Esswood's CheriOS. This extended the CHERI-MIPS64 processor with system registers and comparators on the capability load path: capabilities pointing into the bounds designated by the registers would have their tag stripped on load. The allocator and operating system worked to detect the single largest region of free VA space and would periodically perform revocation by programming this region into these registers before subjecting all threads to a context switch and every capability in memory to a load and conditional atomic store (of itself). This seeming identity function served to have the side effect of deleting every capability pointing into the programmed region.

   Once revocation began, user programs could not load (and so could not

propagate) pointers into the designated region. On the other hand, capabilities to a particular location of freed address space could persist arbitrarily far into the future, since that location was not guaranteed to be part of a swept region.

3. Cornucopia was born as a data-structure trick using existing security mechanisms in CHERI-MIPS64, specifically its ability to designate pages as trapping on capability stores. Similar, but generally more efficient, mechanisms also exist on CHERI-RISCV-64 and Morello hardware PTWs. Using this architectural foundation, Cornucopia tracks the flow of capabilities to pages as they are stored and maintains the invariant that any capability in potential need of revocation is on a page considered "capability dirty." UAF is possible due to the use of a large, batch quarantine as well as the use of store-based tracking: pointers into quarantine persist until revocation is finished. On the other hand, revocation is global, so all of quarantine is made available at once.

4. Cornucopia Reloaded explores extending the architecture with additional capability flow intercept points, specifically adding the ability to mark a page as triggering traps when a (tagged) capability is *loaded from* it. (The store tracking mechanism of Cornucopia is still present, but it now merely tells us which pages may contain capabilities and does not participate directly in the revocation sweep per se.) This allows us to strengthen the invariant of the system: all valid capabilities observable by userspace point to objects that were live as of the start of the last revocation "epoch," and so userspace cannot propagate capabilities not yet checked for revocation. The use of quarantine, however, still necessitates distinguishing UAF from UAR.

5. Most recently, these ideas have entered CHERIoT, which avails itself of a relatively unique design permitted by being a small single-core system. In this approach, capabilities referencing quarantined address space are invalidated by the capability load instruction itself, which is able to read the (architectural) quarantine state. This closes the UAF/UAR distinction: as soon as free() returns, it is impossible to load a pointer to the freed object. (And because free() is a cross-compartment function call, all registers are spilled and reloaded or explicitly updated as part of the call and return sequence, and so the register file also holds no capabilities to freed memory.)

The purpose of this CHERI+MTE exploration is to get us something with many of the *observable properties* of the CHERIoT approach while being feasibly implemented on a much larger, multi-core machine. Especially, we are after finally closing the UAF/UAR distinction on these larger machines. We expect to be able to do this in a way that increases performance relative to the Cornucopia (Reloaded) approaches –

making it much closer to the CHERI baseline – while also permitting *simplification* of malloc implementations and the use of more clever quarantine strategies.

