CHERI under Fault Injection

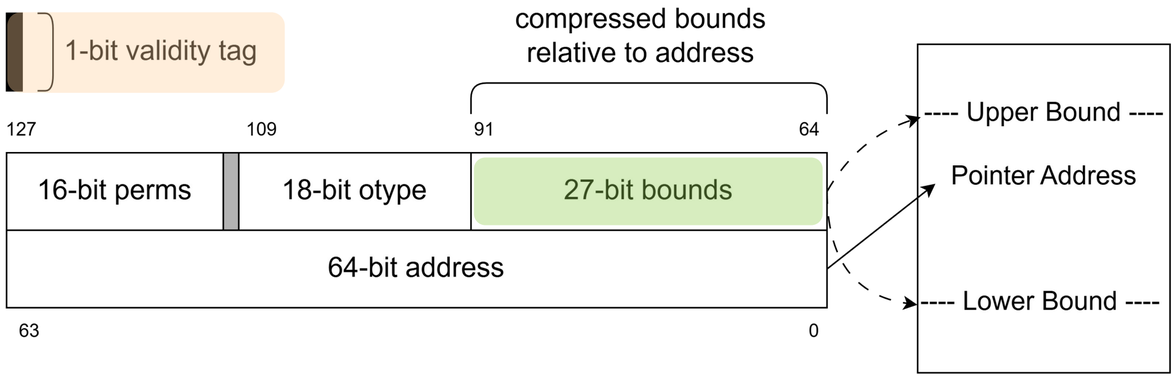
An Emulation-Based Security Evaluation of Capability Hardware Enhanced RISC Instructions

Eva Sophia Schreyer, Merve Gülmez, Thomas Nyman

To what extent is the CHERI architecture robust against fault injection attacks and what fault injection vulnerabilities could compromise CHERI's memory-safety guarantees?

Hardware-assistance for memory safety, like **CHERI**, is attractive for addressing **memory-safety** for codebases developed in unsafe languages, e.g., C and C++

- CHERI addresses spatial and temporal safety for a **software threat model**



CHERI capabilities are twice the size of a native pointer by extending them with metadata: permissions, object type, bounds and validity tag

Fault injection (FI) can induce bit flips in memory, alter instructions or even skip instructions /security checks

Physical Methods	Software-induced Methods
Voltage/clock glitching, Optical/electromagnetic faults	CLKSCREW, Rowhammer

We **emulate FI primitives** within a QEMU-based CHERI-RISC-V emulator:

Primitive	Function
Tag Flip	Corrupt tag bit
Bounds Flip	Corrupt bounds
Bit flip in memory relevant to revocation	Corrupt revocation-relevant memory
Different granularity Store Skip Check	Skips bounds check on store instructions (different granularities)

Attack	Success	# of Faults	Bypassed Protection
Forged Pointer	✓	1	Tag
Buffer Overflow	✓	1	Bounds
Control-flow Hijacking			
- Return address part overwrite	✓	2	Bounds and Tag
- Return capability overwrite [1]	✓	1	Bounds
- Skip check	✓	Depends	Check on Store
Data-only Attack			
- Standard	✓	1	Bounds
- Skip check	✓	Depends	Check on Store
Use-after-free	✓	1	Revocation-relevant memory

[1] Limitation: requires a valid and executable capability to overwrite the entire return capability with it

Findings

- Stored metadata has no integrity verification
  → **Bit flips in capabilities and revocation-relevant memory go undetected**
- CHERI relies on checks in stores and loads
  → **Skipped checks leave no trace**
- Not design flaws: fault injection falls outside CHERI's stated threat model
  → **New hardware defenses should explicitly consider FI**

Limitations

- Hardware emulation (QEMU) used for reproducibility and fine-grained control but does not model fault injection surface of a real chip
- Injected faults assume ideal spatial and temporal precision

→ Reported **fault counts are lower bounds**

- FI-induced control-flow hijacking needs suitable code gadgets

CHERI **does not prevent** the demonstrated attacks, but it **raises the difficulty**:

- *Software* means *cannot manipulate* metadata or craft capabilities
- Each *fault bypasses only one specific protection*
  → **Complex attacks need to chain multiple FIs**
- Effect: instead of needing only a vulnerable program
  → **Vulnerable program + ability to inject fault**

Complete thesis work will be published on the DIVA website of KTH Royal Institute of Technology with the same title

Possible mitigations	Effect	Applicability
ECC memory	Detects and corrects bit flips in tag storage and revocation bitmap	Yes, but requires hardware changes
Software redundancy	Detects and masks or recovers from check skips	Partially[2]
Physical tamper prevention	Makes physical fault injection more difficult	Yes, but requires specialized hardware
ASLR	Limits CFH exploitability	Yes

[2] CHERI: check is part of store/load instructions → repeating the entire instruction cannot undo the previous executions write/read → repetition of check must occur before the actual load/store

In ongoing work we evaluate other memory-safety technologies and their fault resistance.

Legend:
Enc. = encrypted/encryption
IN = in-band with protected value
A = accessible
SW = Software
OOB = out-of band with protected value
IA = inaccessible
HW = hardware

	Structural Properties			Enforcement properties				Fault properties		
	Metadata Location	Software Accessibility	Basis	Location	Type	Explicit vs Implicit	Frequency	1 Fault Detectability	# Faults	Type
CHERI	IN: bounds, perm, otype OOB: tag	A: bounds, perm, otype IA: tag	Structural	HW	Permit check	Implicit	Per-access	No	1-2 ≥1	Bitflip Skip check
PAC	IN: signature OOB: key	A: signature IA: key	Cryptographic	HW	Match metadata and value	Explicit	Control-flow boundaries	Yes	1	Skip instruction OR defeat crypto
LIPPEN	IN: encrypted pointer OOB: key	A: enc. pointer IA: key	Cryptographic	HW	No-explicit-check	Explicit	Per-access	Partially	2	Skip enc./ decrypt OR defeat crypto
PointGuard	IN: enc. pointer OOB: key	A: enc. pointer IA: key	Cryptographic	SW	No-explicit-check	Explicit	Per-access	Partially	2	Skip enc./ decrypt OR defeat crypto
Intel Shadow Stack	OOB	IA	Duplication	HW	Match metadata and value	Implicit	Control-flow boundaries	Yes	1 2	Skip check Bitflip
Software Shadow Stack	OOB	A, but hidden by changing location	Duplication	SW	No-explicit-check	Explicit	Control-flow boundaries	No	1	Bitflip
Memory Tagging	IN: pointer tag OOB: memory tags	A	Structural	HW	Match metadata	Implicit[3]	Per-access	Partially	1 ≥1	Bitflip Skip check

[3] The comparison of tags is implicitly done on store/load, but the assignment of tags is done explicitly through instructions.