Advanced Topics in Computer Architecture

Secure Processors I: CHERI

Prof. Simon W. Moore

UNIVERSITY OF CAMBRIDGE
Computer Science & Technology

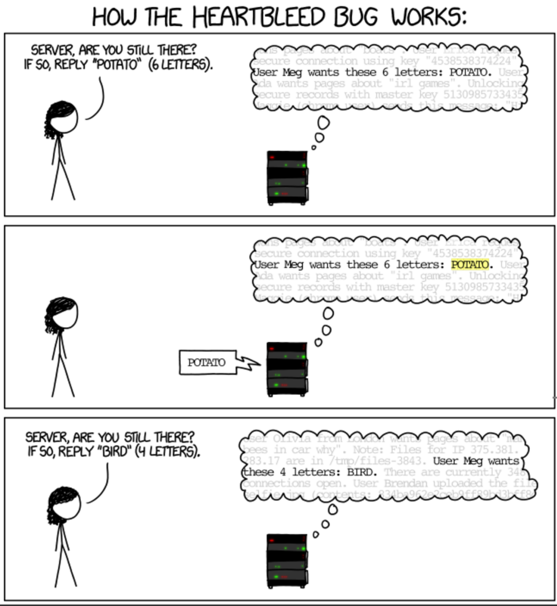
Motivation

- CHERI: secure processor design by Cambridge + SRI International
- Timely:
  - Big UK funding push to commercialise the technology:
    Industry Strategy Challenge Fund: **Digital Security by Design**
    - £70m UK government funding + £116m from industry
  - Started 26th September 2019
  - ARM committed to making the Morello test chip and board platform
- Based on substantial research
  - 120+ engineer/research years of effort
  - >$24m of DARPA funding

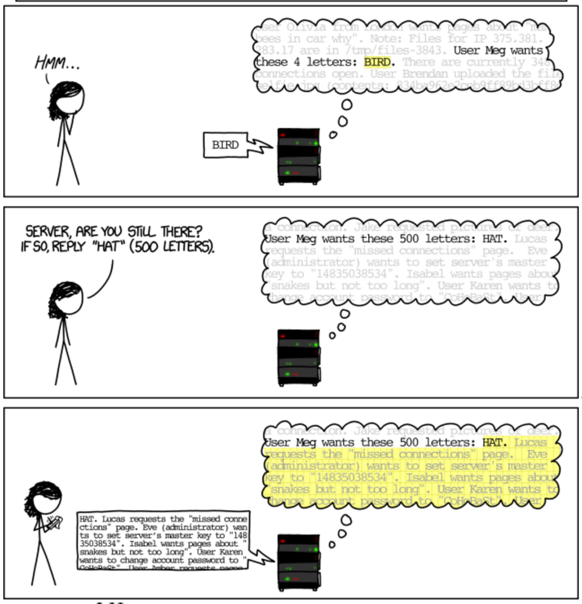
Motivation – The Eternal War in Memory*

- Many security vulnerabilities exploit memory safety violations

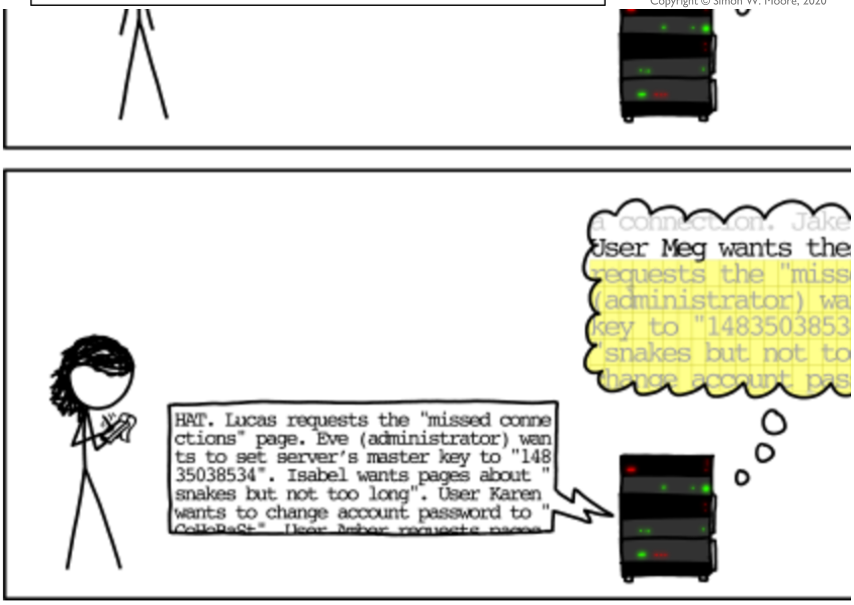
* Title based on Oakland 2013 paper: SoK: Eternal War in Memory, László Szekeres, Mathias Payer, Tao Wei, Dawn Song