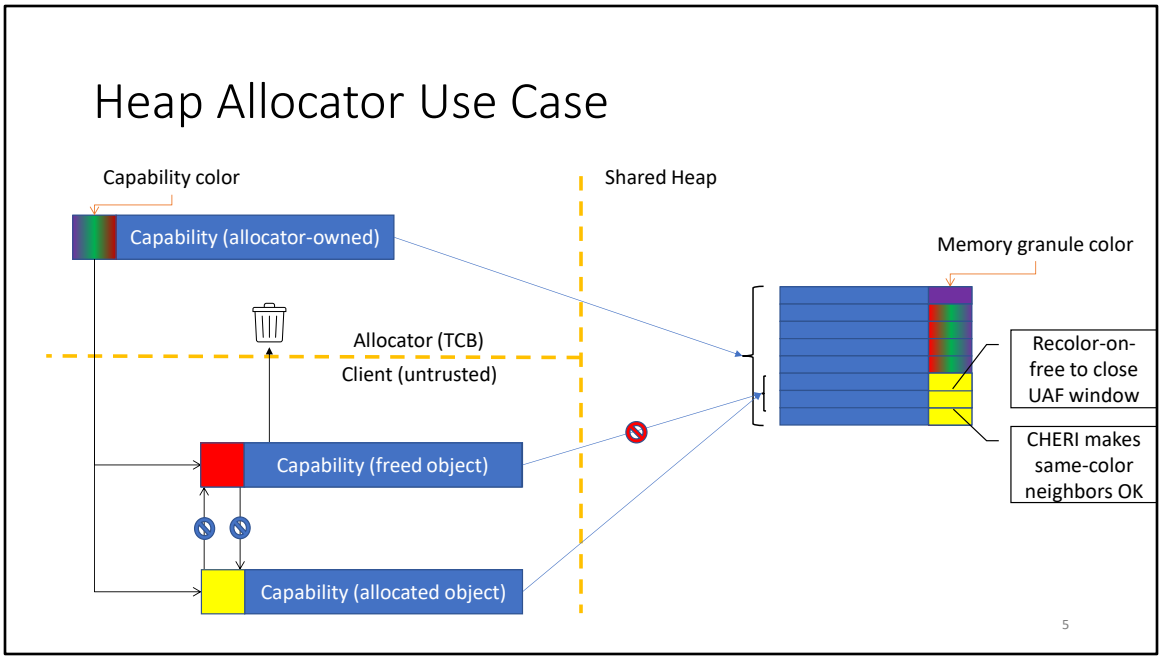
CHERI+MTE In One Slide

- MSR is pursuing a *non-orthogonal* composition of CHERI and MTE (not just CHERI w/ TBI addresses for MTE)
- "Lock and key" memory story, now with ability to *destroy the key* (CHERI) or *change the lock* (MTE)
  - MTE colors still physical (one per physical address granule, no additional indirection visible to EL0/1/2)
  - Extends CHERI provenance and monotonic authority model onto MTE colors themselves
  - Dedicated instructions for manipulating MTE colors in RAM and caps
- Promising consequences to both uarch and software:
  - Relieves MTE of spatial concerns; simplifies CHERI's temporal concerns
  - MTE color values can now be fully public without loss of security
  - New light-weight semi-synchronous trapping mode justified by C language semantics

3

Caveats

- Proposal here emerged from *heap* temporal safety work; other use cases much less explored
  - Assumes Cornucopia-style revocation available
- Unclear exactly where is the right place to steal bits in a CHERI cap for MTE bits.
  - Unclear implications for capability encoding and capability getter / setter instructions
  - Tracking discussion for CGetAddr vs CToPtr vs as integer alias vs CGetLow (github.com)



MTE augments each pointer and memory granule with a "color". In CHERI, color bits are part of the capability metadata, and so protected against corruption.

1) On allocation, allocator derives bounded, colored capability to heap memory and grants this to the client.
   The distinguished "rainbow" color value is allowed to derive capabilities of any color and change the color of memory.
   Other colors can only produce the same color progeny and cannot change memory's color.

2) The client is then free to use the derived capability, and eventually frees it.

3) The allocator uses its elevated authority to recolor memory, preventing the client's *valid capability* from reaching memory. (Can zero memory itself with little/no additional cost at the same time.)

4) Recolor-on-free closes UAF window, which gives a better debugging story, and enables secure *in-band metadata*, simplifying allocator design. (More on that in a moment.)

5) CHERI handles the spatial safety concerns, so adjacent heap objects can have the same color without loss of security.

6) Re-allocation proceeds as last time, with the allocator constructing a new capability of the right color for the client. (Any in-band metadata cleared before return.)

7) Clients cannot change the color of their capabilities, nor can they recolor memory.

- Mismatching loads trap; data dependence may delay retirement of subsequent independent instructions, but no other costs.
  Mismatching stores can *fizzle*: can retire immediately and will be dropped from the store buffer rather than updating in L1. Some complexity around store-to-load forwarding (wait, don't trap, if colors mismatch?), but should hide latency of fetch for color comparison.

- A purpose-built atomic compare-and-decrement-color instruction catches would-be double-frees and handles concurrent interaction with the revocation.

