Advanced Topics in Computer Architecture

Secure Processors I: CHERI

Prof. Simon W. Moore

UNIVERSITY OF CAMBRIDGE
Computer Science & Technology

Background

- 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 making the Morello test chip and board platform to be shipped to partners Q1 2022
- Based on substantial research
  - 120+ engineer/research years of effort
  - >$24m of DARPA funding

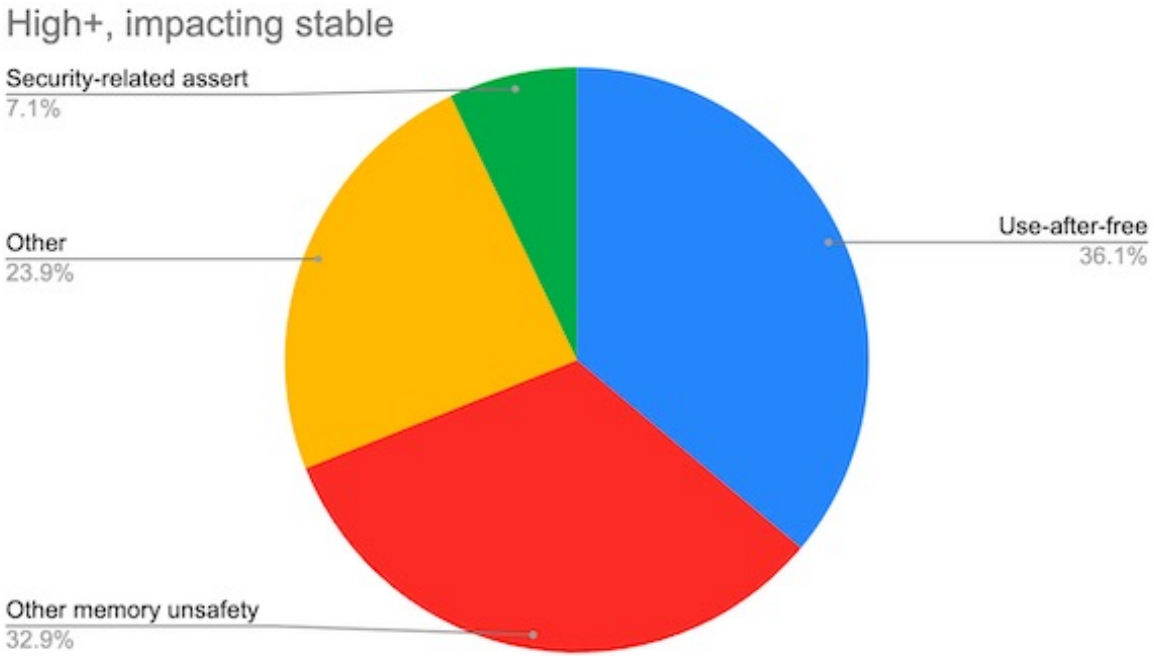
Motivation – Memory Safety

- Matt Miller (MS Response Center) @ BlueHat 2019:
  - From 2006 to 2018, year after year, 70% MSFT CVEs are memory safety bugs.
  - First place: spatial safety
    - Addressed directly by CHERI
  - Second place: use after free
    - Temporal memory safety is made efficient by exploiting CHERI capability validity tags to quickly and precisely find pointers during revocation

Motivation – Chromium Browser Safety

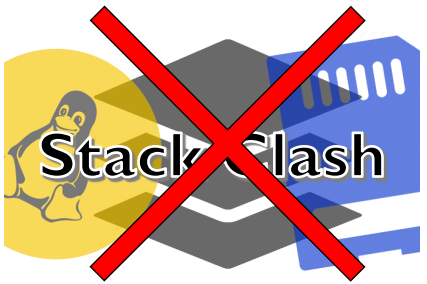
"70% of our serious security bugs are memory safety problems"

www.chromium.org/Home/chromium-security/memory-safety

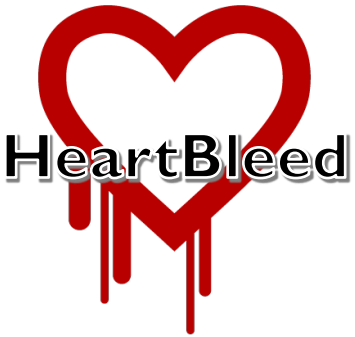


Motivation – The Eternal War in Memory*

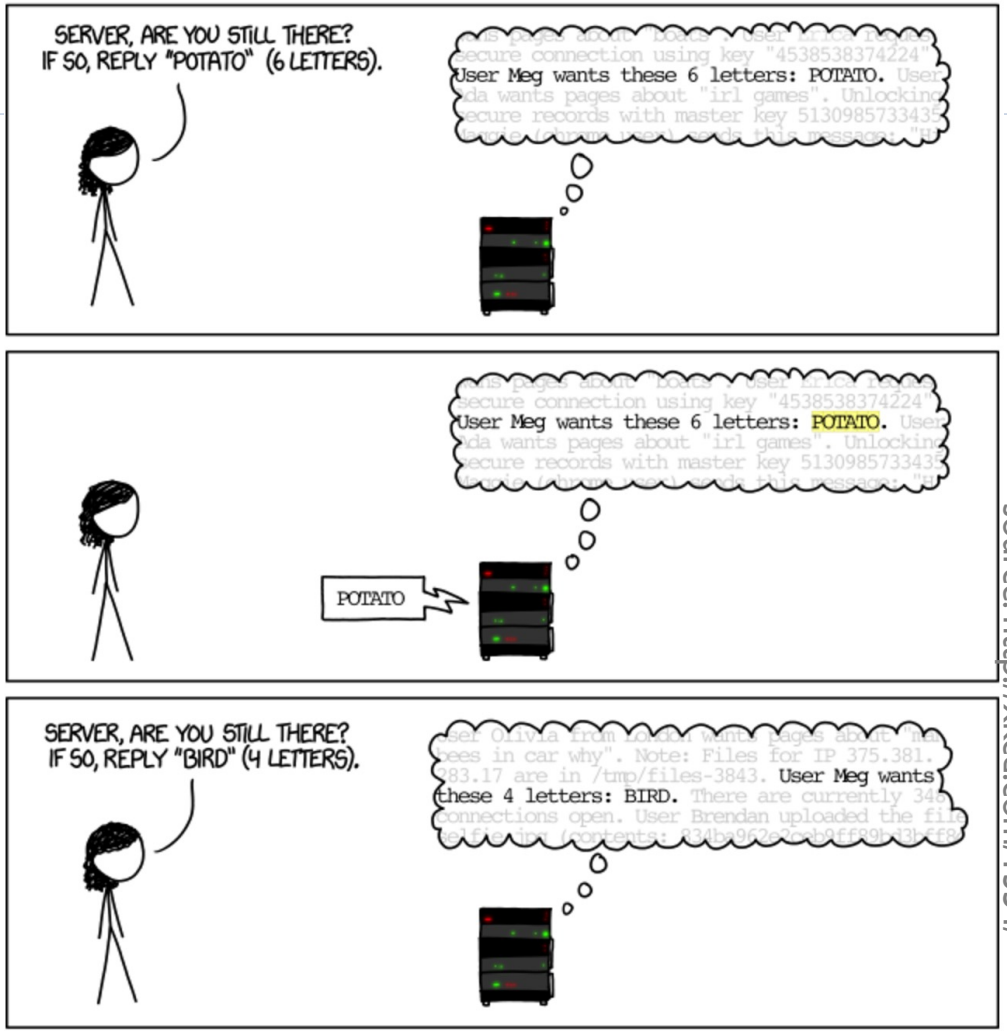
- Many security vulnerabilities exploit memory safety violations