source: http://xkcd.com/1354/



source: http://xkcd.com/1354/



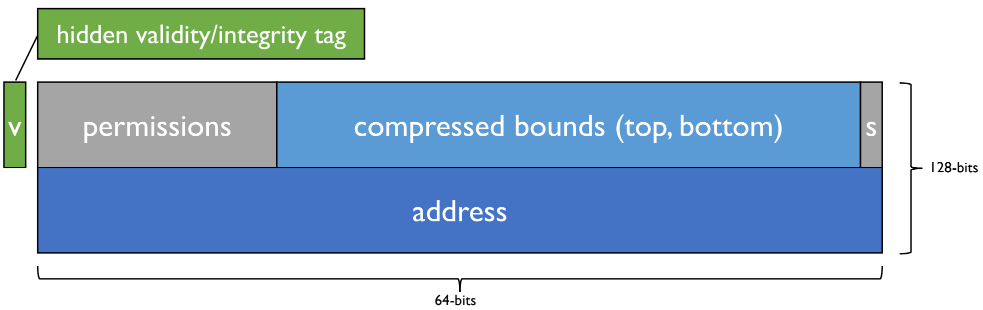
Went wrong? How do we do better?

- Classical answer:
  - The programmer forgot to check the bounds of the data structure being read
  - Fix the vulnerability in hindsight – one line fix:
    if (1+2+payload+16 **>** s->s3->rrec.length) **return** 0;
- Our answer:
  - Preserve bounds information during compilation
  - Use hardware (CHERI processor) to dynamically check bounds with little overhead and guarantee pointer integrity & provenance

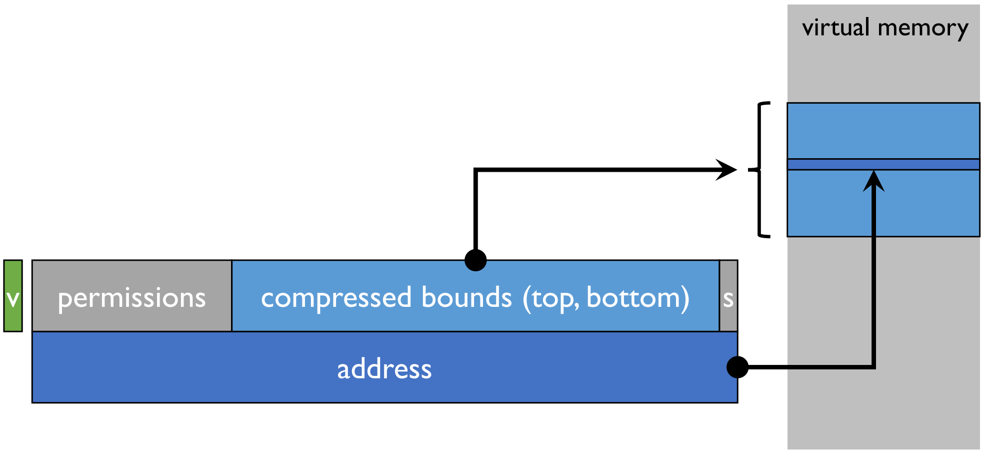
CHERI HARDWARE ARCHITECTURE

A new type – the **Capability**

- CHERI Capability = bounds checked pointer with integrity
- Held in memory and in (new or extended) registers