- Even 1-bit "scaled down" MTE has useful security properties (closes UAF window) and simplifies software design (allows in-band metadata) but loses performance win of delayed revocation

Extending CHERI *Authority* to MTE

CHERI capabilities represent the *authority* to perform some action. MTE impacts:

1. Constructing capabilities must obey a new "colors stick" rule:
   - Monochromatic caps have only *equal-color* descendants
   - Any progeny of rainbow caps may be either rainbow or monochromatic
2. Load/store of *colors* from/to memory requires a *rainbow* authority capability
3. Load/store of *data* must also satisfy either:
   - Rainbow authorizing cap
   - Equal colors in authorizing cap and memory

6

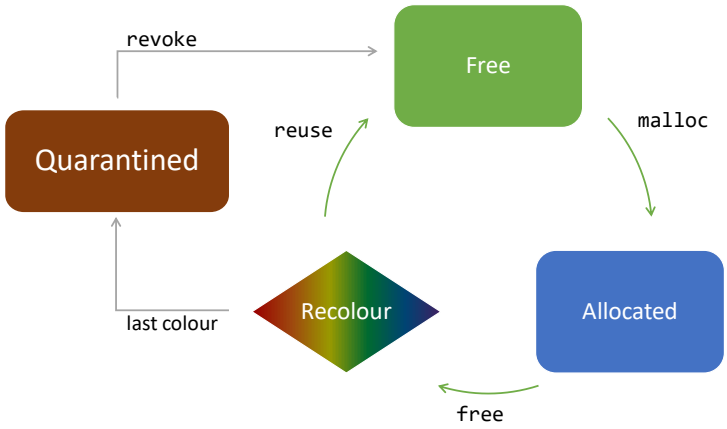
CHERI capabilities represent authority to perform actions. This hybridization of CHERI and MTE has introduced new things upon which actions may be performed, and so must extend CHERI's notion of authority to these things.

1. When building a new capability from another, we have to say how the progeny's color field can be changed relative to its parent. We introduce a "colors stick" rule, which says that any progeny of a monochromatic capability must *inherit* their color from their parent, while rainbow capabilities may give rise to either rainbow or monochromatic capabilities.

2. When attempting to access the MTE colors for a granule of memory, the authority capability must be a rainbow capability, in addition to satisfying all the other CHERI and system requirements for dereferencing that granule.

3. When attempting an ordinary data dereference, the authorizing capability must either be a rainbow capability or have color matching the granule's current color, with the color comparison being interpreted the same way as value comparison in atomic CAS.

Colouring & Revocation



7

Putting everything together, CHERI and memory colouring let us give out spatially-bounded pointers to heap objects at particular colours.

1. When those objects are freed, we can recolour their backing memory, invalidating pointers to the freed object.

2. If we have not exhausted the colour space, memory can be queued for reuse immediately, including in-band metadata (more in a slide)!

3. When we have exhausted the colour space for a given piece of memory, it is instead unmapped, if possible, and the address space held in quarantine.

4. Only when we are close to exhausting address space or when mapped memory is sufficiently fragmented must we run the revoker, which then makes address space safe for reuse.

Therefore, address space enters quarantine at a significantly reduced rate: 1/(ncolor-1) of the rate without MTE. A 4-bit MTE scheme lets us accumulate quarantine at a 15^{th} the pace, and so with the same quarantine size, we need to revoke a 15^{th} as

often, or we could have a 15th the quarantine and revoke at the current pace, or some other tradeoff.

CHERI+MTE Allocator Optimization

- Unforgeability of colors in capabilities => security can depend on MTE
- Allocators like to use free space for free lists
  - Requires active defenses: OOB metadata or obfuscation of in-band metadata
  - Recolor-on-free closing UAF window => removes need for defenses!

8

1) Now that MTE colors are part of the protected CHERI capability metadata, we can rely on them for system security.

2) Using free space for free lists makes a great deal of sense; however, by turning free into a tacit reallocation, it opens the door to UAF/UAR attacks on the allocator itself, frequently giving rise to very powerful “write what where” primitives.
   Thus, “hardening” allocators often means moving metadata out of band entirely or engaging in some kind of secret-based, checkable obfuscation of in-band metadata, all of which requires cycles and/or risks the existence of a disclosure vulnerability or suitable gadget.

