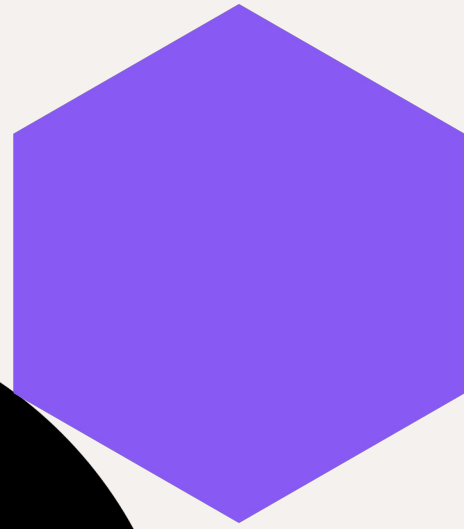
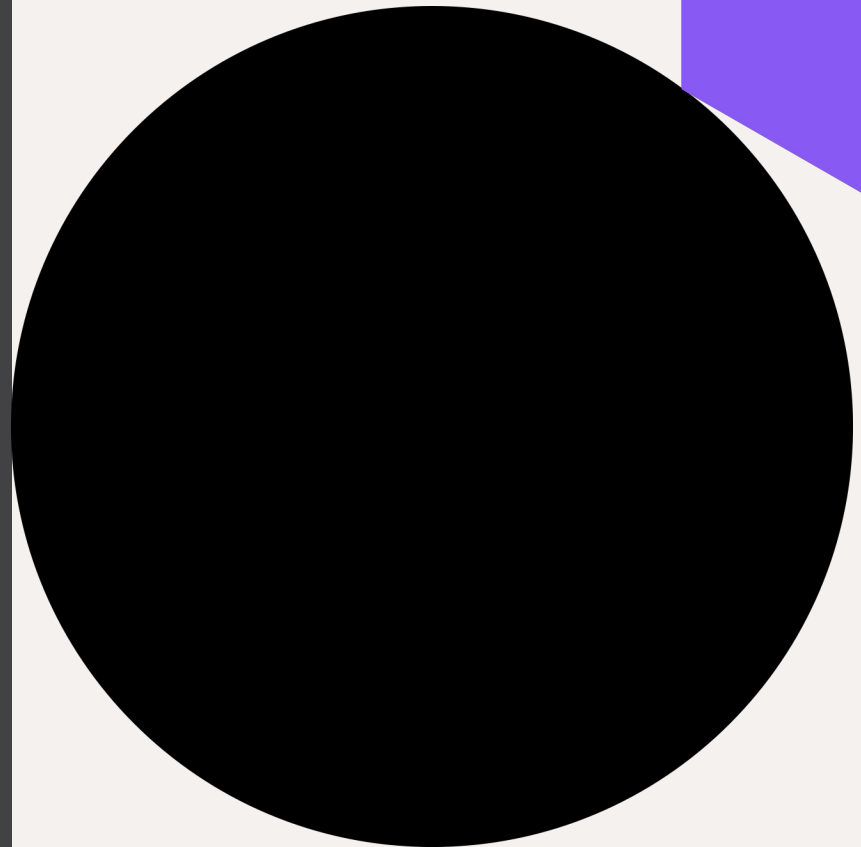
CHERI Linux

Codasip





Using capabilities to protect the kernel

Christian A. Ehrhardt <christian.ehrhardt@codasip.com>

→ Existing Efforts

Codasip

- Linux:
  - Huawei Kernel
    - Boots but based on an ancient kernel
    - Very relaxed handling of bounds
    - Huawei
  - Morello
    - Excellent work on purecap UABI
    - Kernel itself operates in legacy mode
- Non-Linux:
  - CheriBSD
  - SeL4 port

