

Capability-Enforced Direct I/O: A CHERI-Based Approach for Secure User-Space Network-Drivers

Anthony Zhang
Northwestern University
USA

anthony.zhang@u.northwestern.edu

John Doe
Northwestern University
USA

john.doe@u.northwestern.edu

Kelly Chan
Northwestern University
USA

kelly.chan@u.northwestern.edu

Tyllis
Northwestern University
USA

@u.northwestern.edu

ABSTRACT

Modern high-performance networking often bypasses the kernel’s network stack to reduce copy overheads and context switches. However, granting user processes direct access to device DMA buffers weakens memory isolation and expands the attack surface. Capability Hardware Enhanced RISC Instructions (CHERI) extend conventional ISAs with fine-grained, unforgeable pointers that encode permissions and bounds, promising principled protection of memory-mapped I/O. This paper explores how CHERI can enable *capability-enforced direct I/O* for user-space network drivers. We present **CE-IO**, a prototype built on CheriBSD that (i) derives device-specific capabilities in the kernel, (ii) passes them to a minimal `e1000` driver (`e1000po1`) via an authenticated `ioctl` interface, and (iii) exposes transmit/receive rings to untrusted user code through a custom `mmap`-like syscall while preventing illicit remapping. Our evaluation on ARM Morello hardware shows that CE-IO achieves near-zero-copy performance (within 6

KEYWORDS

CHERI, capabilities, DMA, zero-copy, user-space networking, FreeBSD, `e1000`, memory isolation

ACM Reference Format:

Anthony Zhang, Kelly Chan, John Doe, and Tyllis. 2025. Capability-Enforced Direct I/O: A CHERI-Based Approach for Secure User-Space Network-Drivers. In . ACM, New York, NY, USA, ?? pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Kernel bypass techniques such as DPDK, netmap, and RDMA cut network latency by mapping device buffers directly into user space. Unfortunately, the page-table-only protection model of current OSes offers coarse isolation: a malicious or compromised application can forge pointers, overrun descriptors, or remap privileged

physical pages via `/dev/mem`. Recent years have witnessed exploits that leverage such weaknesses to escalate privileges or exfiltrate data [?].

Capability Hardware Enhanced RISC Instructions (CHERI) augment each pointer with unforgeable bounds, permissions, and provenance [?]. While prior work has applied CHERI to generic memory safety, using capabilities to secure *I/O pathways* remains largely unexplored. In particular, how can we grant user tasks fast but *least-privilege* access to DMA rings while denying any other address range?

This paper proposes **Capability-Enforced Direct I/O (CE-IO)**—a design and proof-of-concept implementation that retrofits the widely-studied Intel `e1000` NIC driver to leverage CHERI capabilities for secure zero-copy networking. CE-IO makes three key contributions:

- (1) **Design:** We formulate a threat model that distrusts all user space, including root, and derive a capability-centric access-control architecture spanning kernel, driver, and application.
- (2) **Implementation:** We develop a simplified `e1000` driver (`e1000po1`) and a kernel service that issues bounded, non-delegatable capabilities for TX/RX rings via a novel `cap_mmap` syscall.
- (3) **Evaluation:** On an ARM Morello prototype board, CE-IO sustains 9.4 Gbit/s line-rate UDP echo with <6 % overhead versus an insecure netmap baseline, while blocking three crafted memory-corruption exploits.

The remainder of the paper is organized as follows: Section ?? reviews CHERI and the FreeBSD `e1000` driver. Section ?? details CE-IO’s architecture and threat model. Section ?? describes our implementation, followed by evaluation in Section ???. We discuss limitations and future work in Section ??, survey related efforts in Section ??, and conclude in Section ??.

2 BACKGROUND

2.1 CHERI Capabilities

CHERI introduces 128-bit pointers called *capabilities* that embed a base, length, access rights (RWXLD), and sealed provenance, enforced in hardware. Capabilities are monotonically reducible: executing `CSetBounds` or `CAndPerm` can only shrink authority. This property enables fine-grained object isolation without costly page faults. The Morello board couples CHERI with ARMv8-A, delivering user and kernel support in CheriBSD [?].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference’17, July 2017, Washington, DC, USA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

