The CHERI capability model: Revisiting RISC in an age of risk

Jonathan Woodruff, Robert N. M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, Michael Roe

Presented by Richard Muri
MIT 6.888 – Secure Hardware Design

Motivation

Memory safety has been an active research area for many years, and continues to be an open problem in the modern era. In fact, **70% of security patches** made to Microsoft Windows between 2007 and 2019 were addressing memory safety issues.[1]

Program security is dependent on maintaining isolation and pointer safety (bounds and permissions on a memory region). Address translation is a insufficient mechanism for making security guarantees. A stronger protection mechanism is required, but it must meet practical demands such as compatibility and scalability.

Capability Hardware Enhanced RISC Instructions (**CHERI**) introduces a hybrid capability-based memory system:

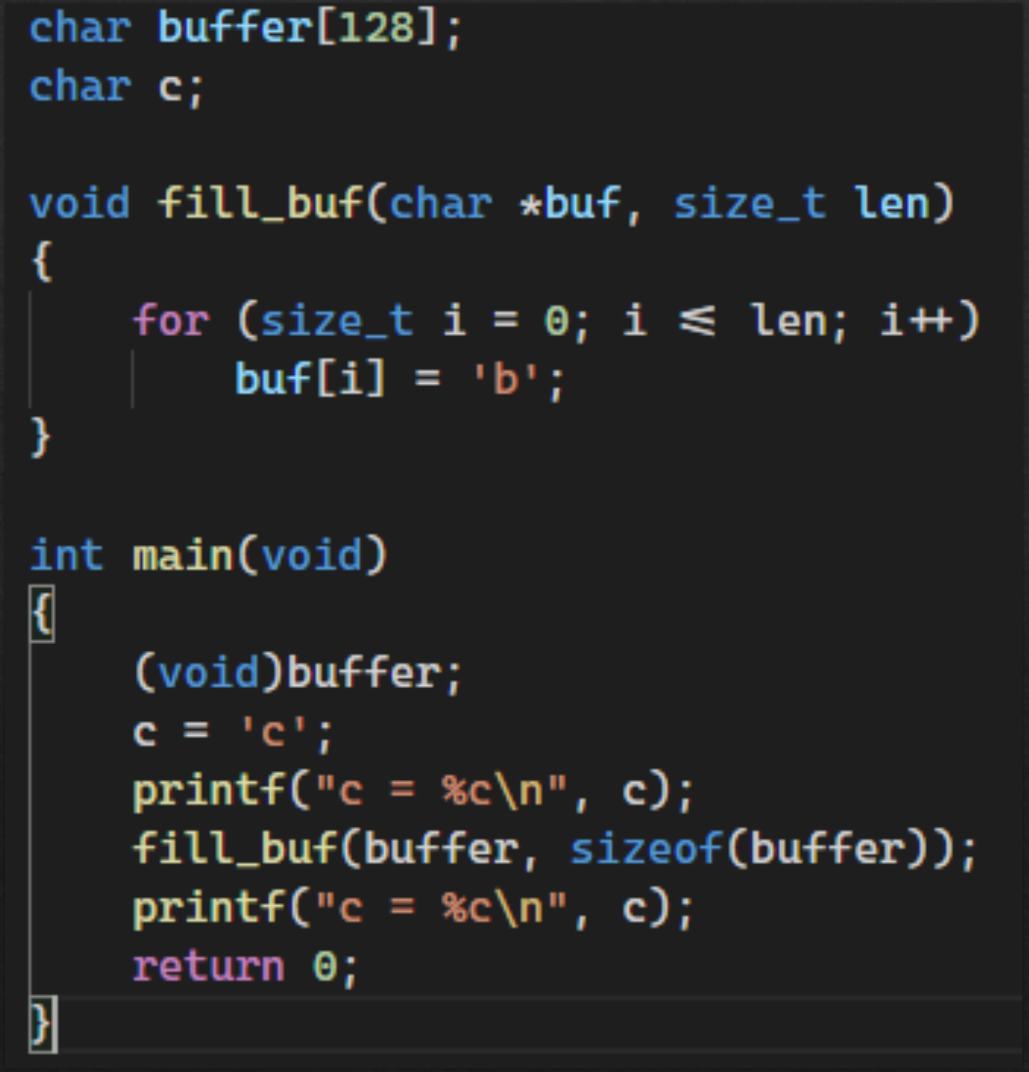
- Capabilities are hardware-accelerated objects with stronger safety guarantees than pointers
- Offers binary and source capability with existing systems

