Address space reservations

Re-thinking address space management for pointer provenance

Brooks Davis brooks.davis@sri.com
BSDCan 2024, Ottawa, Canada
"Implementations are permitted to track the origins of a bit-pattern and treat those representing an indeterminate value as distinct from those representing a determined value. They may also treat pointers based on different origins as distinct even though they are bitwise identical."

–WG14 (C standard committed) DR260 Committee Response
“Just because two pointers point to the same address, does not mean they are equal and can be used interchangeably.”

– Pointers Are Complicated, or: What's in a Byte? Ralf J

What has pointer provenance?

• C: DR 260 (elaborated in N2577)
• C++: from C (standard unclear)
• Rust: RFC 3559-rust-has-provenance
• CHERI capabilities
• **Capabilities** extend integer memory addresses

• **Metadata** (bounds, permissions, ...) control how they may be used

• **Guarded manipulation** controls how capabilities may be manipulated; e.g., *provenance validity* and *monotonicity*

• **Tags** protect capability integrity/derivation in registers + memory
CHERI enforces protection semantics for pointers

- **Integrity** and **provenance validity** ensure that valid pointers are derived from other valid pointers via valid transformations; **invalid pointers cannot be used**

- **Bounds** prevent pointers from being manipulated to access the wrong object

- **Monotonicity** prevents pointer privilege escalation – e.g., broadening bounds

- **Permissions** limit unintended use of pointers; e.g., W^X for pointers

- These primitives not only allow us to implement **strong spatial and temporal memory protection**, but also higher-level policies such as **scalable software compartmentalization**
CheriABI – pointer provenance & least privilege

- New process ABI where all pointers are CHERI capabilities
  - Implemented in CheriBSD, our FreeBSD fork
- Kernel provides bounded pointers for all mappings
  - Includes initial executable, stack, etc as well as mmap
- System calls do not violate bounds (no kernel escape hatch)
  - Necessary for compartmentalization
- mmap can only manipulate backing of a new mapping or via an existing capability
  - This decouples address space reservation and backing store configuration
  - Software capability permission (SW_VMEM) required to change/unmap
Hilbert curve

https://xkcd.com/195
CC BY-NC 2.5 DEED
Processes memory layout (not to scale, no ASLR)

- **NULL (no mapping)**
- **.text**
- **.data**
- **.bss**
- **End of “heap”**
- **jemalloc**
- **break**
- **stack**
- **GUARD**
- **vdso**
- **v**
mmap overview

void *mmap(
    void *addr,  /* address to map at aka hint */
    size_t len,  /* size */
    int prot,    /* page protections */
    int flags,   /* how to map */
    int fd,      /* file descriptor (often -1) */
    off_t offset /* offset in file (often 0) */
);
mmap does many things

- Space for malloc
  - addr=0, flags=MAP_ANON
- Shared file mapping
  - addr=0, flags MAP_SHARED
- Shared library
  - addr=0, flags=MAP_GUARD
  - addr=base, flags=MAP_FIXED
  - addr=base+ts, flags=MAP_FIXED
  - addr=base+ds, flags=MAP_ANON|MAP_FIXED
Maybe too many things?

- Extending an allocation
  - `addr=0, flags=MAP_ANON`
  - `addr=base+len, flags=MAP_ANON`
  - if `new` is `base+len`, treat as one
  - jemalloc does this

- Absurd things
  - Shingled mappings
  - Random fixed mappings
The problems with mmap

• mmap conflates two things:
  • Address space allocation
  • Configuration of backing store and permissions

• All mmap callers can do anything
  • MAP_GUARD and MAP_EXCL prevent some errors
  • ...but every call is with ambient authority

• Lack of cross-platform agreement beyond POSIX
Updating mmap for CheriABI
Normal mmap use is unchanged

- Space for malloc
  - `addr=0, flags=MAP_ANON`
- Shared file mapping
  - `addr=0, flags=MAP_SHARED`
- Shared library
  - `addr=0, flags=MAP_GUARD`
  - `addr=base, flags=MAP_FIXED`
  - `addr=base+ts, flags=MAP_FIXED`
  - `addr=base+ds, flags=MAP_ANON|MAP_FIXED`
What about weird mmap use?

