CTSRD

CRASH-worthy Trustworthy Systems Research and Development

Introducing CHERI

Capability Hardware Enhanced RISC Instructions

Robert N. M. Watson

Simon W. Moore, Peter G. Neumann, Jonathan Woodruff, Jonathan Anderson, Hadrien Barral, Ruslan Bukin, David Chisnall, Nirav Dave, Brooks Davis, Lawrence Esswood, Khilan Gudka, Alexandre Joannou, Chris Kitching, Ben Laurie, A. Theo Markettos, Alan Mujumdar, Steven J. Murdoch, Robert Norton, Philip Paeps, Alex Richardson, Michael Roe, Colin Rothwell, Hassen Saidi, Stacey Son, Munraj Vadera, Hongyan Xia, and Bjoern Zeeb

University of Cambridge, SRI International

CHERI Microkernel Workshop – 23 April 2016

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 ('CTSRD') and FA8750-11-C-0249 ('MRC2'). The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

SRI International

University of Cambridge

DARPA CRASH

If you could revise the fundamental principles of computer-system design to improve security…

…what would you change?

Principle of least privilege

Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job.

Saltzer 1974 - CACM 17(7)
Saltzer and Schroeder 1975 - Proc. IEEE 63(9)
Needham 1972 - AFIPS 41(1)

Principle of least privilege (2)

- **Access control**
  - Minimize privileges held by users (and hence their processes) in accordance to policy
- **Fault tolerance**
  - Limit the impact of software/hardware faults
- **Vulnerability and Trojan mitigation**
  - Constrain rights gained as a result of software supply-chain compromise (Karger IEEE S&P 1987)
  - Motivation for *sandboxing*, *privilege separation*, and *software compartmentalization* used to mitigate vulnerabilities in contemporary applications

What is CHERI?

- CHERI is a **capability architecture** supporting fine-grained, pointer-based memory protection in hardware:
  - pointer integrity (e.g., no return-address corruption)
  - bounds checking (e.g., prevent buffer overflows)
  - permission checking (e.g., W^X for pointers)
  - compartmentalization (efficient, fine-grained sandboxing within address spaces)
- Requires modest additions to the:
  - Instruction set, processor, compiler, OS
- Aims to mitigate known and unknown classes of memory – and other – exploit techniques

CHERI design goals

- Target C/C++-language software TCBs
- Strong in-address-space protection
  - Robust memory safety for data pointers
  - CFI-like protection for code pointers
  - Fine-grained, scalable compartmentalisation
- Compose naturally with MMU-based designs
  - Reference- rather than address-centric protection
  - Supplement paging-based protection as page sizes grow
- RISC prototyping approach, applied to 64-bit MIPS
  - Underlying model could be applied to many ISAs

New fundamental type: the **capability**

- capability = tag + virtual address + bounds + permissions
  - all in 128(+1) bits
- Spatial protection for pointers to data and code
- Managed by software
  - Directed by OS, compiler, application
- Enforced by hardware
  - Held in registers or memory
  - Strong integrity guarantees, ISA-level enforcement
  - Higher performance, strong atomicity guarantees

Hardware Guarantees

- **Capabilities** can only be used (dereferenced) if:
  - **Valid** – derived from past pointers through only permitted manipulation and without in-memory corruption
  - **In bounds** – no overflow/underflow of allocations or compartments
  - **Permitted** – authorized for {load, store, execute, …}
  - **Unsealed** – no bypass of encapsulation for software-defined objects
- **Guarded manipulation**: monotonic rights decrease
  - capabilities delegation narrows permitted access
  - enforced provenance tree: pointers come from pointers