→ Codasip's CheriLinux Kernel

Codasip

- Based on the morello purecap UABI work
  - More ABI changes required
- Currently based on Linux 6.10
  - Update to 6.14 in Progress
- Can boot and run Userland (X-Server, Doom, …)
  - Qemu and FPGA
  - Might need a Qemu-Patch for SMP
- Focus on basic CHERI features (bounds, permission)
  - No levels, compartments etc., yet

→ Challenges: Overview

Codasip

- Linux assumes that a pointer fits in an `unsigned long`
- UABI changes
  - Structures with pointers change their layout, e.g. `struct iovec`
  - Shared kernel/user buffers (.eg. `io_uring`)
- Treewide `ioctl` ABI changes
- Pointers with additional data
- General pointer misuse that has worked forever
  - E.g. pointer difference between old and reallocated pointer
- Get some benefit from CHERI, i.e. apply bounds

→ Global Capabilities (64-bit)

Codasip

- `kernel_code_cap`: Read/Exec, high part of address space
- `kernel_ro_cap`: Read-Only, hight part of address space
- `kernel_data_cap`: Read/Write, high part of address space
- `user_root_cap`: Low part of address space
- (Almost) no need for an infinite cap
- Open Issues:
  - Bootup for secondary CPUs
  - CBO.INVAL

→ Applying Bounds to Pointers

Codasip

- The easy stuff:
  - `kmalloc()`/`kmem_cache_alloc()`: Just add bounds
  - `get_free_pages()`: The same
- PerCPU pointers:
  - Need to fabricate a pointer from global capability
  - Currently, pointer must be valid on CPU zero
  - Permissions are inherited, bounds are transferred.

→ Applying Bounds to Pointers

Codasip

- `lib/genalloc.c`:
  - Generic allocator for everything:
    - Physical Memory
    - Virtual Memory (that needs tags)
    - Other stuff
  - User provides ranges and allocates from them
  - Returns bounded tagged pointers if provided range is tagged

→ Applying Bounds to Pointers

Codasip

- `__va()`, `phys_to_virt()` and `page_to_virt()`:
  - No precise bounds information available
  - Best effort
  - Direct uses of `__va()` are rare
    - Returns a pointer that is bounded by the surrounding page
    - Caller must fabricate a capability if this is insufficient
- `page_to_virt()` etc. use information from the `struct page`
  - Pointer is bounded by the page allocation
  - Special case needed for boot allocated memory
  - No permission information and no tight bounds for sub-allocations

→ Pointers with Additional Data

Codasip

- Aligned pointer stored as an `unsigned long`
- Low bits re-used as some kind of flags
  - E.g. node color in a red-black tree.
- Very frequent pattern throughout the kernel
- Solution:
  - Change `unsigned long` into a `uintptr_t`
  - Detect and fix assignments
  - More on that later

→ Atomic Pointers

Codasip

- In some cases pointers need to be atomic
- Pointer can carry additional information in the low bits
- Implement `atomic_ptr_t` similar to `atomic_long_t`
- Automatically generated code

→ Pointers with data: `work_data_bits`

Codasip

```
struct work_struct {
    atomic_long_t data;
    [...]
};

#define work_data_bits(work) ((unsigned long *)(&work->data))
```

- The `data` field stores a pointer and additional bits
- The whole data field is modified atomically
- Data bits are modified atomically with stuff like:

  `__set_bit(SOME_BIT, work_data_bits(work))`
- This needs rewrite
- Issue is only detected at runtime

→ Out of bounds pointers: Unintentional

Codasip

```
struct sk_buff {
    unsigned char *head, *data; /* “data” points into “head” */
};
int pskb_expand_head(struct sk_buff *skb, int space, ...) {
    buf = alloc(...); memcpy(buf + space, skb->head, ...);
    off = buf + space - skb->head;
    free(skb->head);
    skb->head = buf;
    skb->data += off;                          /* Kills provenance */
    cheri_fixup_bounds(buf, skb->data);        /* Restore provenance */
}

#define cheri_fixup_bounds(_AUTH, _TOFIX) do { \
    void * __tofix = (__TOFIX), __auth = (__AUTH);
    (__TOFIX) = __auth + (__tofix - __auth);
} while (0)
```

- Reallocation of buffer with embedded pointers to that buffer
- Offset between old and new pointer does not work with CHERI
- `cheri_fixup_bounds()` is a NOOP for non-CHERI.