• Extending an allocation
  • addr=0, flags=MAP_ANON
  • addr=base+len, flags=MAP_ANON
    - if new is base+len, treat as one

• Absurd things
  • Shingled mappings
  • Random fixed mappings
What about compressed bounds?

- Bounds are compressed relative to the address
- Different alignments and lengths require different numbers of bits to represent
- Consider these four ways to represent 14 pages
- With CHERI, some lengths can’t be represented and thus must be padded
- Note: CHERI has much broader representability than illustrated
Unmapping memory

• Like mmap, munmap can unmap anywhere!
• Even with CheriABI restrictions, you can punch holes in a mapping
• Now we have aliasing between pointers to active mappings
Reservations to the rescue!

- Initial mapping creates a reservation (and populates)
  - Can not be merged with other reservations

- Unmapping pages creates UNMAPPED VM entries
  - Can not be mapped over

- Padding starts UNMAPPED

- Reservation is removed when all entries are unmapped
Temporal safety issues: use-after-munmap

- Consider the sequence:
  - mmap a file
  - ...
  - munmap the file mapping
  - cause malloc to mmap more space
  - <bug> access pointer to file mapping which aliases with malloc’d memory

- Usual answer: don’t do that
- CHERI answer: capability revocation

Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety

Nathaniel Wesley Filardo  
Microsoft  
Canada

Jessica Clarke  
University of Cambridge  
UK

Mark Johnston  
University of Cambridge  
UK

Simon W. Moore  
University of Cambridge  
UK

Brett F. Gutstein  
University of Cambridge  
UK

Peter Rugg  
University of Cambridge  
UK

Robert Norton  
Microsoft  
UK

Peter G. Neumann  
SRI International  
USA

Jonathan Woodruff  
University of Cambridge  
UK

Brooks Davis  
SRI International  
USA

David Chisnall  
SCI Semiconductor  
UK

Robert N. M. Watson  
University of Cambridge  
UK

Abstract
Violations of temporal memory safety ("use after free", "UAF") continue to pose a significant threat to software security. The CHERI capability architecture has shown promise as a technology for C and C++ language reference integrity and spatial memory safety. Building atop CHERI, prior works – CHERI revoke and Cornucopia – have explored adding heap temporal safety. The most pressing limitation of Cornucopia was its impractical "stop-the-world" pause times.

We present Cornucopia Reloaded, a re-designed drop-in replacement implementation of CHERI temporal safety, using a novel architectural feature – a per-page capability load barrier, added in Arm’s Morello prototype CPU and CHERI-RISC-V – to nearly eliminate application overheads across SPEC CPU2006 and nucopia and with median 87% of Cornucopia’s DRAM traffic.
How it works

- Reservation is unmapped over time
- Fully UNMAPPED reservation becomes a QUARANTINE entry
- When an adjacent reservation becomes a QUARANTINE entry they can be merged
- Revocation pass invalidates all capabilities to largest QUARANTINE entry
- QUARANTINE entry is removed
An odd side effect

• Revocation is batched → unmapped address space isn’t immediately available

• Stale capabilities will become invalid at some arbitrary point after unmap

• We want mmap to behave consistently:
  • If addr is a valid capability, it must correspond to an active reservation and MAP_FIXED must be set in flags
  • If addr is NULL-derived it must not correspond to any reservation
  • By implication: addr must not have metadata and be invalid
    • We don’t want it to work IFF enough time has passed for revocation
    • Too confusing
Unmapping bug in the wild

/* From autoconf prior to 2.72 (heavily trimmed) */
data2 = mmap (0, pagesize, PROT_READ | PROT_WRITE, MAP_SHARED, fd2, 0L);

munmap (data2, pagesize);

/* Next, try to mmap the file at a fixed address
 * which already has something else allocated at
 * it. [...] */
data2 != mmap (data2, pagesize,
            PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_FIXED,
            fd, 0L);

Race!
One other change: PROT_MAX()