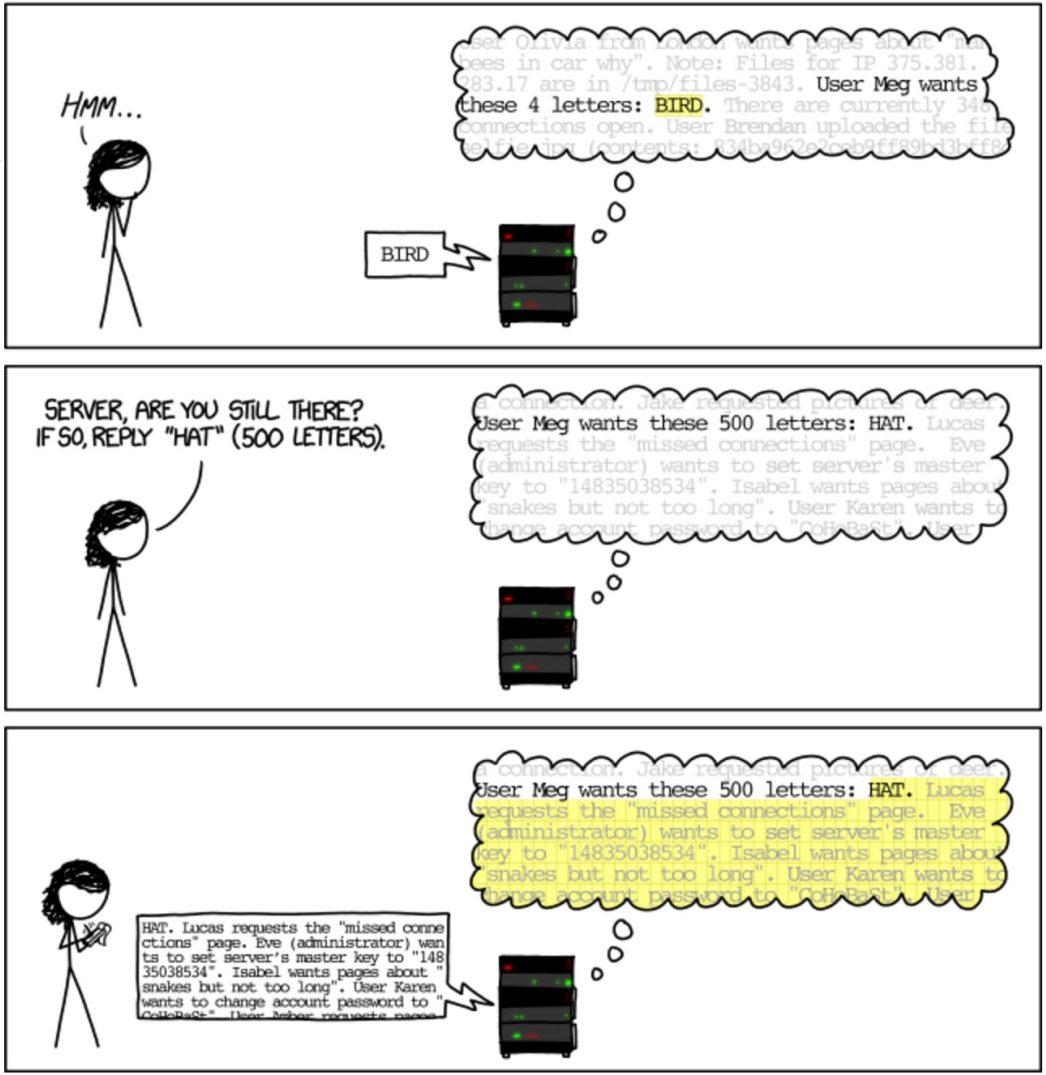
* Title based on Oakland 2013 paper: SoK: Eternal War in Memory, László Szekeres, Mathias Payer, Tao Wei, Dawn Song