[1] Trends, challenges, and strategic shifts in software vulnerability mitigation landscape. Miller 2019

Key Contributions

- ISA extension and architecture that supports hardware capabilities
- Support for incremental adoption
- Feasibly scalable method of providing fine grained, dynamic memory protection domains

Spot the buffer overflow



```c
char buffer[128];
char c;

void fill_buf(char *buf, size_t len)
{
    for (size_t i = 0; i <= len; i++)
        buf[i] = 'b';
}

int main(void)
{
    (void)buffer;
    c = 'c';
    printf("c = %c\n", c);
    fill_buf(buffer, sizeof(buffer));
    printf("c = %c\n", c);
    return 0;
}
```

Virtual Addresses Aren't Enough!

A key observation behind CHERI is address translation and page tables provide coarse-grained inter-program isolation and protection, but are less useful for intra-program protection

- Address validity: associate protections with a region of memory (such as W^X)
  - Paged virtual memory is the de facto modern technique for address validity
  - If a program accesses a valid address, it can use the memory
- Pointer safety: associate protections with an object
  - Capabilities provide bounds, type information, and permissions for individually allocated objects
  - A program must use a sufficiently privileged capability to access an object, even if it owns the whole page
- Traditional systems view memory as "flat", while capabilities are "segmented"

Threat Model

Unlike many of the papers we have read in 6.888, CHERI does not explicitly define a threat model. It focuses on introducing a framework of primitives to build a secure ecosystem on top. My interpretation of an implicit threat model

- A user space process gains fine-grained memory segmentation to provide memory safety
- Trusted:
  - Hardware
  - Operating System
  - Firmware
  - Compiler
- Forces ‘intentionality’ to memory accesses – capabilities bundle bounds and permission checks

Discussion

In which people other than Richard express their opinions about CHERI

Richard’s Opinion

Strengths:

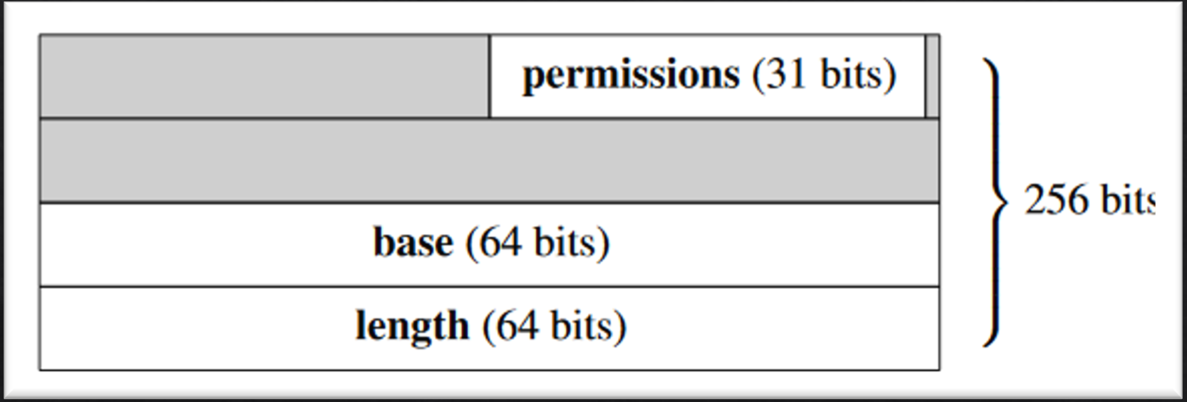
- Unforgeability provides a ‘root of trust’ approach to memory access
- CHERI takes an address the problem, not the symptom approach to spatial memory safety
- Heavy emphasis is placed on compatibility – until you have code for it, a processor is just cleverly arranged sand

Weaknesses:

- This paper advertises amazing new primitives, but then largely describes theory or future work when it comes to actually using new capabilities
- I would like to see some presentation of the problems CHERI is solving – what memory safety vulnerabilities would CHERI have caught? What vulnerabilities does CHERI fail to catch?