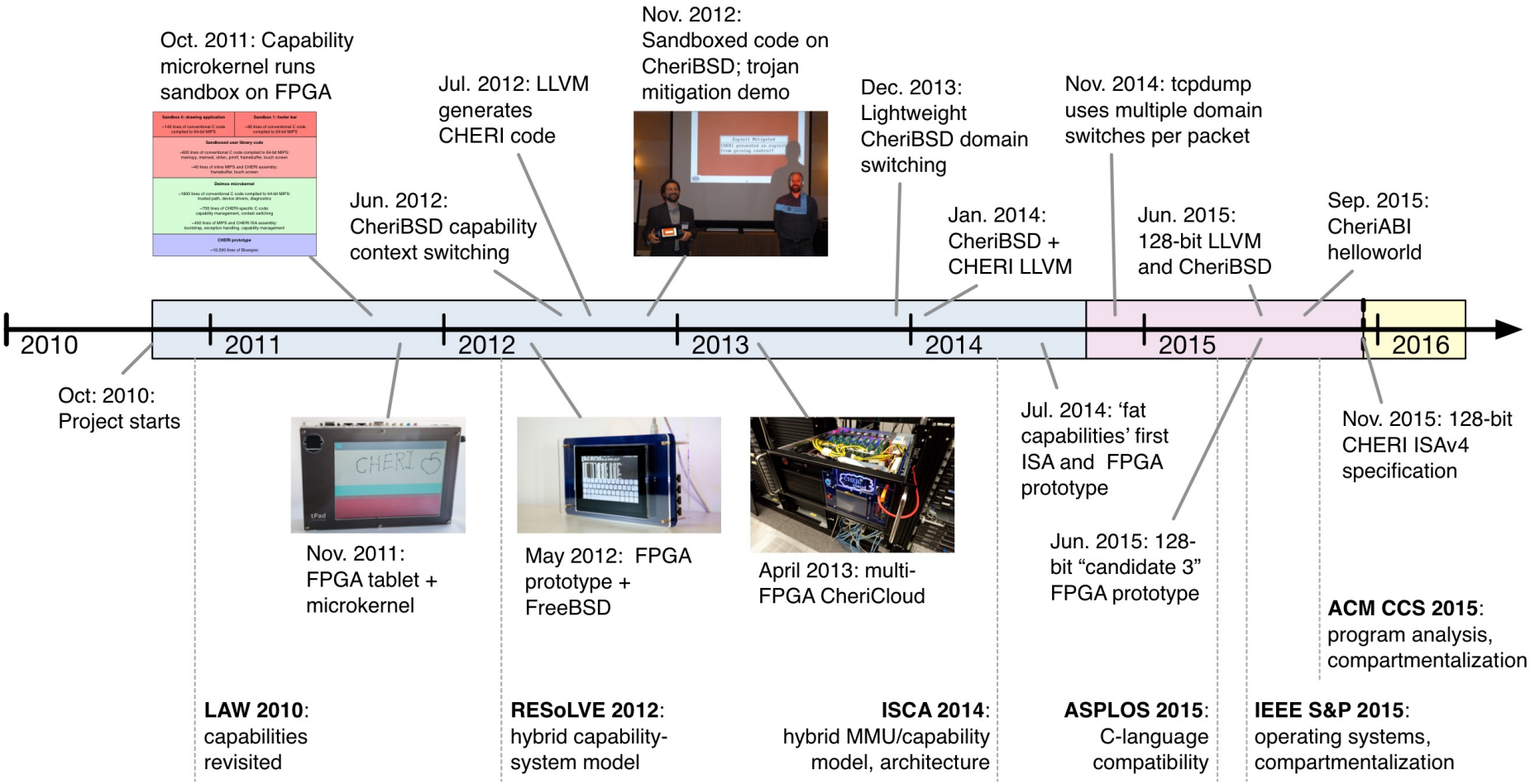
Capabilities are for the compiler

- Capabilities refer to virtual addresses:
  - Implement explicit pointers and implied addresses
  - OS/run-time linker provide initial capabilities
  - OS + compiler + runtime + software determine how capabilities are refined, delegated, utilized
  - E.g., to heap/stack allocations, memory mappings
- MMU still implements virtual addresses:
  - Page tables retained (pretty much) as-is
  - Still supports processes, full-system virtualization
- Each address space is a "virtual capability machine"

Status

- Prototype implementations:
  - Single- and multicore CHERI pipelines on FPGA
  - Qemu fast ISA simulator; L3 formal model of ISA
- Language support:
  - C using LLVM; C++ in progress
  - Most C code can be recompiled to benefit from memory protection
  - Compartmentalization requires programmer intervention
- OS support:
  - FreeBSD + MIPS hybrid capability process environment
  - FreeBSD + CheriABI pure-capability process environment
  - Just starting on CheriOS microkernel prototype (CMW tomorrow)
- It works and is efficient…

Q&A


Oct. 2011: Capability microkernel runs sandbox on FPGA
Jul. 2012: LLVM generates CHERI code
Nov. 2012: Sandboxed code on CheriBSD; trojan mitigation demo
Dec. 2013: Lightweight CheriBSD domain switching
Nov. 2014: tcpdump uses multiple domain switches per packet
Jun. 2012: CheriBSD capability context switching
Jan. 2014: CheriBSD + CHERI LLVM
Jun. 2015: 128-bit LLVM and CheriBSD
Sep. 2015: CheriABI helloworld
2010
2011
2012
2013
2014
2015
2016
Oct: 2010: Project starts
Nov. 2011: FPGA tablet + microkernel
May 2012: FPGA prototype + FreeBSD
April 2013: multi-FPGA CheriCloud
Jul. 2014: 'fat capabilities' first ISA and FPGA prototype
Jun. 2015: 128-bit "candidate 3" FPGA prototype
Nov. 2015: 128-bit CHERI ISAv4 specification
LAW 2010: capabilities revisited
RESoLVE 2012: hybrid capability-system model
ISCA 2014: hybrid MMU/capability model, architecture
ASPLOS 2015: C-language compatibility
IEEE S&P 2015: operating systems, compartmentalization
ACM CCS 2015: program analysis, compartmentalization