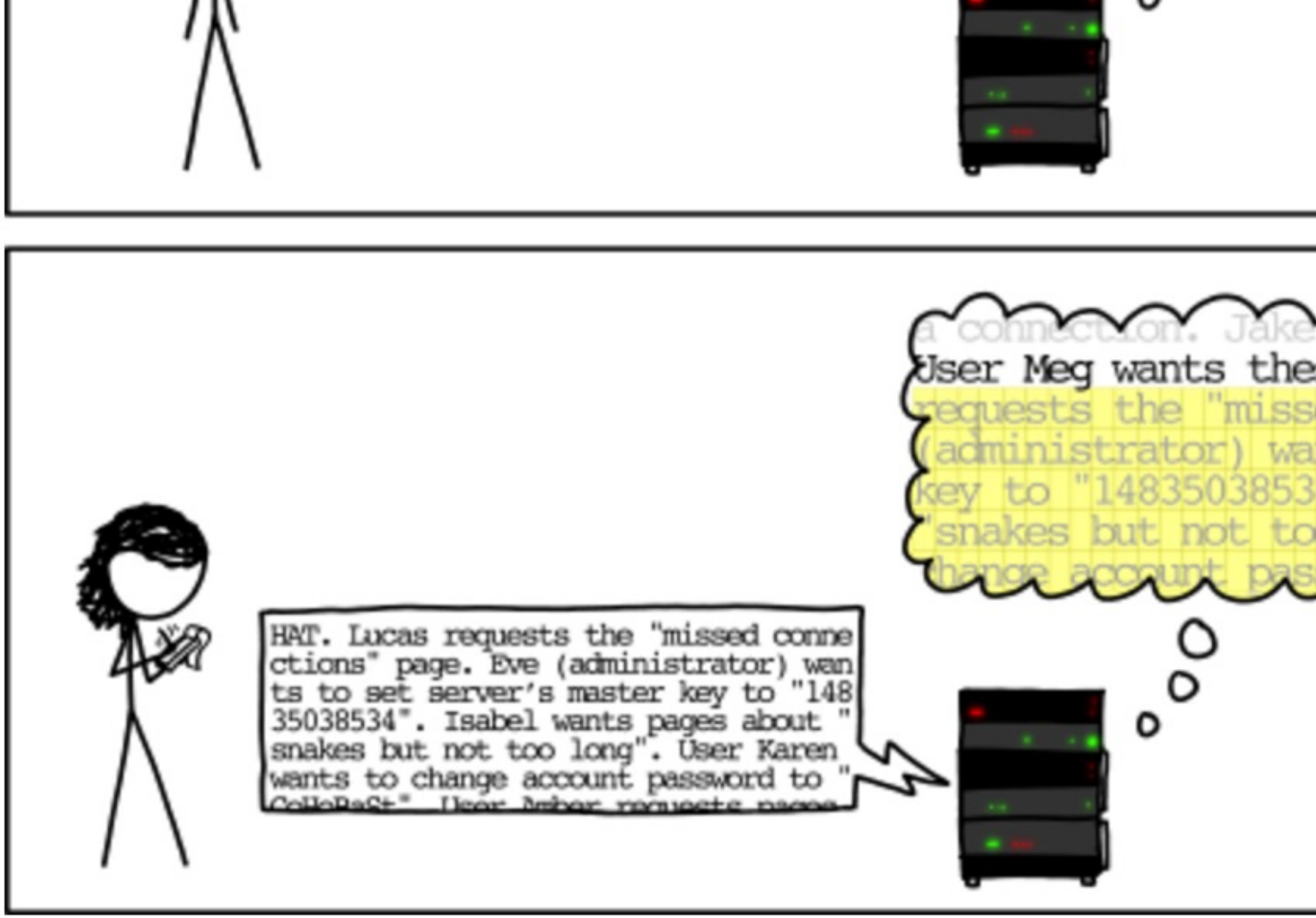
HOW THE HEARTBLEED BUG WORKS:



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



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


a connection. Jake
User Meg wants the
requests the "miss
(administrator) wa
key to "1483503853
snakes but not to
hange account pas
HAT. Lucas requests the "missed conne
ctions" page. Eve (administrator) wan
ts to set server's master key to "148
35038534". Isabel wants pages about "
snakes but not too long". User Karen
wants to change account password to "

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 Approach to Memory Safety

- The **principle of intentional use**
  - Ensure that software runs the way the programmer intended, not the way the attacker tricked it
  - Approach:
    - Guaranteed pointer integrity & provenance; efficient dynamic bounds checking
    - Instructions always accept capability operands, and never look them up automatically (unlike an MMU)
- The **principle of least privilege**
  - Reduce the attack surface using software compartmentalization
    - This mitigates known and unknown exploits
  - Approach: highly scalable and efficient compartmentalization
- Robust **deterministic protection**, not probabilistic debugging measures

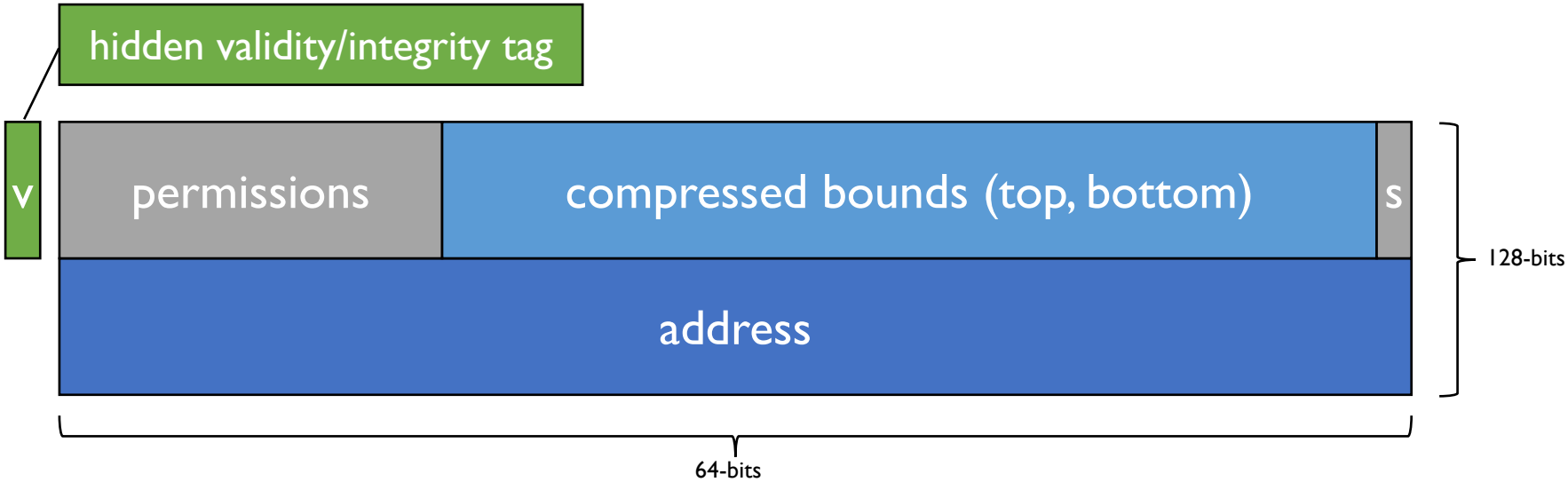
Preserve RISC philosophy

- "No" to:
  - Microcode
  - New table lookups
  - Exotic external memory
  - Extended pipeline
  - Reduction in clock frequency
  - Reduced addressing modes
  - Crypto (not needed here)
- "Yes" to:
  - Low-level hardware functionality on which many software structures can be built
  - Compiler friendly
  - Get the compiler, linker and run-time system to do much of the work, not the ISA+decoder
  - Keep it as simple as possible!