3) But recoloring on free means that our implicit reallocation is just as protected from tampering as an actual allocation would be, so we are free to go back to using un-obfuscated, in-band metadata, so long as we remove it before exposing the memory to the client again.
   There is, of course, cost to recoloring, but it is probably equivalent to the cost of zeroing, another thing we should be doing in hardened allocators.

These security benefits of CHERI+MTE exist for any positive number of bits; additional bits just slow the rate at which address space enters quarantine. If convenient for uarch, could imagine CHERI+MTE as 1+3 bits per physical granule, for example.

Relaxed MTE Behavior

- C object model: Assume every pointer from malloc() is to distinct memory.
- Heap reuse model: MTE mismatch *permanent* not ephemeral; mismatches *fatal*.
- Proposed MTE behavior: "*mismatched loads **trap**, mismatched stores **fizzle**"*
  - Loads act like poisoned ECC (& limited imprecision?)
  - Stores to mis-colored memory are *silently discarded*
    - CPU pipeline able to fire-and-forget stores after translation: no post-translation store fault.
    - Some subtlety for store-load forwarding
    - A little more state in caches to hide latency

9

1) The C object model lets us get away with revocation because not only is the use of a free pointer UB, the *value of* a freed pointer is also undefined. Every return from malloc() can be, and is, magically, assumed to not alias with any other return.

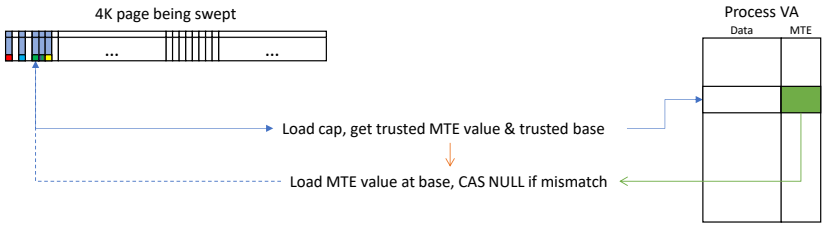
2) In order to enforce that lack of aliasing, Cornucopia imposed quarantine on free pointers until revocation. Here, we're using MTE colors to defer revocation, effectively quarantining (virtual address, color) pairs and only quarantining the VA once the colors are exhausted. That is, MTE mismatch is *permanent* until revocation, and mismatching dereferences are always fatal. Contrast translation faults, which are caught and routinely suppressed by system software.

3) Putting all that together, we propose the a "loads trap, stores fizzle" mode for MTE:
   - Loads, which wait for the memory subsystem anyway, now might trap later, but this is already available in some uarch-es apparently without a huge amount of pain, in the guise of poisoned ECC results.
   - Since software cannot load through the pointer obtained from malloc to see the consequence of any store, we let mismatching stores proceed and

fizzle in the memory hierarchy.

- Store buffer or caches hide fetch-and-compare latency
- Store buffer must not unconditionally forward stored data or write to L1$ until colors are confirmed to match memory (unless forwarded-to instructions also check colors or unless auth cap was rainbow, in which case the stores can be forwarded and write-allocate as usual).

Revoker Page Sweep w/ MTE



- Intended software policy
  - *any* capability with mismatching, non-rainbow MTE color might get its tag cleared at any moment.
  - non-rainbow caps to unmapped addresses (*no* associated MTE color) also revoked.
- All (heap) temporal safety state is now *architectural*!

10

Rather than needing a bitmap to articulate quarantine (as in Cornucopia and Cornucopia Reloaded), we can use the MTE bits themselves.

Revocation now eliminates all capabilities whose color mismatches the memory color at their *base*. We rely on the non-TCB's inability to change memory colors and monotonicity of the base (if the capability has non-zero range, then the base is somewhere within the original allocation). If the page being pointed to has been unmapped, the load either will soft fault in a zero page of some color, if the page is released but the AS reservation remains, or will hard fault, if the reservation has been torn down; in the latter case, capabilities will be revoked regardless of color.