A new type – the **Capability**



New Instructions

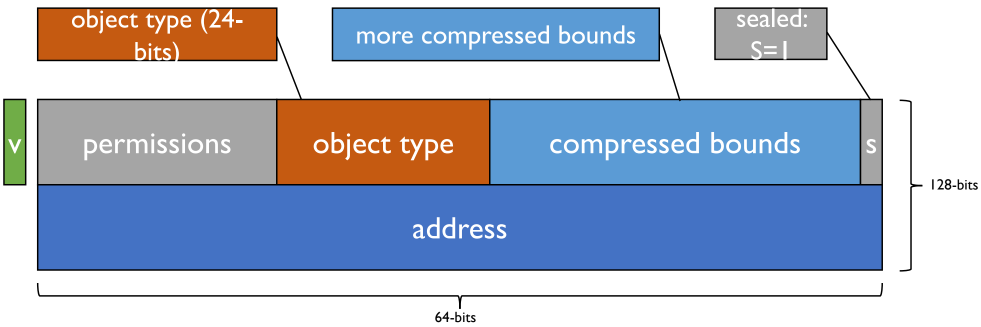
- Memory access
  - Loads and stores via a bounds checked capability
  - Exception if address is out of range
- Guarded manipulation of capabilities
  - Decrease bounds
  - Decrease permissions
  - Adjust the address
  - Extract/test fields

monotonic decrease in rights guaranteed by formally verified hardware

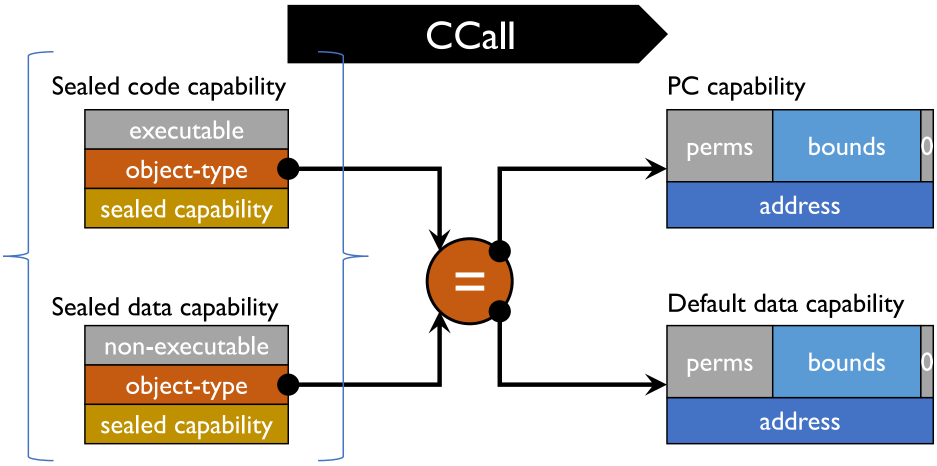
critical property for security

Sealed Capabilities for Compartmentalization

- Sealed capabilities are none dereferencable capabilities
- Have to be unsealed (e.g. inside a compartment) before use



Calling a Compartment



SOFTWARE MODELS

Background to CHERI Software Models

- Machine-level capabilities and instructions provide the building blocks on which new abstract capability software models can be built
- Analogy:
  - Machine-level translation lookaside buffer (TLB) and page table walker enables the OS to represent virtual memory
  - Virtual memory can then be used in different ways to impose new security features, e.g. guard pages

Low-level CHERI software models

More compatible | Safer

Unmodified
All pointers are integers

Hybrid
Annotated and automatically selected pointers are capabilities

Pure-capability
All pointers are capabilities

- **Source and binary compatibility**: **C-language idioms,** multiple **ABIs**
  - **Unmodified code**: Existing code runs without modification
  - **Hybrid code**: E.g., used in return addresses, for annotated data/code pointers, for specific types, stack pointers, etc.