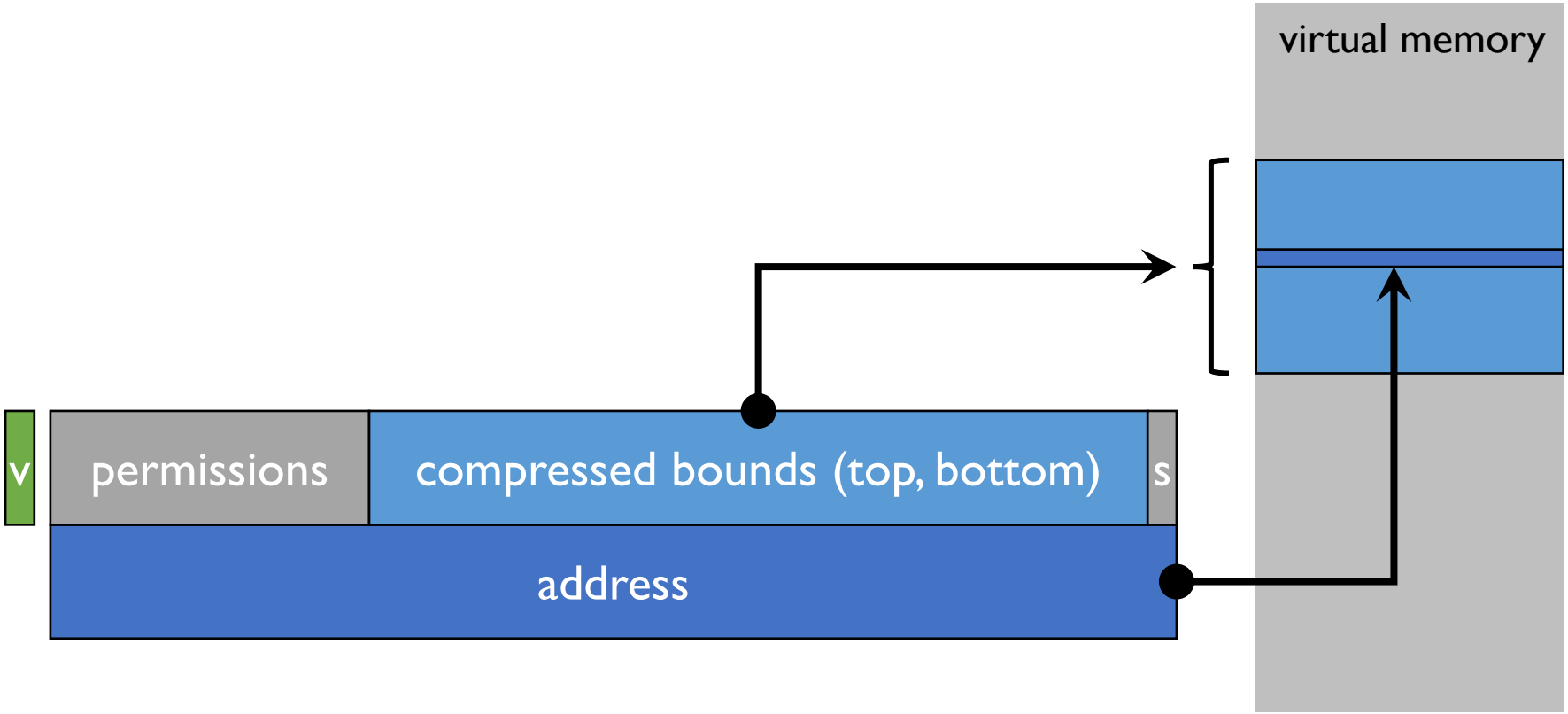
CHERI HARDWARE ARCHITECTURE

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

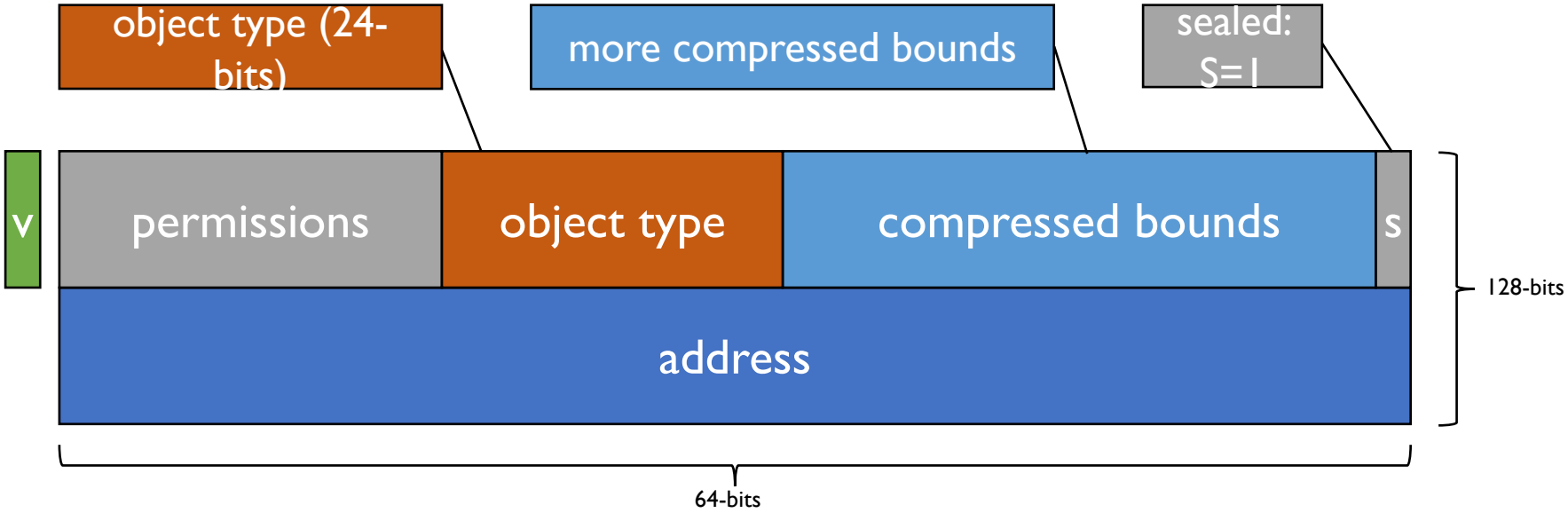
- Memory access
  - Loads and stores via a bounds checked capability
  - Exception if address is out of range
- Guarded manipulation of capabilities
  - Decrease bounds
  - Decrease permissions
  - Adjust the address
  - Extract/test fields