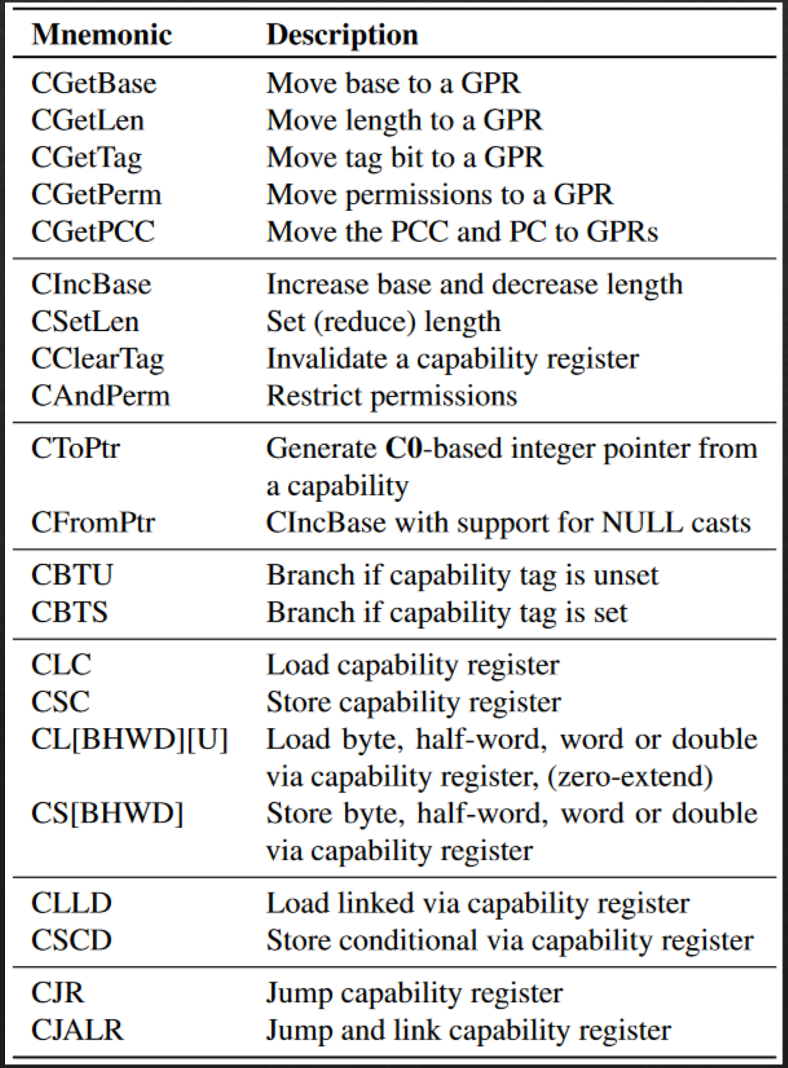
What is a capability?

- Capabilities replace pointers as the key to accessing memory
- Analogous to a fat pointer – a starting address plus a valid range
- Includes permission field describing WRX data/capability privileges
- Enforces run-time invariants, such as bounds checking or permission checking
  - Hardware exception issued if capability used incorrectly
- Single bit validity tag