- \texttt{mmap} returns capabilities
- The property of monotonicity means the capability returned at reservation create must have all required permissions
  - ...but reserving space with PROT\_NONE is a common pattern
- Convert to: PROT\_MAX\((\text{PROT\_READ}|...)|\text{PROT\_NONE}\)
  - Capability gets read/write permissions
  - Page access restricted until later updates
  - Maximum page permission limited by PROT\_MAX()
  - (Arm Morello Linux returns RWX caps instead.)
• Initial allocation reserves fixed amount of address space
• No growth without relocation
• No immediate reuse of unmapped address space
• This reuse is unsafe in general
• (Test programs can use libprocstat to find empty address space)
• Use PROT_MAX() to control capability permission

Improving Memory Permissions in FreeBSD
by Brooks Davis

The virtual address space of a process contains a number of physical pages mapped into memory. These might be pages from a program, a library, an ordinary file, or anonymous pages that begin life as a zerod page. These mappings are maintained in a translation lookaside buffer (TLB). On modern architectures, the TLB allows pages to be mapped with a combination of read, write, and execute permissions. This enables things like read-only sharing of code and data between processes for physical memory utilization.

Older architectures (e.g., MIPS, early i386) only supported read and write permission, but modern CPUs generally support an execute permission as well. Used correctly, the execute permission can mitigate a number of common security vulnerabilities. For example, it used to be common to exploit a program by writing code (commonly known as shell code) to an input buffer and then changing the saved return address of the function to point to this code. By removing the execute permission from the stack, we can prevent this attack. Most FreeBSD architectures do this.

As expected, breaking simple, stack-based attacks leads attackers to look for other vulnerabilities. One of the simplest next steps was to find a way to write code to a page that was mapped executable. A popular mitigation for this is the write-XOR-execute-policy (W^X). This policy prevents mapping pages with both the write and execute permission. For most programs, this works without program changes outside the runtime linker, but some programs such as Java virtual machines and web browsers use just-in-time (JIT) compilers to generate code and run it. These JTs are critical to achieving reasonable performance, but, implemented naively, they don’t work with W^X. Fortunately, it is usually a simple matter to map pages writable, writable generated
Are these incompatibilities worth it?
Hyrum's Law

With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody.

https://www.hyrumslaw.com
CHANGES IN VERSION 10.17:
The CPU no longer overheats when you hold down spacebar.

COMMENTS:

LONGTIME_USER4 writes:
this update broke my workflow! My control key is hard to reach, so I hold spacebar instead, and I configured Emacs to interpret a rapid temperature rise as "control".

ADMIN writes:
that's horrifying.

LONGTIME_USER4 writes:
look, my setup works for me. just add an option to reenable spacebar heating.

Every change breaks someone's workflow.
Arguments in favor

• Pointer provenance respected
  • Systems languages (C, C++, Rust) expect this
• Races eliminated
• Changes are portable
  • (As much as mmap is...)
• 50-100 MLoC memory safe C/C++ code show viability!
• CheriABI enables fine grained compartmentalization

Memory safe desktop with library compartmentalization! (Chrome and OpenJDK currently excluded)
Implementation notes

- Reservations are enabled on a per-vm_map basis
- Reservations are identified by the lowest VA in the reservation
  - Member added to struct vm_map_entry
  - Entries from different reservations can’t be merged
- UNMAPPED entries have the MAP_ENTRY_UNMAPPED flag set
  - Mostly the same as MAP_GUARD, but can’t be mapped over
    - Morello Linux allows mremap to extend over UNMAPPED pages
- QUARANTINE state is indicated by an inheritance of VM_INHERIT_QUARANTINE
  - Chosen because special handling is required in vm_space_fork()
  - Adjacent entries can be merged (gaps filled during revocation)
Open questions

• How should mremap interact with reservations?
  • First documented along with mmap, but never implemented in FreeBSD
  • Extending reservations is theoretically possible, but fraught

• Do address space reservations make sense for non-CHERI ABIs?
  • Would need opt-out for old code
  • Does increased vm map entry count have a measurable impact in practice?

• When should we merge CHERI support to FreeBSD?
  • CHERI-RISC-V standardization in progress
  • Silicon in 2025?
Questions?

https://www.cheribsd.org