monotonic decrease in rights guaranteed by formally verified hardware

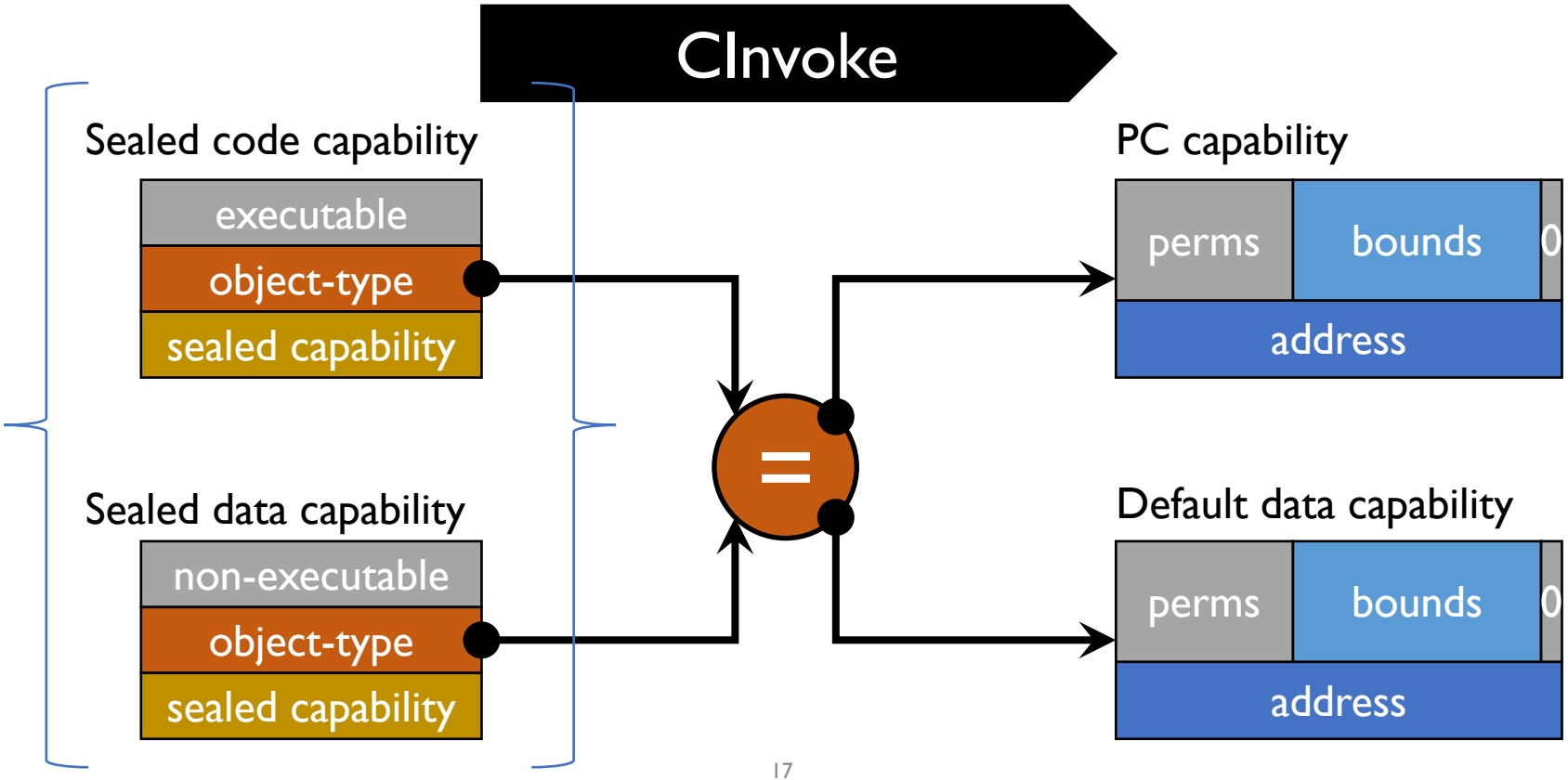
critical property for security

Sealed Capabilities for Compartmentalization

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



Calling a Compartment



CHERI Software Models

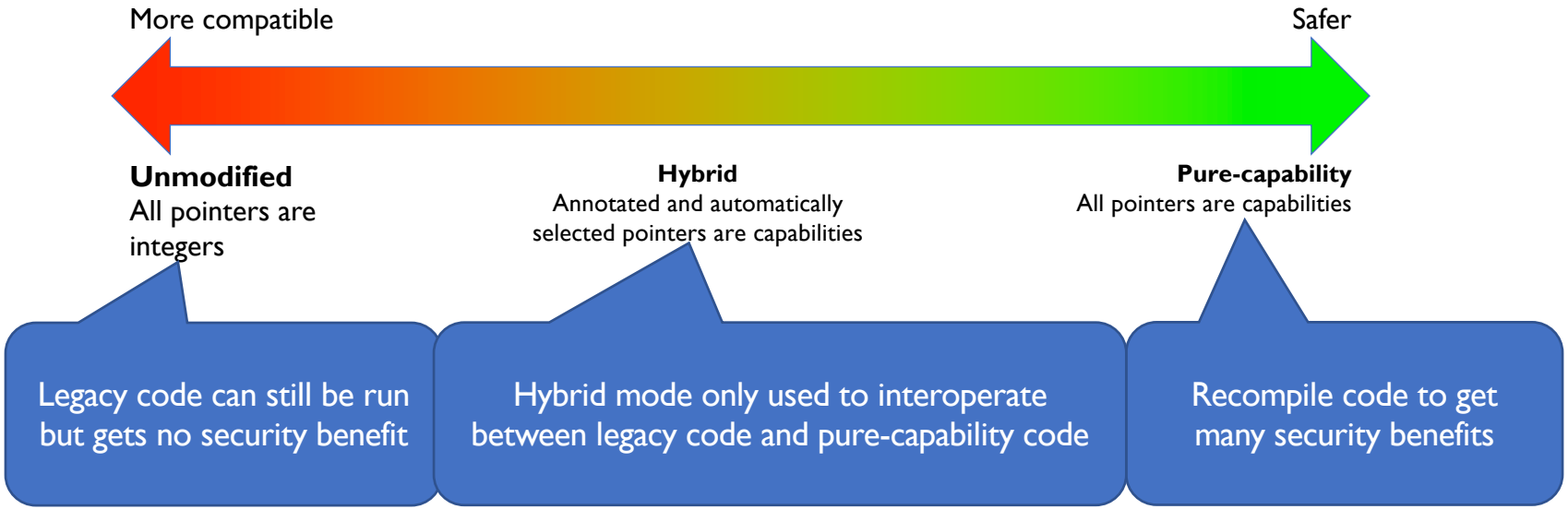
UNIVERSITY OF CAMBRIDGE

Computer Science & Technology

Background to CHERI Software Models

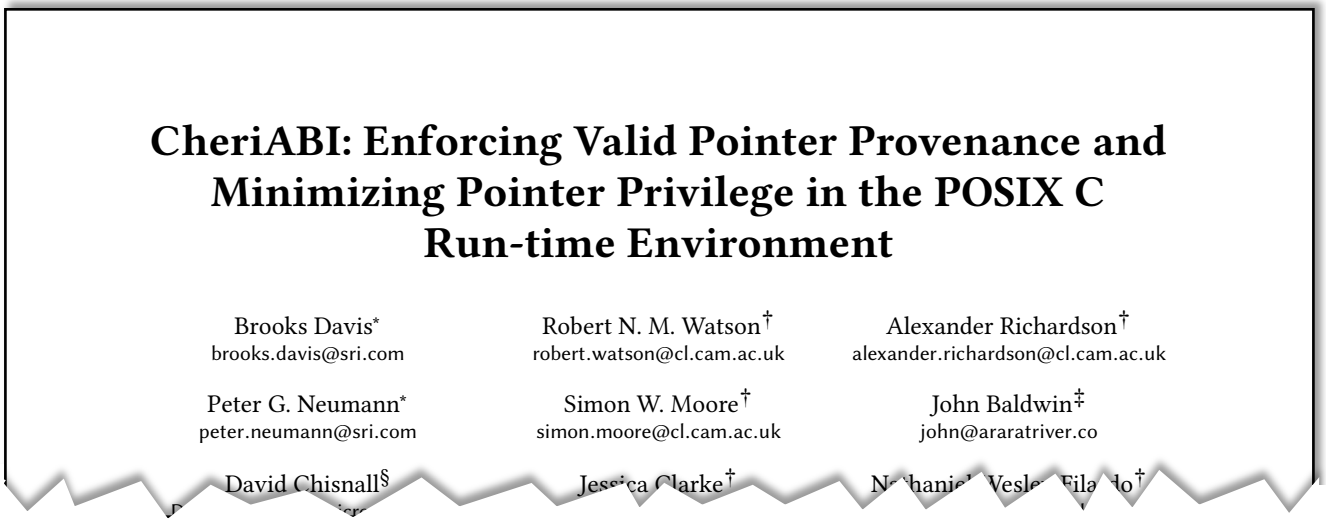
- Machine-level capabilities and instructions provide the building blocks on which new abstract capability software models can be built
- Analogy:
  - Machine-level translation lookaside buffer (TLB) and page table walker enables the OS to represent virtual memory
  - Virtual memory can then be used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

Low-level CHERI software models