→ Out of bounds pointers: Intentional

Codasip

```
    #define ARCH_PFN_OFFSET (0x80000000 >> PAGE_SHIFT)

    struct page *page_array = alloc(....);
    struct page *vmemmap = page_array - ARCH_PFN_OFFSET;

-   #define __pfn_to_page(pfn)    (vmemmap + (pfn))
+   #define __pfn_to_page(pfn)    (page_array + (vmemmap + (pfn) - page_array))
```

- `vmemmap`:
  - Sparsely populated array
  - Different offsets per memory section in NUMA
- Base pointer in **`vmemmap`** has invalid tag and bounds
- Must restore provenance manually

→ Detecting problematic code

Codasip

- CheriLLVM has some warnings:
  - Address to pointer casts
  - Alignment issues
- Insufficient for Linux Kernel needs
- We don’t want warnings with many false positives

→ Warnings: Examples

Codasip

- `ulong f(void *p) { return (uintptr_t)p; }`
  - ➢ Want a warning for the implicit downcast
- `if ((unsigned long)p & (ALIGN-1)) { … }`
  - ➢ No need for a waring, alignment check
- `uintptr_t alignedp = (unsigned long)p & ~(ALIGN-1);`
  - ➢ Warn because the result is an aligned address, not an aligned pointer
- `if ((ulong)p1 < (ulong)p2) { … }`
  - ➢ No need to warn.
- `ulong *pptr(struct list *l) { return (ulong *)&l->pprev; }`
  - ➢ Warn because `l->pprev` is a pointer

→ Warnings: Examples

Codasip

- `void *toptr(ulong a) { return (void *)a; }`
  - ➢ Want a warning because the pointer cannot be dereferenced
  - ➢ Compiler will complain for this
- `void *toptr2(ulong a) { return (void *)(uintptr_t)a; }`
  - ➢ Want a warning
  - ➢ Compiler will not complain about this.
- `#define MAP_FAILED ((void *)-1)`
  - ➢ No need to warn about this, though
  - ➢ Hard to tell for the compiler in general!
- Many more cases

→ False Positives

Codasip

- Two contradictory goals:
  - Catch real errors at compile time
  - Avoid false positives in the checker output
- Final source should not produce warnings
  - Annotations needed to silence remaining false positives
  - Used to document that the code was looked at and is ok

→ Annotations

Codasip

- `__c_pa()`: Convert a pointer to a plain address
- `__c_ua()`: The same for an `uintptr_t` aka `__uintcap_t`
- `__c_fakep()`: Create a provenance-free pointer from an address
- `__c_fakeu()`: The same for `uintptr_t`
- Only use these if you checked the code and the cast is ok!

→ Checking casts with sparse

Codasip

- Smoothless integration into kernel development
- Hacky check to get at least something:
  - Define `__uintcap_t` as a 128-bit integer for sparse
  - Add warnings if a 128-bit integer is cast to a smaller one
- Works ok but produces many false positives
  - Most annotations are in place for 6.10.
  - Large RiscV CHERI config compiles without warnings
- Won't work for C++
- Would have to hard code exceptions into the tool
- Cannot handle some cases

→ Move to clang-tidy

Codasip

- Integration with the kernel works
- More widely used tool
- It is our check, so we define the rules
- Checks for all of the above cases implemented
  - Will hit cherillvm repo with the next push
- Can suppress warnings for many cases
- Catches all explicit and implicit casts!

→ Detecting Integer to Capability Promotion

Codasip

```
uintptr_t ptr;
unsigned long add
ptr = ptr + add;
```

- According to the C-rules **`add`** is promoted to **`uintptr_t`** first
- Implicit cast that the tools detect
- Special handling to silence warning for operands to a binary operation where provenance is derived from the other operand.

