Morello Linux kernel overview

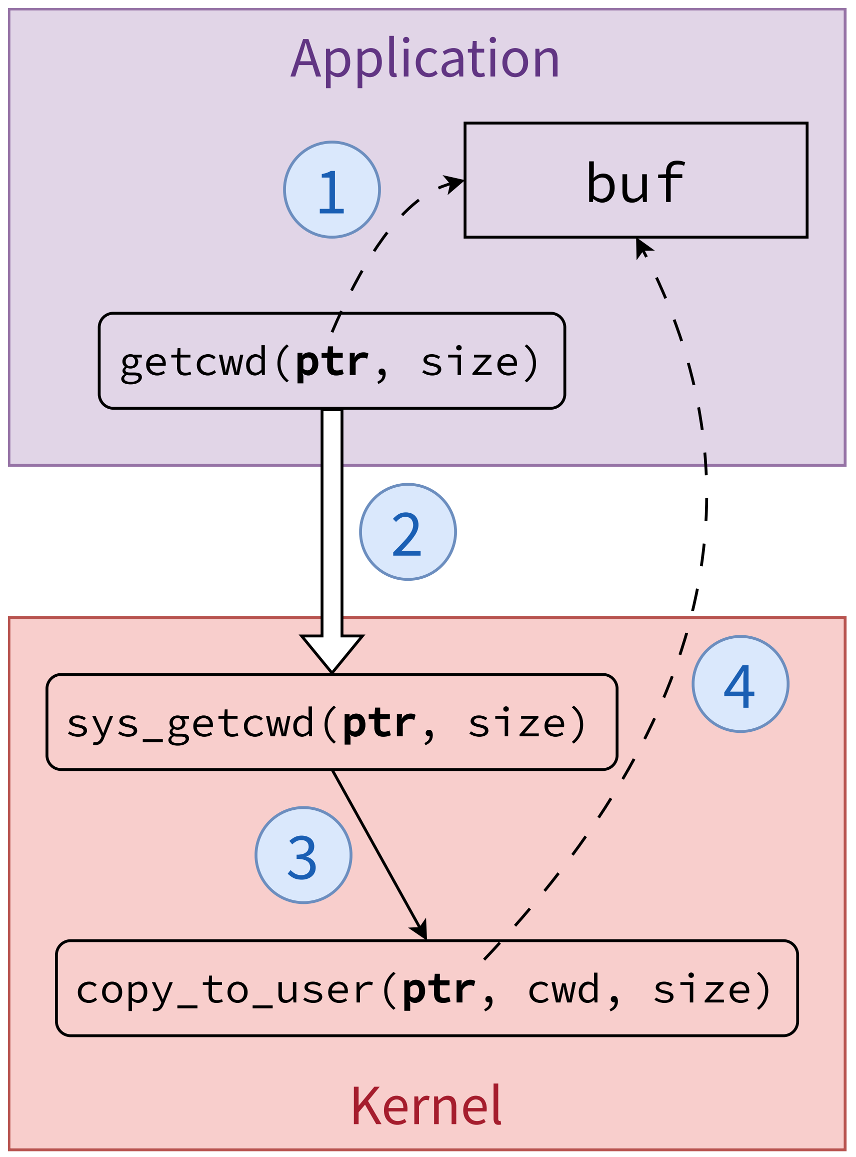
Kevin Brodsky

The pure-capability userspace ABI (PCuABI)

- New kernel-user ABI required
  - **Functional** angle: userspace uses larger, "special" pointers
  - **Security** angle: enforcement of capability properties