- Our **CHERI Clang/LLVM compiler** generates code for all three models

Pure Capability Code → Needs CheriABI



CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment

Brooks Davis* brooks.davis@sri.com

Robert N. M. Watson† robert.watson@cl.cam.ac.uk

Alexander Richardson† alexander.richardson@cl.cam.ac.uk

Peter G. Neumann* peter.neumann@sri.com

Simon W. Moore† simon.moore@cl.cam.ac.uk

John Baldwin‡ john@araratriver.co

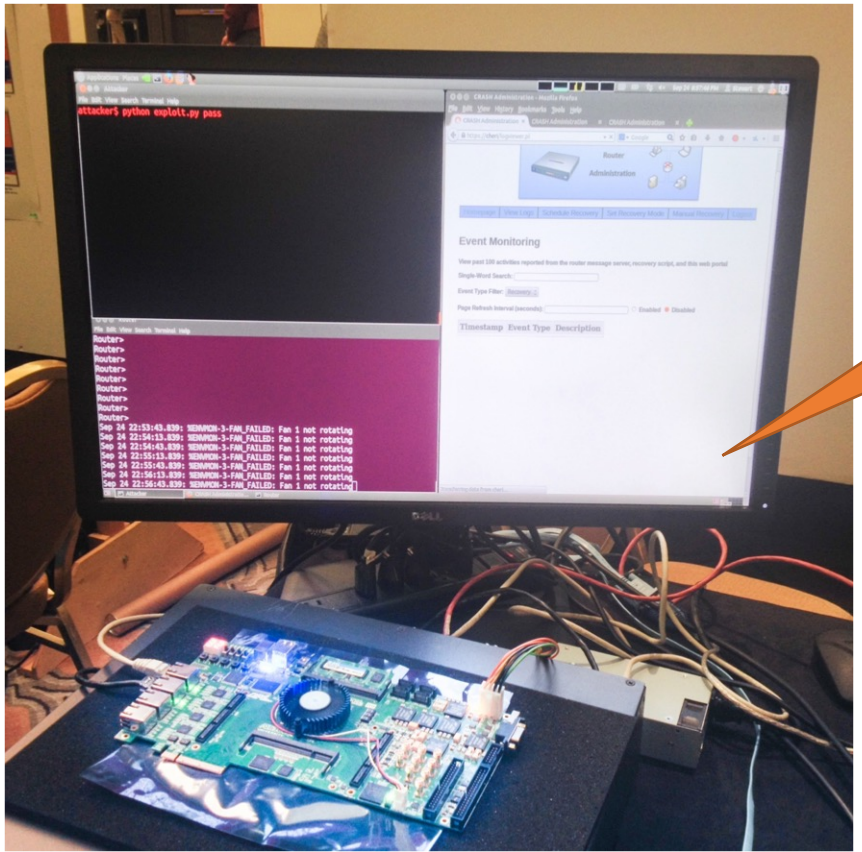
David Chisnall§

Jessica Clarke†

Nathaniel Vesle Filardo†

- Received best paper award at ASPLOS, April 2019
- Complete pure-capability UNIX OS userspace with spatial memory safety
  - Usable for daily development tasks
  - Almost vast majority of FreeBSD tests pass
  - Management interfaces (e.g. ioctl), debugging, etc., work
  - Large, real-world applications have been ported: PostgreSQL and WebKit