Slightly wasteful with the "always quarantine after N colors" policy: concurrent revocation may have cleared more colors that could be used. However, doing better seems like it would be expensive, requiring snapshots of color state as of the start of the most recent revocation? If that could be done, address space could enter quarantine at an even slower pace, as each revocation sweep could buy back some colors even in non-quarantined regions.

Proposed Instructions

Minimal core:

Mnemonic	Synopsis
`CGetMTE rd, cs`	Copy color of cap in **cs** into **rd**
`CSetMTE cd, cs, rs`	Derive a **rs**-colored cap into cd from rainbow cap in **cs**
`CLoadMTE rd, cs`	Load color from ***cs** to **rd** Requirements: **cs** tagged, unsealed, load-bearing, MTE-aligned, rainbow
`CStoreMTEAndZero rs, cs`	Recolor ***cs** to **rs** and zero ***cs** data Requirements: **cs** tagged, unsealed, store-bearing, MTE-aligned, rainbow
`CAMOCDecMTE rd, cs1, cs2`	Atomic Conditional Decrement color of ***cs1** (details on next slide)

Extras:

Mnemonic	Synopsis	Uses
`CStoreMTE rs, cs`	Recolor ***cs** to **rs** w/o altering data	Inter-compartment move elision
`CLoadMTEAsUser rd, cs`	EL1/2 accessing EL0 ignoring PAN	Revoker probe of colors

11

There is no “random tag except for these” selection instruction or such, because we rely on CHERI spatial safety to isolate adjacent objects and on revocation to determinize the use of colors.

For CStoreMTE, CStoreMTEAndZero, and any other “store MTE color” opcodes we want, David has made a cute observation:

- Since (MTE granule size in bytes) >= (MTE colors available), we can store MTE colors in the low bits of addresses of authorizing capabilities.
- That is, “CStoreMTE rs, cs” could be encoded as “CIncOffset cs, cs, rs; CStoreMTE cs” (assuming sufficient alignment of cs.address at the start.

Atomic Conditional Decrement??

- `free(p)` needs to guard against double free, concurrent free, and concurrent revocation
- Allocator maps **p** to rainbow capability **r** to a heap object by *rederiving from heap root(s)*.
  - **r** may have different address from **p**, if free accepts interior pointers; **r** points to first word of allocation.
- Atomically, test that **p** is tagged (& permissive) and has the same color as memory ***r**…
  - If **p** is a UAF from a free in a previous revocation epoch, it will be untagged.
  - If **p** is a UAF and its memory is not yet de-quarantined, the colors will not match.
- … and, if so, recolor granule at ***r** to lower color.
  - Rules out subsequent and concurrent attempts to free **p** (or any other address within this object).
  - Allocator will next recolor and zero the rest of the object (using non-atomic `CStoreMTEAndZero`).

12

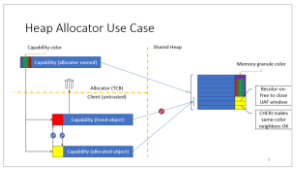
1) Added challenge for snmalloc: without taking locks.

2) Presently we map offset-0 and interior pointers **p** for a given object all to the same **r**, so that we don't have to worry about accidental recoloring of intermediate words.

3) The instruction performs "just enough" testing on **p** to know that it hasn't been revoked; in particular, the bounds of **p** are not checked, only those of **r**.

4) Because we're using every color once, we don't need to specify the target color as an input operand; instead, we just let the instruction decrement the color.
   We use 0 as the concrete rainbow value, too, so decrement has a very natural feel to it: colors count down until quarantine.

Open Questions



- Is this generalizable outside the heap?
- Overheads orthogonal between CHERI and MTE? How cheap is both?
  - Is “loads traps, stores fizzle” as viable as it seems?
- PTE bit? “Store MTE” permission?
  - Expedient for prototyping, but maybe that’s all it’s for.
- Capability encoding: 60-bit address field?
- …?

13