# CheriBSD Deep Dive

Cheri Alliance Linux WG meeting
29 Jan 2025

Alfredo Mazzinghi

CHERI

CAPABILITIES LIMITED

UNIVERSITY OF CAMBRIDGE

# Outline

- Approaching an OS kernel for CHERI adaptation
  - CHERI C programming model
  - Compile-time vs Run-time issues
- CHERI OS design space
  - What are pointers?
  - What are allocators?
  - Where pointers come from?
  - Capability precision
  - Intra-allocation bounds
  - Capability revocation
  - Compartmentalisation
- Evaluation metrics

# Approaching an OS kernel for CHERI adaptation

I have an OS kernel fork, what do I do now?

- Pragmatic approach
  - I started with a working hybrid kernel with a purecap userland
  - Uncertainty about problematic C language idioms
  - Uncertainty about run-time capability manipulation
- Broadly split the effort in three phases
  - Handle static warnings flagged by compiler (e.g. `-Werror`)
  - Drive run-time changes from boot up to login shell
  - Refine critical kernel subsystems

# Incompatible C language idioms – 1

- Now generally well-known
  - Invalid use of integer types for pointers (and viceversa)
  - Insufficient alignment
  - Invalid or undecidable provenance
  - Incompatible arithmetic
  - Monotonicity violations (e.g. `containerof`, allocators)
- Manifest as a mix of compile-time and run-time errors

# Incompatible C language idioms – 2

- CHERI compiler warnings are useful
  - Sometimes quite verbose (e.g. `-Wcheri-inefficient`)
  - Should catch most incompatible issues
    - Sub-object bounds are a known gap
- Static analysis tools are now available

# Incompatible C language idioms – 3

In many cases these are simple fixes

```
@@ -246,7 +246,7 @@ bintime_off(struct bintime *bt, u_int off)
        do {
                th = timehands;
                gen = atomic_load_acq_int(&th->th_generation);
-               btp = (struct bintime *)((vm_offset_t)th + off);
+               btp = (struct bintime *)((char *)th + off);
                *bt = *btp;
                scale = th->th_scale;
                delta = tc_delta(th);
```

# Incompatible C language idioms – 4

Explicit provenance for uintptr arithmetic can also be disruptive.

```
@@ -1104,9 +1104,10 @@ __rw_wlock_hard(volatile uintptr_t *c, uintptr_t v
LOCK_FILE_LINE_ARG_DEF)
                 * ownership and maintain the pending queue.
                 */
                setv = v & (RW_LOCK_WAITERS | RW_LOCK_WRITE_SPINNER);
-               if ((v & ~setv) == RW_UNLOCKED) {
+               if ((v & (ptraddr_t)~setv) == RW_UNLOCKED) {
                        setv &= ~RW_LOCK_WRITE_SPINNER;
-                       if (atomic_fcmpset_acq_ptr(&rw->rw_lock, &v, tid | setv)) {
+                       if (atomic_fcmpset_acq_ptr(&rw->rw_lock, &v,
+                           tid | (ptraddr_t)setv)) {
                                if (setv)
                                        turnstile_claim(ts);
                                else
```