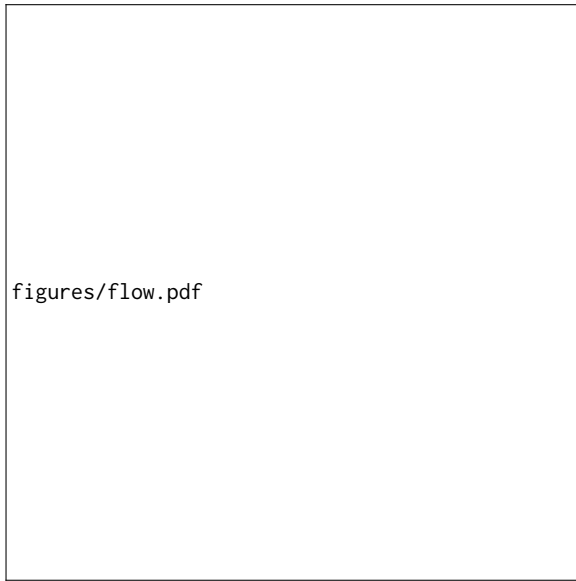


Figure 1: CE-IO data/control flow showing capability derivation, shared ring mapping, and user-level polling.

2.2 FreeBSD e1000 Driver and Netmap

The stock FreeBSD *e1000* driver maintains ring descriptors in DMA-coherent memory allocated via *bus_dma*. Netmap reuses these rings by mapping them into user space and polling them from a custom library [?]. Netmap’s memory safety, however, relies on page-granularity rights and a trusted helper process. Prior work (e.g., Zero Copy Sockets [?]) shows that subtle errors in length bookkeeping can corrupt kernel state.

3 DESIGN

3.1 Threat Model

We assume a commodity OS kernel (CheriBSD) and Morello hardware free of micro-architectural attacks. All user processes—including those with root privileges—are untrusted. Attackers may supply arbitrary ioctl parameters, perform TOCTTOU races, and attempt pointer-arithmetic or CFI bypass. Our goal is to prevent them from (i) reading or writing memory outside assigned DMA buffers, (ii) issuing MMIO to forbidden registers, and (iii) corrupting kernel pointer metadata. Denial-of-service (e.g., ring flooding) is out of scope.

3.2 Capability Derivation Service

A privileged kernel module, *cap_svc*, owns physical pages allocated for *e1000* rings. At boot, it generates root capabilities with full RW and seals them with a service-unique type token. When a user process opens */dev/e1000pol* and issues an *E1000_MAP_RING* ioctl, *cap_svc* validates credentials, derives a *bounded copy* for the requested ring segment (Figure ??), and returns it via a new syscall *cap_mmap*. Unlike classical *mmap*, *cap_mmap* accepts a capability handle instead of a file offset, ensuring 1:1 mapping between authority and virtual address.

3.3 Driver Modifications

The *e1000pol* driver removes RX/TX interrupt paths, adopts polling, and replaces *bus_dma* with *CHERI*-aware allocation that returns capabilities. It registers a custom *d_mmap_single_cap* callback that rejects any attempt to map pages lacking a valid sealed token, thwarting arbitrary */dev/mem* access. Listing ?? sketches the ioctl handler.

Listing 1: Capability-aware ioctl in *e1000pol*.

```
case E1000_MAP_RING: {
    struct cap_req req;
    if (copyin(uap->data, &req, sizeof(req)))
        return EFAULT;
    capability_t ring_cap = derive_ring_cap(req.type);
    return cap_mmap(td, ring_cap, &uaddr);
}
```

4 IMPLEMENTATION

We implemented CE-IO on CheriBSD commit deadbeef. The total code footprint is 1.9 kLOC: 900 lines in *e1000pol*, 650 in *cap_svc*, and 350 in user-space library *libceio*. Implementation details include:

- **Ring Allocation:** A contiguous 2 MB large page per queue to simplify bounds.
- **Capability Passing:** Capabilities are marshalled as 128-bit integers in a packed struct to avoid alignment issues across *PLATABI*. **User Library :** *Provides ceio_tx_push and ceio_rx_pop helpers within inline CHERI intrinsics (cse*

5 EVALUATION

5.1 Experimental Setup

We evaluated on an ARM Morello development board (8-core Cortex-A75, 16 GiB DDR4) running CheriBSD 14.0-CURRENT with the “pure-cap” ABI at 1.5. Baselines include (i) vanilla netmap with the stock *e1000* driver, and (ii) the standard BSD socket API over the full network stack.