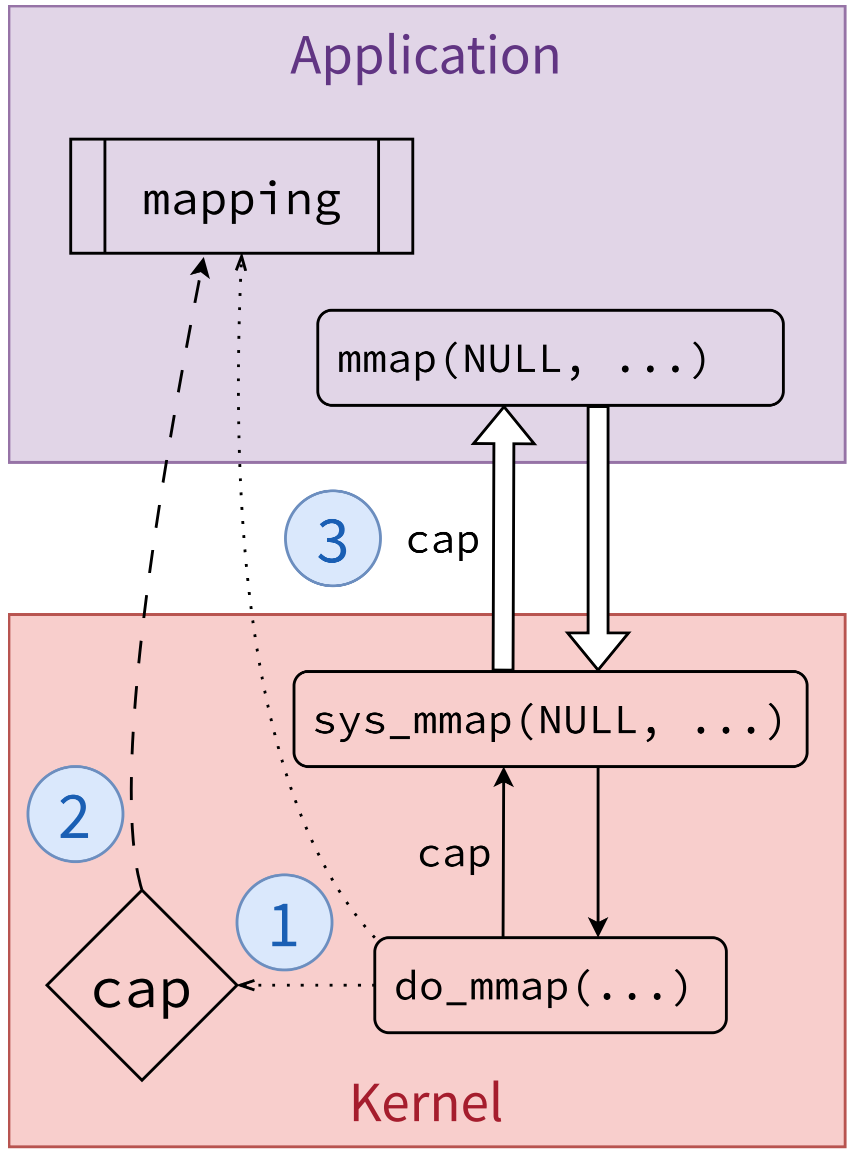
- Input pointers (user → kernel)
  - Most common (syscall arguments)
- Output pointers (kernel → user)
  - Mostly address space management (`mmap()`, ...)

Kernel-user interactions: input pointers



1. `ptr` is a capability allowing access to `buf` only
2. PCuABI syscall: `ptr` passed as a capability (`c0`)
3. `ptr` is **propagated** as a capability
4. Capability-based uaccess: `buf` is accessed via the `ptr` capability
   - Exception triggered if `ptr` does not authorise the access (`-EFAULT`)

Kernel-user interactions: output pointers



1. New mapping and capability created
2. `cap` grants access to `mapping`
   - Minimal bounds and permissions
   - `cap` owns `mapping` (VMem permission)
     → allows calling `mprotect()`, etc.
3. `cap` is returned to userspace

Capability propagation: hybrid approach

- All **user pointers** become capabilities through annotations
  - C extension: `void * __capability`
  - Leveraging `__user` — some fixups needed as `__user` prefixes `*`
- Primitive types (in-kernel ABI) **unchanged**
  - Kernel pointers, `long` still 64-bit
- Strict separation between kernel and user pointers
  - New APIs to manipulate user pointers: `<linux/user_ptr.h>`
  - **Address ≠ pointer**