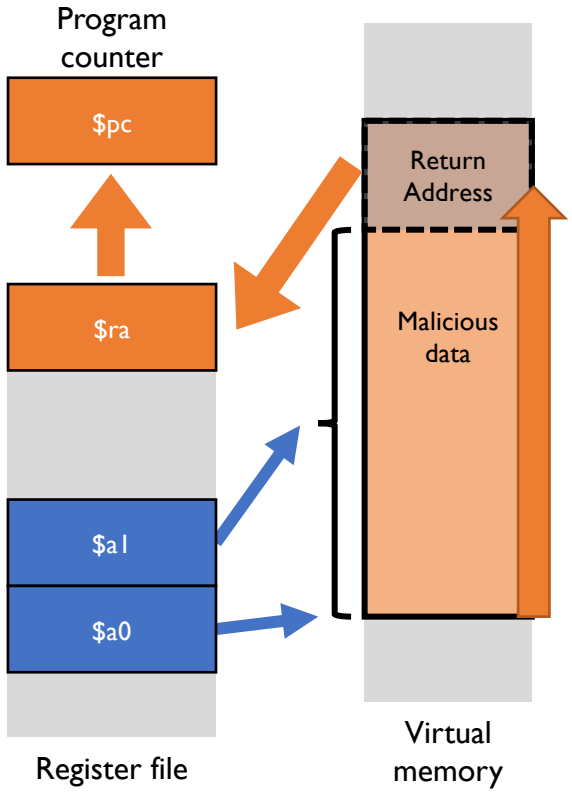
Red Team Evaluation by MIT Lincoln Labs



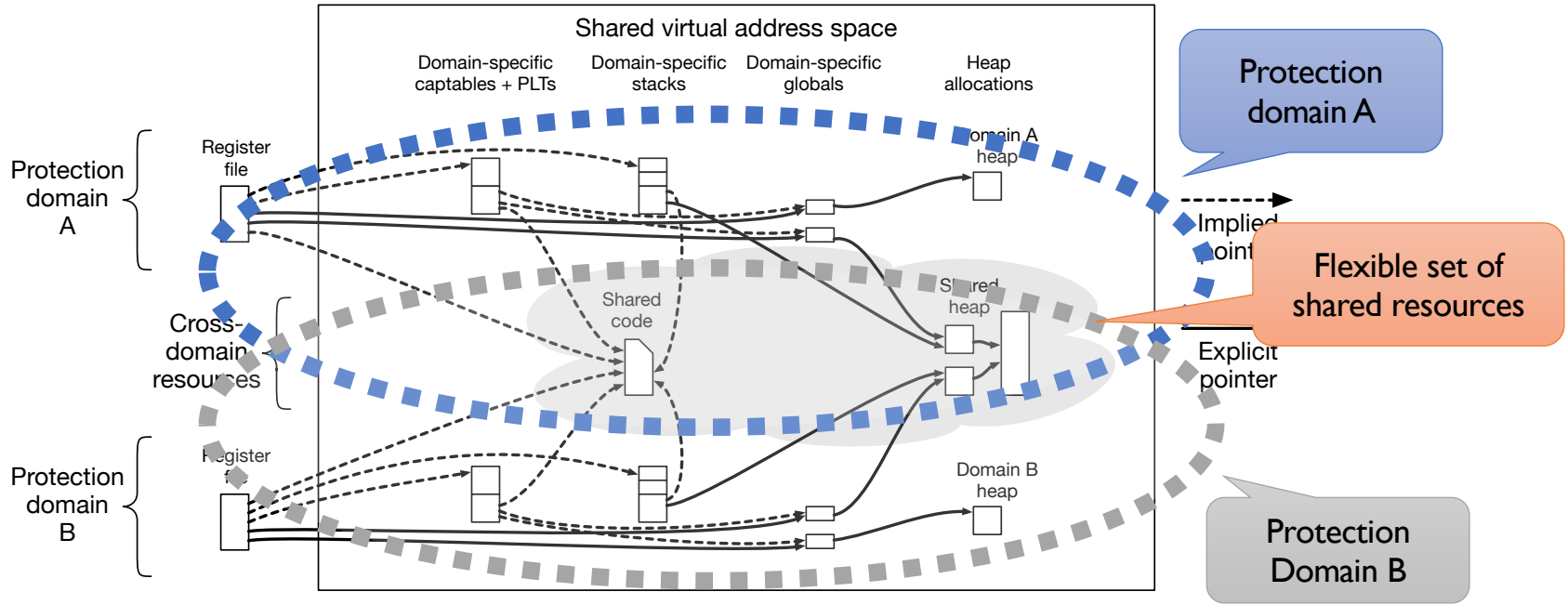
CHERI mitigates Heartbleed exploit!

Capabilities for Control-Flow Robustness

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



Example of CHERI-based compartmentalization



- Isolated compartments can be created using closed graphs of capabilities, combined with a constrained non-monotonic domain-transition mechanism

Temporal Memory Safety

- E.g. use-after-free vulnerabilities are common
- CHERI hardware directly implements strong spatial safety and enables efficient temporal safety:
  - Tags and object bounds held by capabilities makes software-based temporal safety efficient via revocation
  - Hardware optimisations makes software scanning for tags (i.e. scanning for object references) efficient

Papers on CHERI Revocation

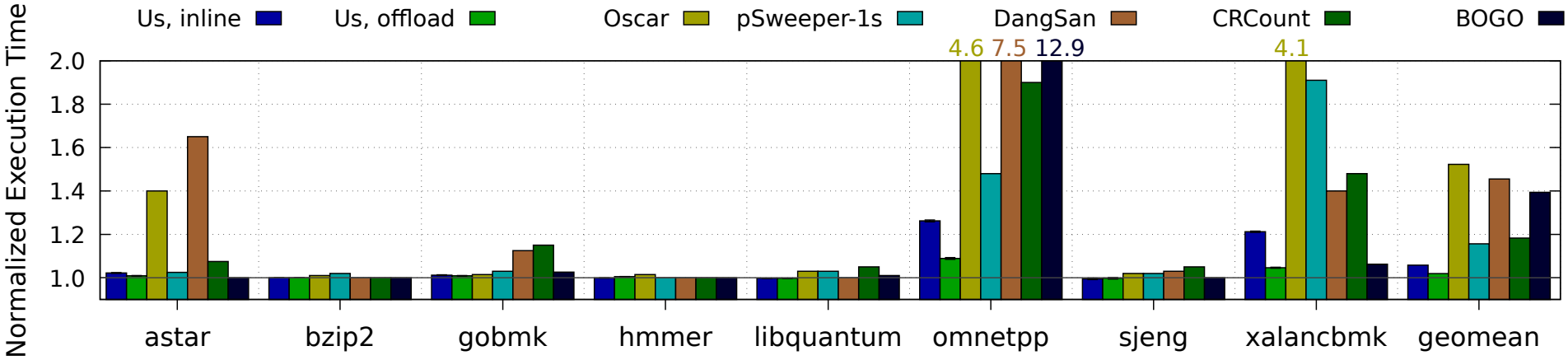
CHERIvoke: Characterising Pointer Revocation using CHERI Capabilities for Temporal Memory Safety