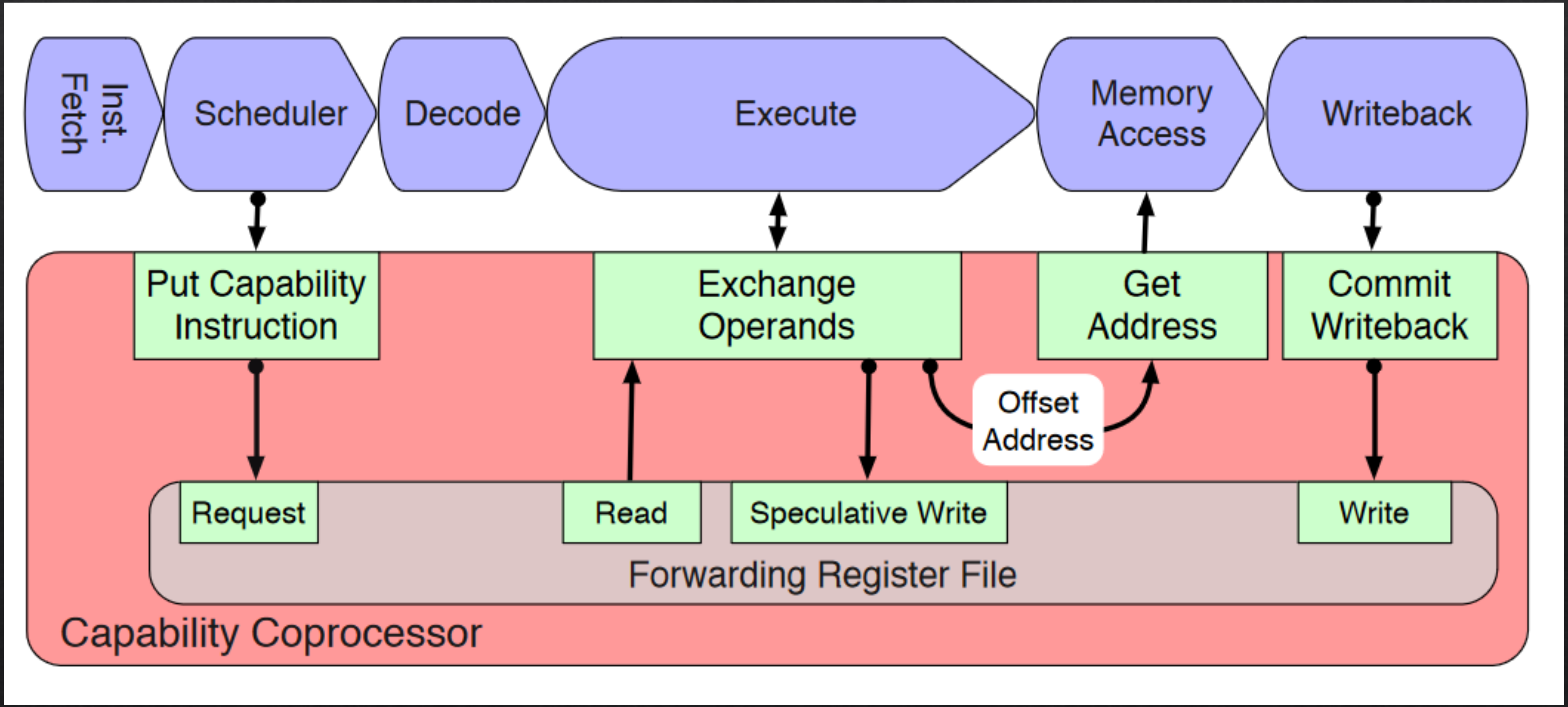
Capability Architecture

- Capabilities reside in 256 bit registers
- Dedicated separate size 32 register file
- Acts as an extension to existing ISA



Mnemonic	Description
CGetBase	Move base to a GPR
CGetLen	Move length to a GPR
CGetTag	Move tag bit to a GPR
CGetPerm	Move permissions to a GPR
CGetPCC	Move the PCC and PC to GPRs
CIncBase	Increase base and decrease length
CSetLen	Set (reduce) length
CClearTag	Invalidate a capability register
CAndPerm	Restrict permissions
CToPtr	Generate **C0**-based integer pointer from a capability
CFromPtr	CIncBase with support for NULL casts
CBTU	Branch if capability tag is unset
CBTS	Branch if capability tag is set
CLC	Load capability register
CSC	Store capability register
CL[BHWD][U]	Load byte, half-word, word or double via capability register, (zero-extend)
CS[BHWD]	Store byte, half-word, word or double via capability register
CLLD	Load linked via capability register
CSCD	Store conditional via capability register
CJR	Jump capability register
CJALR	Jump and link capability register