→ Bulk editing

Codasip

```
struct file_operations {
    [...]
    long (*unlocked_ioctl)(struct file *, unsigned int cmd, user_uintptr_t arg);
    [...]
};
```

- Prototype of **`->unlocked_ioctl`** changes
- Dozens of similar function pointers in other structs
- Several hundred different functions assigned to these members
- Dependant functions that **`arg`** is passed to
- Similar issues for **`taskelet_init()`** and others

→ Bulk editing: Coccinelle Example

Codasip

```
@r1@
expression __arg1, __data;
identifier __func;
typedef uintptr_t;
@@
     tasklet_init(__arg1, __func,
-    (unsigned long)
+    (uintptr_t)
     __data
     );

@@
identifier r1.__func;
identifier __data;
typedef uintptr_t;
@@
     void __func(
-    unsigned long __data
+    uintptr_t __data
     )
```

```
@@
-static void tasklet_function(unsigned long priv)
+static void tasklet_function(uintptr_t priv)
 {
      [...]
 }

@@
         tasklet_init(&priv>tasklet,
                  tasklet_function,
-                 (unsigned long)priv);
+                 (uintptr_t)priv);
 }
```

→ Bulk editing: Coccinelle

Codasip

- Understands a bit of C
- Can bulk edit code that is otherwise not compiled.
- Not precise enough for more complicated stuff
- Manual fixups and review required

→ Other UAPI issues

Codasip

- Goal: Never create pointers to user space from addresses
- Fine for register passed pointers
- struct layout for memory passed pointers changes
- Sometimes size increase in structures outgrows hard limits
- Ring buffers shared between kernel and user are problematic
- Compat handling for non-CHERI userland code missing in many cases
  - Full blown solution would have 3 compat variants

→ Major Open Issues

Codasip

- Cleanup commit history
- Module support is missing
  - Not rocket science
  - Was blocked by upcoming compiler changes
  - Slightly complicated because on disk ELF structures are modified in-place
- Direct user page access (e.g. `pin_user_pages()`) must check capabilities
- Check Capabilities on free?!
- Compat support

→ Major Open Issues

Codasip

- eBPF
  - Compiles but not tested
  - Classic BPF should be doable
  - eBPF needs pointers in BPF registers and arithmetic on them
- io_uring:
  - Structures in the ring exceed maximum entry size
  - UAPI breakage
  - Force use of larger structures
  - Some support already there.
- Other Features with similar issues, e.g. async io

→ Major Open Issues

Codasip

- ACPI and UEFI
  - Disabled for now
  - Especially UEFI API will need more thought
- OpenSBI
  - Similar issues
  - Compiled with CHERI support
  - API requires the kernel to provide capabilities

→ Major Open Issues

Codasip

- Issues with drivers:
  - Several drivers do problematic stuff
  - Opaque cookies passed to hardware
  - Shared structure layout between hardware and software
  - Overlayed structures
- Jump Labels:
  - Need more compiler support on RISC-V
  - Disabled for now
- Virtualization
  - Not yet supported by Codasip hardware or Qemu

→ Debugging and Hardening Features

Codasip

- Many debug/hardening features still disabled
- KASAN, UBSAN, LockDep
- Several sanitizers lack full compiler support for CHERI
- GCOV, KCOV
- PERF events
- Structure Layout Randomization
  - Compiler gets offsets wrong
  - Pointer alignment issues

→ LTP Testing

Codasip

- LTP tests compile and run
- No kernel crashes :-)
- Lots of tests pass
- Still many failures and skipped tests
  - Needs investigation on a case-by-case basis
  - Some failures caused by musl with CHERI

→ Future Features

Codasip

- Virtualization
- Temporal safety support for user space
  - And for the kernel…
- CHERI Levels
- Subobject bounds
- Compartmentalization support for user space
  - And for the kernel…

Thank you!

Questions?