MICRO'52, 2019

Cornucopia: Temporal Safety for CHERI Heaps

IEEE Security and Privacy (Oakland), 2020

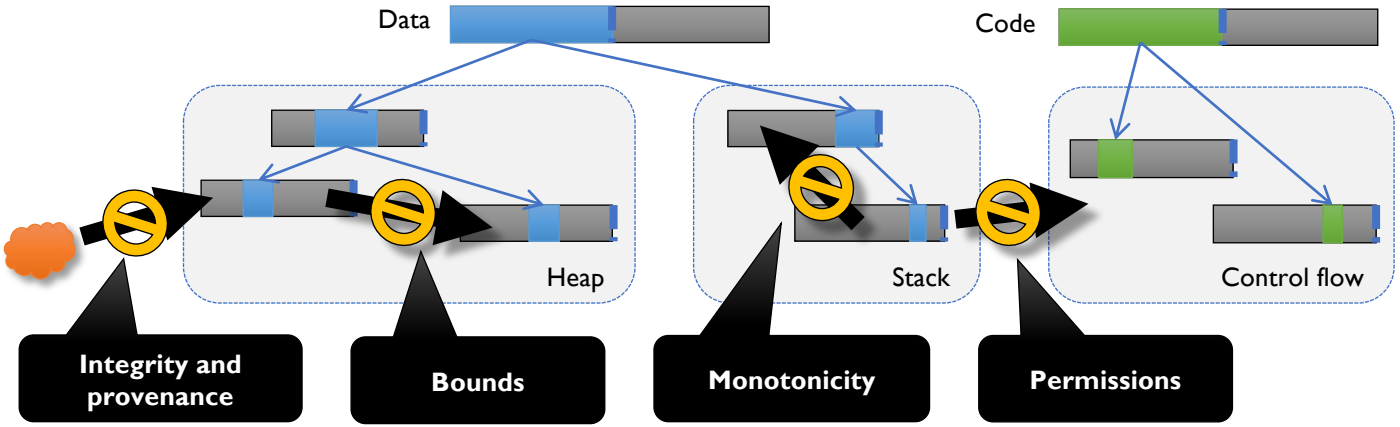
Cornucopia: State-of-the-Art Runtime Overheads



- CHERI-compatible SPEC CPU2006 benchmarks
  - *Without* second thread, geomean is 5.8% (worst 26.2%)
  - *With* second thread and core, geomean **1.9%** (worst 8.9%)
  - When offloading, pause times 10-20% of single-thread sweep typical
- Applications unmodified beyond existing small patches for CheriABI compatibility

Research is ongoing to further optimize temporal safety!

Summary of Capability Protections



- **Low-level capability hardware is a foundation for software models**
  - Guarantee a valid userspace pointer set with pointer privilege reduction
  - Highly efficient compartmentalization
  - Efficient, deterministic temporal safety

Building CHERI-RISC-V

UNIVERSITY OF CAMBRIDGE

First we made an FPGA-based hardware tablet

Several Processors Implemented

- Early CHERI-MIPS:
  https://github.com/CTSRD-CHERI/cheri-cpu
- Current CHERI-RISC-V cores:
  - Piccolo 32b microcontroller:
    https://github.com/CTSRD-CHERI/Piccolo
  - Flute 64b/32b scalar core:
    https://github.com/CTSRD-CHERI/Flute
  - Toooba 64b out-of-order core based on MIT Riscy-OOO core
    https://github.com/CTSRD-CHERI/Toooba

Specification and Test

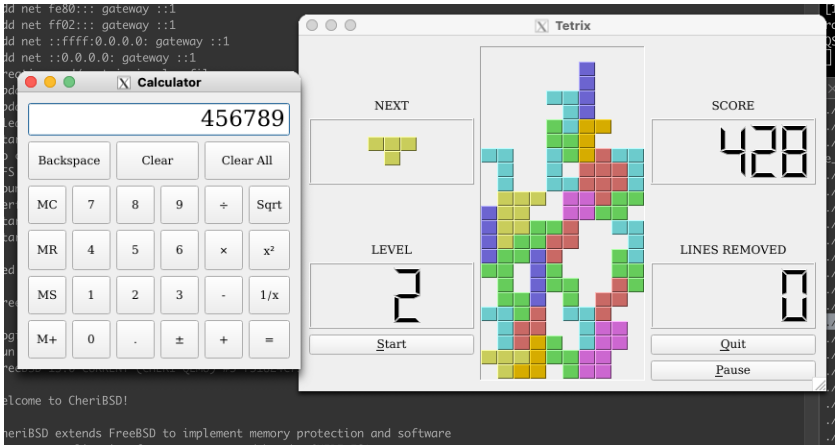
- SAIL-based executable formal model of RISC-V
  https://github.com/riscv/sail-riscv
  - Originally work from Cambridge but now the official RISC-V formal specification
- SAIL-based CHERI-RISC-V spec:
  https://github.com/CTSRD-CHERI/sail-cheri-riscv
- TestRIG for directed-random testing with test case shrinkage
  https://github.com/CTSRD-CHERI/TestRIG

Toolchain and OS support

- **C compiler** (Clang/LLVM) supporting capabilities
- **Full OS** (FreeBSD, FreeRTOS) 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
    - Mainly localized to low-level run-time support

User space and demo applications

- 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, WebKit (C++)
  - **GUI:** X11 client libraries, Qt…



Current Research Directions

- Compartmentalisation
  - Demonstrably much more efficient than process-based compartmentalisation
  - But need new software models
  - Need to ensure that CHERI compartments are robust even against transient executions attacks (see next lecture)
- Temporal memory safety
  - Microarchitectural and run-time optimisations
- CHERI for the whole SoC
  - CHERI for accelerators
- Toward exascale: CHERI for partitioned global address spaces
- CHERI everywhere: CHERI for x86
- Refining CHERI and the Morello architecture to bring it into the main stream Arm v9 ISA

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
- *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
- *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

- Further optional reading:
  - Cornucopia: Temporal Safety for CHERI Heaps, IEEE Symposium on Security and Privacy, 2020
    https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/2020oakland-cornucopia.pdf
  - 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-951
    https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-951.pdf
  - CHERI publications list:
    https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheri-publications.html