Capability Co-processor Block Diagram



Protecting Capabilities

- Tagged memory prevents capability manipulation – all non-capable stores invalidate in-memory capabilities
  - All capability manipulations must happen explicitly via the ISA
- Protection mechanisms are implemented in hardware to avoid requiring syscall overhead
- Capabilities are “unforgeable”
  - All changes to capabilities result in privilege de-escalation
  - Starting with the One Capability to Rule Them All arbitrary restricted domains can be constructed by deriving new capabilities

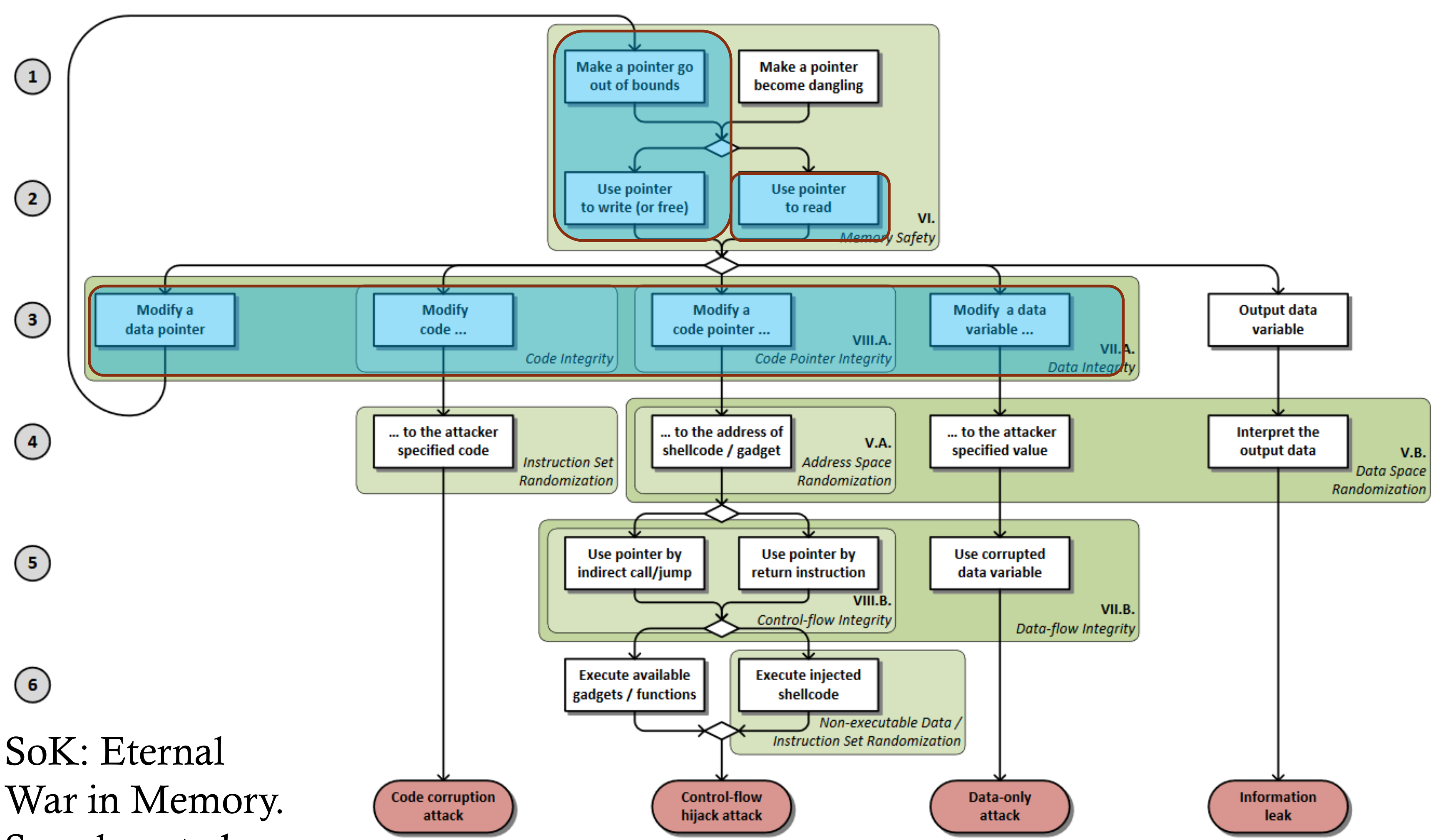
Compatibility is Key

- All loads and stores happen via capabilities, but legacy software does not call the CHERI ISA
- Dedicate special capability registers to instruction fetch and load/store
- All memory accesses are intercepted, and the pointer is used as an offset to the base special register
- Legacy code can call CHERI libraries which offer internal segmentation
- CHERI code can call legacy code, and set coarse limits via special registers
- Not perfect; pointer subtraction unsupported, and code relying on C undefined behavior with significant bounds violations may fail to compile

Poll Question

- What kind of memory safety does CHERI provide?
  - Spatial safety
  - Temporal safety
  - Spatial and Temporal safety
  - None of the above

CHERI and the Attack Model



1
2
3
4
5
6
Make a pointer go out of bounds