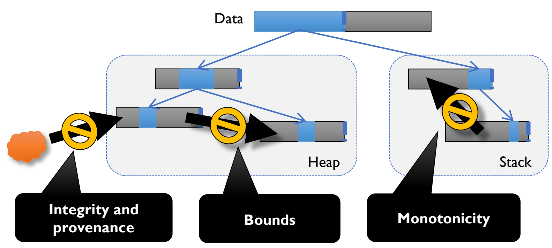
    … But "hybrid" is a spectrum: many different choices for manual and automatic selection of integers vs. capabilities, API and ABI impacts
  - **Pure-capability code**: Ubiquitous data- and data-pointer protection. Not interoperable with legacy code due to changed pointer size.
- **CHERI Clang/LLVM compiler prototype** generates code for all three

Pure Capability Code → Needs CheriABI

- CheriABI
  - Compatibility layer to the OS
  - Allows capabilities to be used in place of pointers
  - A bit like a 32-bit compatibility layer for a 64-bit OS
- Result – we can now recompile large corpuses of C code into a pure capability form with virtually no code changes
- Award winning paper at ASPLOS 2019:
  CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment

Capabilities for Bounds Checking and Integrity

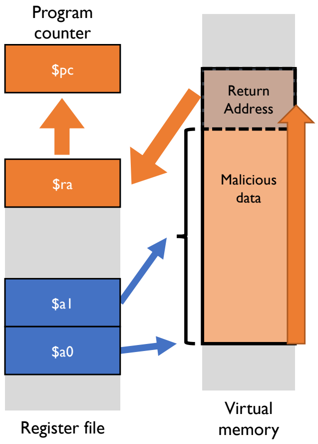
- Pure capability code – all pointers become capabilities
- Compiler + malloc() derive bounds for objects
- Strong pointer provenance and integrity properties (validity tag)



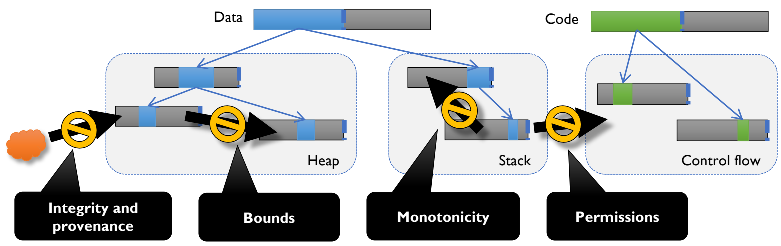
- Mitigates buffer overflow/overread vulnerabilities with no code change!

Capabilities for Control-Flow Robustness

- Capabilities used for return addresses
- Integrity bit mitigates code reuse attacks:
  - ROP – Return Oriented Programming
  - JOP – Jump Oriented Programming
- Much better than current statistical technique ASLR (Address Space Layout Randomisation)



Summary of Capability Protections



Valid userspace pointer set – pointers not generated using derivation rules are not part of the valid provenance tree and will not be dereferenceable

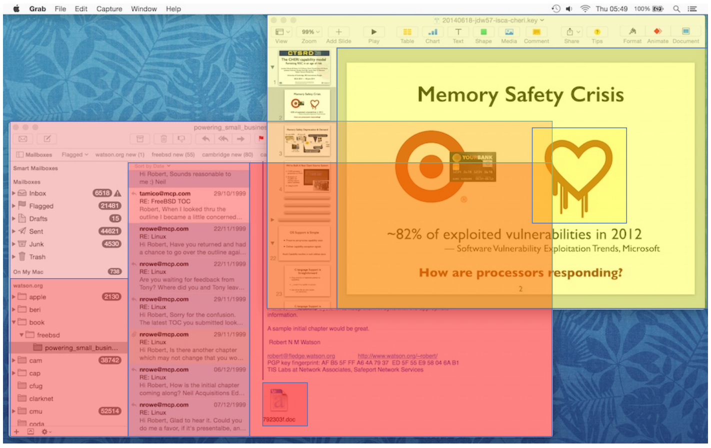
Pointer privilege reduction – capabilities allow pointers to carry specific privileges, which can be minimized with OS, compiler, and linker support