# Incompatible C language idioms – 5

But some hint at deeper design decisions to be made

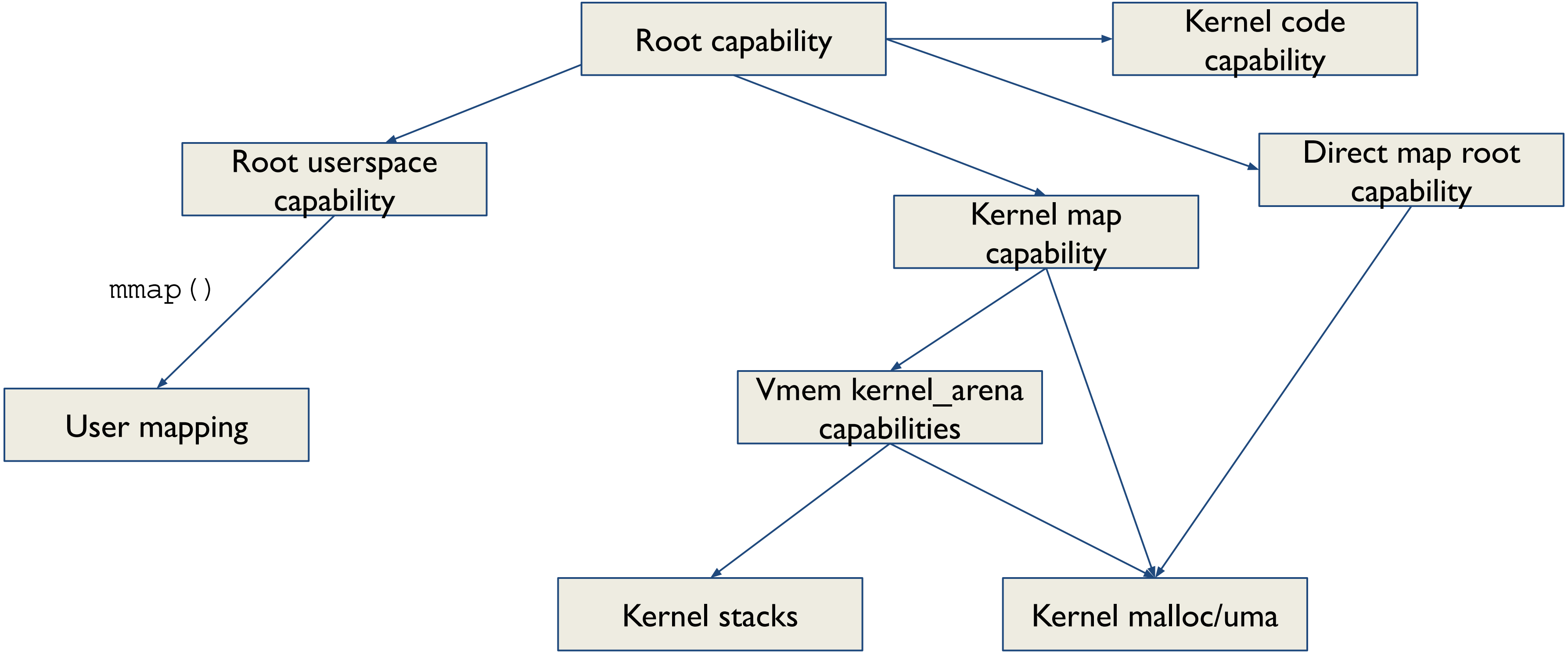
```
@@ -82,7 +82,7 @@ static void
 sf_buf_init(void *arg)
 {
         struct sf_buf *sf_bufs;
-        vm_offset_t sf_base;
+        vm_pointer_t sf_base;
         int i;
         // [...] elided code
         sf_base = kva_alloc(nsfbufs * PAGE_SIZE);
```

# CHERI OS design space

Some initial questions

- Where do pointers come from?
  - Traditionally not a problem, can materialise a pointer anywhere as long as there is mapped memory at the desired address.
  - Need a provenance tree from boot capabilities
- What is a pointer and what is an address?
  - Traditionally interchangeable, see FreeBSD `vm_offset_t`
  - There is a clear and enforced distinction now
  - At what point in the virtual memory system an address becomes a capability? (see `kva_alloc` example before)

# Provenance Tree – Booting CheriBSD



# CHERI OS design space – Allocators

Distinction between pointers and addresses has ramifications in the system

- Allocators must guarantee CHERI invariants
  - What is an allocator? (e.g. `PHYS_TO_DMAP`)
  - What properties must it guarantee?
    - Pointers are dereferenceable
    - Capabilities do not alias
    - Do we have expectations about uninitialized memory?
  - What about free?
- Are there missing abstractions?
  - Reservations introduced to CheriBSD VM map interface

# CHERI OS design space – Intra-allocation bounds

Narrowing bounds for logically nested objects is important

- Not limited to compiler-assisted sub-object bounds
  - Sub-allocation
  - Explicit setbounds operations (e.g. isolate packet headers)
- Trade-off: require more manual intervention
  - Sub-object bounds break with `containerof` and C inheritance.
  - Sub-allocation require auditing

# CHERI OS design space – Precision

Capability precision plays a role

- Not limited to reservations
  - Affects explicit bounds narrowing decisions
  - Affects security properties of intra-allocation bounds
  - Platform-dependent property
- Aim to enforce exact bounds by default
  - There should be a clear reason for having best-effort bounds
  - Can be done incrementally
  - Flex array members are tricky
- Pointer arithmetic loses associativity!

```
pcpu_base + (symbol - pcpu_start) ≠ (pcpu_base - pcpu_start) + symbol
```

# CHERI OS Design Space – Temporal safety

- Kernel side under investigation
  - Trade-off between allocation reuse and revocation frequency
    - UMA design maximises reuse and locality
  - Do we really need to revoke UMA allocations?
    - Is the problem temporal aliasing with a different C type?
  - Multiple allocator layers need coordination
  - Direct map does not have a free()

# CHERI OS Design Space – Compartmentalisation

- Kernel compartmentalisation under development
  - Current model extends user library compartmentalisation techniques for kernel modules.
  - Complexity due to privileged kernel interfaces
  - Discussion and design is ongoing

# Evaluation Metrics

- Research focus should expand with Linux support
  - Main goal has been to
    - Demonstrate feasibility
    - Compare hybrid and pure-capability ABIs for research
    - Minimise disruption to code base
- Linux historically introduces more kernel hardening techniques
  - Are our metrics for code disruption the same?
  - Compare with PAC, CFI, KALSR, MTE, allocator hardening, etc.