Make a pointer become dangling
Use pointer to write (or free)
Use pointer to read
VI. Memory Safety
Modify a data pointer
Modify code ...
Code Integrity
Modify a code pointer ...
VIII.A. Code Pointer Integrity
Modify a data variable ...
VII.A. Data Integrity
Output data variable
... to the attacker specified code
Instruction Set Randomization
... to the address of shellcode / gadget
V.A. Address Space Randomization
... to the attacker specified value
Interpret the output data
V.B. Data Space Randomization
Use pointer by indirect call/jump
Use pointer by return instruction
VIII.B. Control-flow Integrity
Use corrupted data variable
VII.B. Data-flow Integrity
Execute available gadgets / functions
Execute injected shellcode
Non-executable Data / Instruction Set Randomization
Code corruption attack
Control-flow hijack attack
Data-only attack
Information leak

SoK: Eternal War in Memory. Szerekes et al.

Scalability

- Simulation based limit study on pointer intensive Olden benchmark
- Compared against other hardware accelerated capability/fat pointer implementations
- CHERI is competitive or better than the existing works
- 256 bits per capability significantly increases memory overhead compared to 128 bit variant

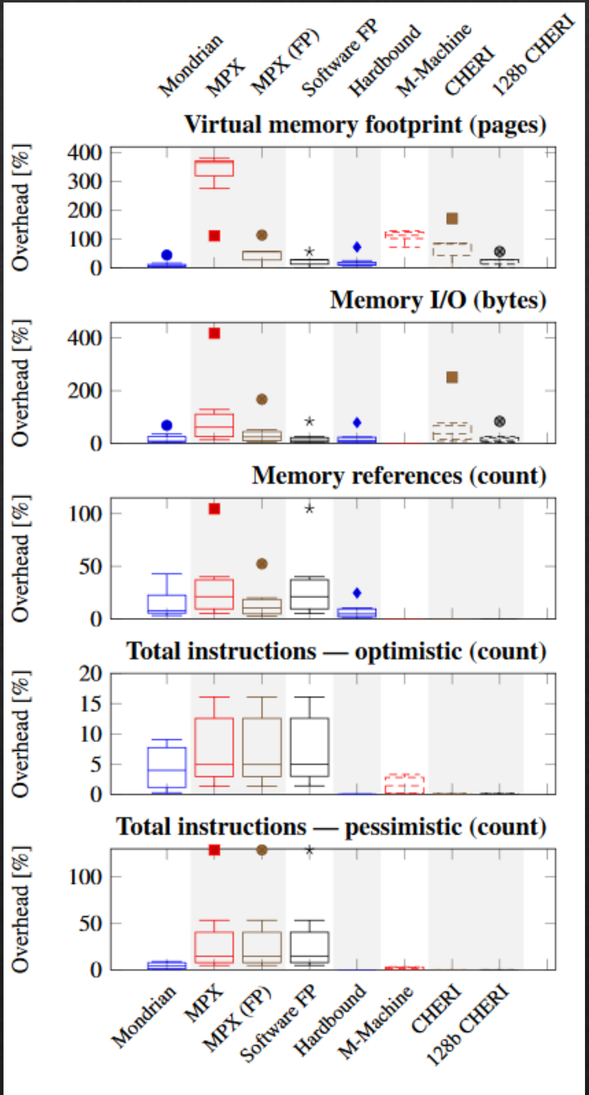


Figure 3: Simulated overheads of Olden benchmarks

Performance Overhead

- Benchmarks performed using a 100 MHz MIPS soft core on FPGA
- Includes CCured software as a reference