Foundation for higher-level models such as **software compartmentalization**

Compartmentalisation

- Compartment can be described using a sealed pair of capabilities: (program counter, default data capability)
- CCall providing the domain transition
- Allows a number of abstract software models:
  - Library compartmentalisation, e.g. of risky or legacy (non-cap.) code
  - Process-based compartmentalisation within an application can be replaced by much more efficient capability-based protection
    - Same virtual address space (more efficient TLB usage)
    - Very similar software model (easy to port code)

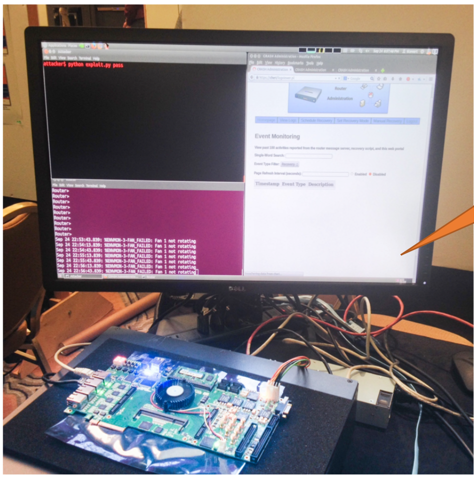
Compartmentalisation Illustrated



RESULTS

First we made it work – Demo tablet platform

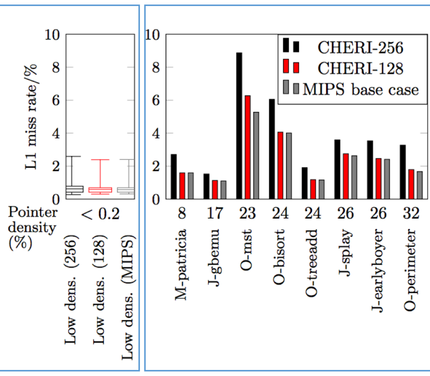
Red Team Evaluation by MIT Lincoln Labs



CHERI mitigates Heartbleed exploit!

Memory-protection performance

Collection of low pointer-density benchmarks from MiBench



High pointer-density benchmarks

(M) MiBench

(O) Olden

(J) Octane JavaScript

L1 cache miss rate for CHERI 256, CHERI-128, and MIPS

CheriABI: A full pure-capability OS userspace

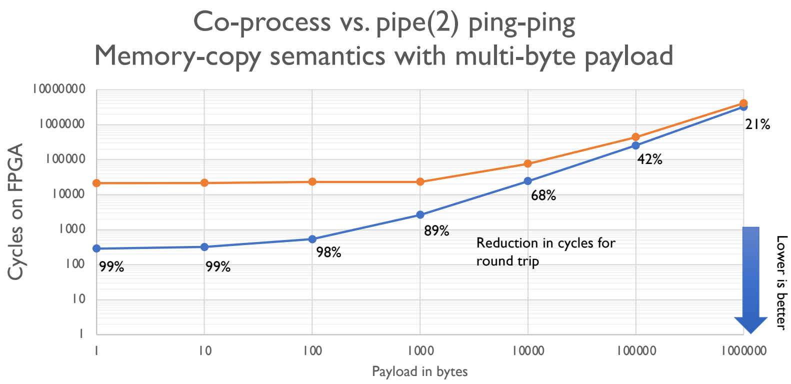
- Complete memory- and pointer-safe FreeBSD C/C++ userspace
  - **System libraries**: crt/csu, libc, zlib, libxml, libssl, …
  - **System tools and daemons**: echo, sh, ls, openssl, ssh, sshd, …
  - **Applications**: PostgreSQL, nginx; bringing up WebKit (C++)
- **Valid provenance**, **minimized privilege** for **pointers, implied VAs**
  - Userspace capabilities originate in **kernel-provided roots**
  - Compiler, allocators, run-time linker, etc., **refine** bounds and perms
- Trading off **privilege minimization**, **monotonicity**, **API conformance**
  - Typically in memory management – realloc(), mmap() + mprotect()