Representing user pointers

- User pointers often represented as integers

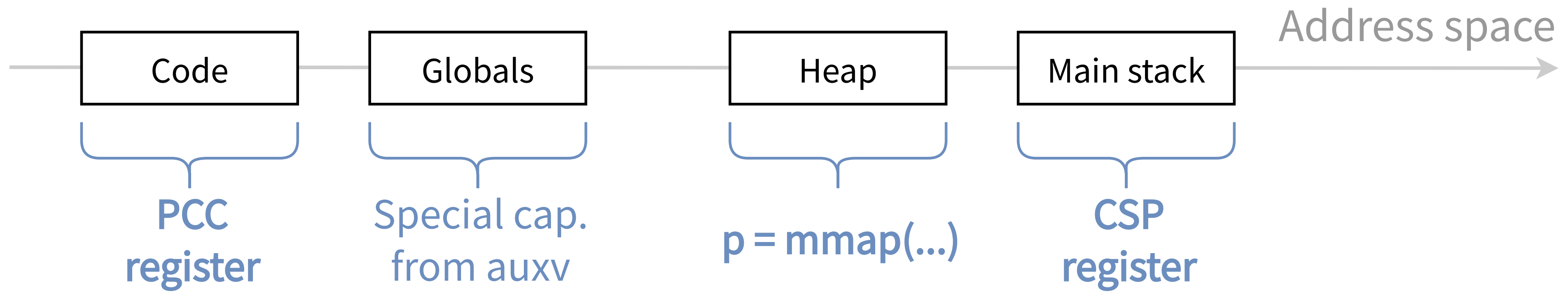
```
long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
```

```
struct io_uring_sqe {
        ...
        __u64 addr; /* pointer to buffer or iovecs */
        ...
};
```

- Not capability-friendly: `(long)uptr` **truncates** `uptr`
- **New types required** — ABI-dependent definition

```
unsigned long  ⟶  user_uintptr_t
        __u64  ⟶  __kernel_uintptr_t
```

Providing root capabilities



- Mainly: `mmap()`, `mremap()`
  - Return a capability with minimal bounds and permissions
  - Special handling of input pointers
- Initial process environment
  - PCC (code capability), CSP (capability stack pointer)
  - Arguments (`argv`) and env. variables (`envp`): arrays of capabilities
  - New `auxv` entries for special root capabilities (mainly for relocation processing)

↺

compat64: 64-bit compat layer

- Standard 64-bit ABI provided via compat
  - Instead of 32-bit (not present on Morello HW)
- compat → native pointer conversion required
  - 64-bit address → valid capability
    - User memory accessed via a capability in any case
  - **Strict usage** of `compat_ptr()`
  - Pointer arguments converted directly in syscall wrapper

Application

```
char buf[128];
struct iovec buf_iov = {
    .iov_base = &buf,
    .iov_len = 128
};
readv(fd, &buf_iov, 1);
```

Kernel

```
sys_readv(int fd, struct iovec __user *iov,
          int iovcnt);
```

- `iov` capability created by the syscall wrapper
- `sys_readv()` uses `compat_ptr(iov[0].iov_base)`

compat64: pain points

- Widespread assumption that compat is 32-bit
  - We want most of the existing compat code… but not all of it (e.g. 32-bit time types)
  - `unsigned int` ⟶ `compat_ulong_t`
  - Sometimes: just use native handler

- Additional handling where types have been enlarged for native pointers
  - **Any uapi struct change requires adding compat handling**
    - Typically: `__u64` ⟶ `__kernel_uintptr_t`
    - Layout conversion not always doable upfront (e.g. `union bpf_attr`)
  - compat ioctl handlers must always convert input pointers

Morello Linux kernel

- Morello Linux kernel fork hosted on morello-project.org
  - Mainline-based (currently 6.7) + ~500 patches
  - Support for Morello and pure-capability userspace (**hybrid** approach)
    - Selection of drivers available in PCuABI
  - 64-bit compat $\longrightarrow$ **major effort**
- Pure-capability kernel-user ABI specification — wiki page on morello-project.org
  - Refined over 3+ years, mostly stable
  - Extensive, many subtleties around `mmap()` and address space management in general

Thank you
Danke
Merci
ありがとう
谢谢
Gracias
Tack
Takk
Kiitos
감사합니다
धन्यवाद
تشکر