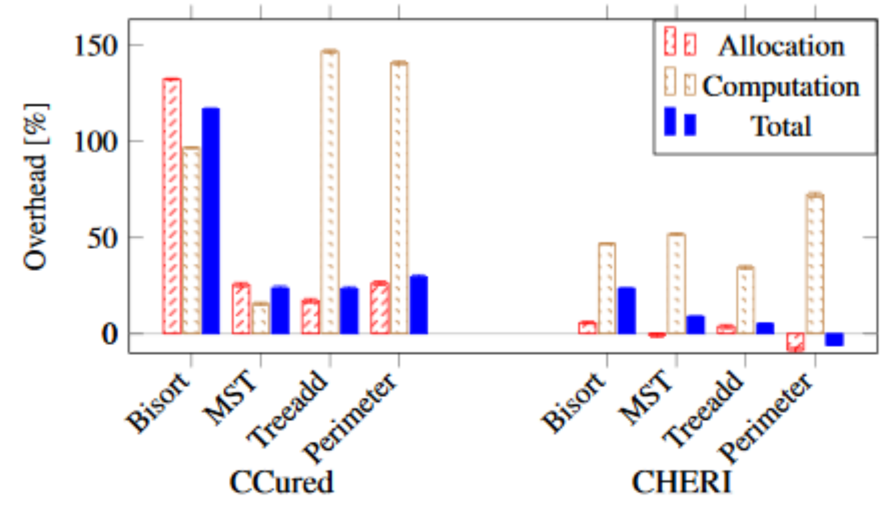


Figure 4: Benchmark results comparing unmodified MIPS code to software and hardware enforcement

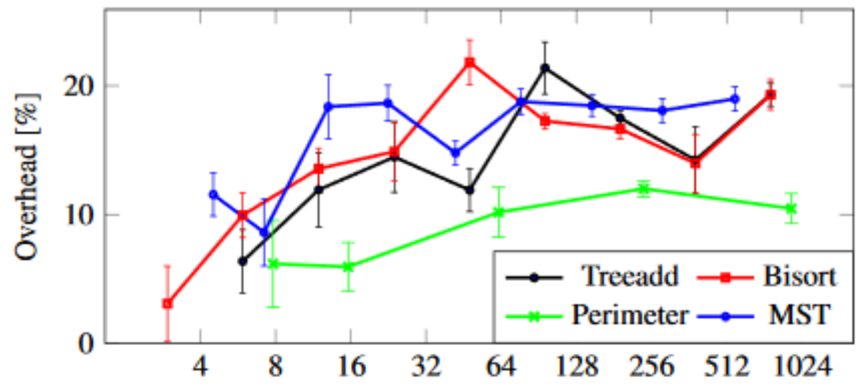


Figure 5: Slowdown for CHERI at different heap sizes (KB)

Potential Cool Uses for CHERI

- Capabilities allow arbitrary segmentation of object permissions – thus W^X permissions can be enforced on JITted code, with a more privileged program having access to the W capability
- Const char buffer[] can be enforced at runtime
  - Set the capability giving access to buffer as read only
- Passing capabilities at function boundaries prevent the confused deputy problem
  - Confused deputy is when user space code tricks more privileged code into doing something it shouldn't
  - By passing a capability, the privileged function is only as capable as the user space program when utilizing the capability

Discussion Questions

- Which is more important for memory safety researchers to protect against contemporary attacks: enforcing *spatial* memory safety, or *temporal* memory safety?
- I've also heard of capability-based OSes - how does this principle in the context of an ISA relate to that? Does CHERI improve when codesigned with the OS/runtime?
- Is CHERI the end of the line in the memory safety space (slight optimizations aside), or do we suspect that there will be larger breakthroughs down the line?
- How much safety does CHERI provide in practice? An unwitting program can still provide access to an overly privileged capability.