Evaluating memory-protection compatibility

- Prototyping approach:
  - "pure-capability" **C compiler** (Clang/LLVM)
  - **full OS** (FreeBSD) that use capabilities for all explicit or implied userspace pointers
- Observations:
  - **Little or no software modification** (BSD base system + utilities)
  - Small changes to source files for 34 of 824 programs, 28 of 130 libraries
  - Overall: modified ~200 of ~20,000 user-space C files/header

CHERI vs. Process-based Compartmentalization

(Early IPC ping-pong microbenchmark results)



Co-process pipe(2)

The fine print: Cycles include IPC setup, amortised over 10,000 iterations of the IPC loop. Both processes use the pure-capability ABI using 256-bit capabilities. 272-entry TLB, 32K L1 I-Cache, 32K L1 D-Cache, 256K L2 Cache.

CURRENT RESEARCH DIRECTIONS

Generalising CHERI support for many ISAs

- 64-bit MIPS for pragmatic reason: needed a 64-bit RISC ISA in late 2010
- Generic CHERI support doesn't mean that all implementations need to be identical
  - E.g. **portable virtual-memory semantics** and **UNIX process model** despite (quite) different MMUs across architectures
- **Architectural abstraction**: Lift CHERI properties above ISA
- **Architectural localization**: E.g., ISA choices, opcode approaches, exceptions, page tables, … → **architecture-specific specifications**

- CHERI-ARM: Morello spec released by ARM October 2020: https://developer.arm.com/architectures/cpu-architecture/a-profile/morello
- CHERI-RISC-V: ISA specification released by us (Cambridge) in CHERI architecture reference manual V8: https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheri-publications.html

Portability implications for software

- **CHERI Clang/LLVM**
  - Modest pointer/capability abstraction improvements in front-end and IR
  - Adapt target back-ends to teach them about capability code generation
  - Optimize for architecture-specific code generation
  - Optimize for available microarchitectures
- **CheriBSD** (CHERI support in FreeBSD)
  - More clear machine-independent / machine-independent split
  - Shift to hybrid capability C in the kernel to improve machine independence
  - Various MD kernel updates: boot code, exceptions, PMAP, …
  - Clean up APIs, header separation, architecture abstraction
  - Various userspace updates: rtld, libcheri, CRT/CSU, …

Conclusions

- CHERI Provides the hardware with more semantic knowledge of what the programmer intended
  - Toward the **principle of intentionality**
- Efficient **pointer integrity** and **bounds checking**
  - Eliminates buffer overflow/over-read attacks (finally!)
- Provide scalable, efficient compartmentalisation
  - Allows the **principle of least privilege** to be exploited to **mitigate known and unknown attacks**
  - Large performance improvement over process-based compartmentalisation
- **Working with industry to bring the technology to market**
- Thanks to sponsors: DARPA, ARM, Google, EPSRC, HEIF, Isaac Newton Trust, Thales E-Security, HP Labs, Huawei

https://www.cl.cam.ac.uk/research/security/ctsrd/

Simon.Moore@cl.cam.ac.uk
Computer Science & Technology

Further reading

- **Background:** *An Introduction to CHERI*, Technical Report UCAM-CL-TR-941, Computer Laboratory, September 2019.
  https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-941.pdf
- *Efficient Tagged Memory*, ICCD 2017
  https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201711-iccd2017-efficient-tags.pdf
- *CHERIvoke: Characterising Pointer Revocation using CHERI Capabilities for Temporal Memory Safety*, MICRO 2019
  https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201910micro-cheri-temporal-safety.pdf
- *CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization,* SSP 2015
  https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201505-ssp2015-cheri-compartment.pdf

- Further optional reading:
  - CHERI Concentrate: Practical Compressed Capabilities, IEEE Transactions on Computers 2019
    https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/2019tc-cheri-concentrate.pdf
  - Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8), Technical Report UCAM-CL-TR-927
    https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-951.html
  - CHERI publications list:
    https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheri-publications.html