Temporal Safety and Capability Revocation

“Temporal safety” is a broad term. Two major facets:

- “use after free”: continued loads and stores to region of memory declared dead.
- “use after reallocation”: reference to former object at some location used to access a different object at the same place.

Of the two, “after reallocation” substantially worse.
(Anyone want to claim to have never written either of these?)

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 contract FA8750-10-C-0237. 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.

Temporal Safety and Capability Revocation

Use after reallocation avoidable if allocator…

- *waits* for references to go away
    - Memory in *quarantine*.
    - Like garbage collector.
    - Possibly large space overheads while waiting
- *revokes* references
    - Indirection? Lookasides?

Temporal Safety and Capability Revocation

Thinking about CHERI, we...

- can find all our capabilities by their tags.
  - code can't play tricks on us
  - *base* of capability always within initial object allocation
- do not have indirecting capabilities.

Temporal Safety and Capability Revocation

Thinking about CHERI, we…

- can find all our capabilities by their tags.
  - code can't play tricks on us
  - *base* of capability always within initial object allocation
- do not have indirecting capabilities.

So, could build a less conservative GC atop CHERI:

- Precisely identify referenced objects
- Still conservatively assume that all references might be used

Temporal Safety and Capability Revocation

- But we are impatient and don't want to wait unboundedly long for some stale reference to go away.
- Because we can find pointers, we can also *clear* them! "Sweeping revocation" (contrast "indirecting")

Temporal Safety and Capability Revocation

- But we are impatient and don't want to wait unboundedly long for some stale reference to go away.
- Because we can find pointers, we can also *clear* them!
  "Sweeping revocation" (contrast "indirecting")
- Need a privileged bit of software…
  …can see all memory and registers.
  For us, that means the (CheriBSD) kernel.

Temporal Safety and Capability Revocation

- `mmap` "returns two things":
  - Capability to the pages you asked for
  - A mutable bitmask for expressing revocation requests.
    1 bit = 16 bytes of memory in returned pages
- Before reusing memory
  - Set bits corresponding to object
  - Call kernel to do sweep
  - Clear bits corresponding to object
  - (Clear the object itself, too?)

Temporal Safety and Capability Revocation

- All of memory every reuse???
  - No! Batch requests!
  - Bitmap allows for arbitrarily many revocations in one go.

Temporal Safety and Capability Revocation

- All of memory every reuse???
  - No! Batch requests!
  - Bitmap allows for arbitrarily many revocations in one go.
- All of memory???
  - Well, just the capability-bearing bits of it.
  - Page table dirty bits can guide us.
  - Added instruction for reading tags without reading data.
  - A very predictable stride, too.

Temporal Safety and Capability Revocation

- All of memory every reuse???
  - No! Batch requests!
  - Bitmap allows for arbitrarily many revocations in one go.
- All of memory???
  - Well, just the capability-bearing bits of it.
  - Page table dirty bits can guide us.
  - Added instruction for reading tags without reading data.
  - A very predictable stride, too.
- Simulation: 5% runtime overhead if heap 25% bigger.

Temporal Safety and Capability Revocation

- All of memory every reuse???
  - No! Batch requests!
  - Bitmap allows for arbitrarily many revocations in one go.
- All of memory???
  - Well, just the capability-bearing bits of it.
  - Page table dirty bits can guide us.
  - Added instruction for reading tags without reading data.
  - A very predictable stride, too.
- Simulation: 5% runtime overhead if heap 25% bigger.
- Pause times?
  - Revoke concurrently with application!
  - Take guidance from concurrent garbage collectors. (Card marking, trap-and-mark)