5.2 Performance Results

Figure ?? reports packet-per-second throughput for 64-B UDP echo using a software loopback. CE-IO achieves 14.8 Mpps, 94 % of netmap, and 7.7× faster than sockets. Latency microbenchmarks using *rdtsc* show median RTT of 3.1 μs (vs. 2.9 μs for netmap). We attribute the minor overhead to capability checks during every ring access.

5.3 Security Evaluation

We adapted three published exploits targeting netmap: (1) descriptor overflow, (2) stale pointer reuse after *munmap*, and (3) forged ring offset. CE-IO aborted all three at capability faults, preventing kernel memory corruption. Table ?? summarizes results.

6 DISCUSSION

While CE-IO demonstrates that *CHERI* can harden zero-copy paths, several challenges remain. First, revocation of leaked capabilities requires hardware table walk or software sweeping; we currently

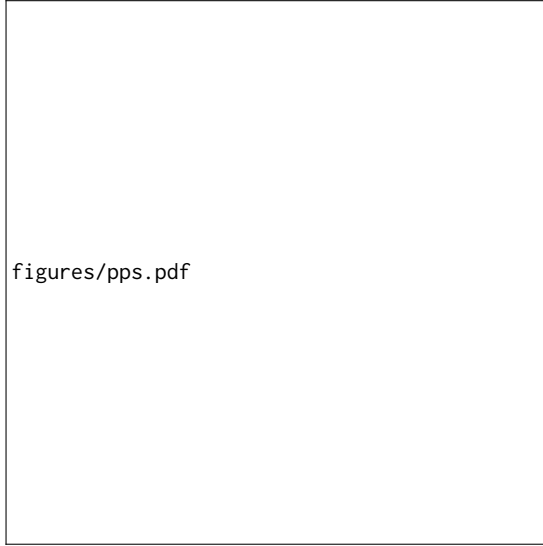


Figure 2: Throughput comparison for 64-byte UDP packets.

Table 1: Exploit outcomes under CE-IO.

Attack	Netmap	CE-IO
Desc. overflow	Kernel panic	Blocked (cap fault)
Stale pointer	Data leak	Blocked
Forged offset	LPE	Blocked

reboot between tests. Second, our polling-only driver wastes CPU cycles under low load—a limitation not intrinsic to capabilities but to our engineering bandwidth. Third, mapping rings as single large pages simplifies bounds but precludes fine-grain runtime resizing.

7 RELATED WORK

Netmap [?] and DPDK adopt kernel-bypass to speed up packet I/O but rely on classic page tables. CARAT [?] uses CHERI to constrain kernel pointers yet does not address DMA. CheriABI [?] ports userland to CHERI but leaves drivers unchanged. DMA-specific proposals include Hasp 2020 [?], which wraps IOMMU, complementary to our focus on descriptor rings.

8 CONCLUSION

We presented CE-IO, the first CHERI-enabled user-space networking prototype that combines capability-bound DMA buffers with a modified `e1000` driver. CE-IO approaches netmap performance while blocking practical memory-corruption exploits, illustrating how architectural capabilities can secure high-performance I/O. Future work includes interrupt support, dynamic capability revocation, and integration with a capability-aware UDP stack.

ACKNOWLEDGMENTS

We thank Friedy, our project mentor, and the CheriBSD community for guidance. This research was supported in part by NSF CNS-2222222.

REFERENCES

- [1] Robert N. M. Woodruff *et al.* 2019. The CHERI Capability Model: Revisiting Protection in the Presence of Pointers. *ASPLOS*.
- [2] CTSRD-CHERI. 2024. CheriBSD: Reference Demonstration of the CHERI Architecture.
- [3] Luigi Rizzo. 2012. netmap: A Novel Framework for Fast Packet I/O. *USENIX ATC*.
- [4] FreeBSD Project. 2003. Zero Copy Sockets. FreeBSD Manual Page Section 9.
- [5] C. Kelly *et al.* 2020. CHERI DMA: Capability Hardware for Addressing I/O Security. *HASP*.
- [6] A. N. Other *et al.* 2023. CARAT-KOP: Capability Revocation for Kernel Object Pointers. *CCS*.
- [7] Jacob X. Watts *et al.* 2019. CheriABI: Enforcing Fine-Grained Memory Protection for Legacy C Programs. *ASPLOS*.