CRASH-worthy Trustworthy Systems R&D (CTSRD): CHERI System Architecture and Research

Final Report A013 Draft, SRI Project 19800

Peter G. Neumann (SRI International, Menlo Park CA, USA)
Robert N. M. Watson (University of Cambridge, UK)
Simon W. Moore (University of Cambridge, UK)

December 13, 2019

Abstract

This is the Final Technical Report for **CRASH-worthy Trustworthy Systems Research and Development (CTSRD)**, SRI Project 19800, which has been sponsored from 24 September 2010 to 30 September 2019 by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237 ("CTSRD") as part of the DARPA I2O **Clean-slate Resilient Adaptable and Secure Hosts (CRASH)** program. CTSRD technical work ended on 30 September 2019, although the project continued adminstratively through the subsequent review period. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the Department of Defense or the U.S. Government.

CTSRD has been a joint project between the SRI International Computer Science Laboratory and the University of Cambridge Department of Computer Science and Technology.

CTSRD is a highly innovative DARPA I2O research project concerned with designing and prototyping new computer hardware-software systems with significantly greater potential trustworthiness than anything currently available, whether commercial or open-sourced. The last few years of the project were devoted primarily to enhancing technology transfer via a close collaboration with Arm Limited. In the final week of the project, this collaboration has been publicly announced, with plans for an experimental CHERI-ARM demonstrator processor, SoC, and evaluation board, *Morello*, available from late 2021.

The work described herein includes a few earlier contributions by team members who were also funded in part by FA8750-11-C-0249 (**MRC**^{2}) for the DARPA Mission-oriented Resilient Clouds (MRC) research program, but whose work was relevant to CTSRD and CHERI. More recently, some contributors to CTSRD also received some funding support from contract HR0011-18-C-0016 (**ECATS**) under the MTO SSITH program, and/or from contract FA8650-18-C-7809 (**CIFV**) from MTO. In addition to the above mentioned DARPA funding, additional support to our CTSRD work was received from St John's College Cambridge, the Google SOAAP Focused Research Award, a Google Chrome University Research Program Award, the RCUK's Horizon Digital Economy Research Hub Grant (EP/G065802/1), the EPSRC REMS Programme Grant (EP/K008528/1), the EPSRC Impact Acceleration Account (EP/K503757/1), the ERC Advanced Grant ELVER (789108), the Isaac Newton Trust, the UK Higher Education Innovation Fund (HEIF), Thales E-Security, Microsoft Research Cambridge, Arm Limited, Google DeepMind, and HP Enterprise.

READING THIS REPORT ONLINE: The Table of Contents sections (in red or blue, depending on the two issues with identical content), section numbers (in red), and citation numbers (in green) are all clickable in the pdf text. In Preview or Acrobat Pro, Clicking on 'View' and then 'Go To' and then 'Previous View' should get you back to where you were before clicking on the hyperlink. (That seems to work everywhere, except for getting to the Glossary or the Index from the Table of Contents pages.)

READING THIS REPORT printed on paper obviously does not have the benefits of hypertexting. The report is evidently optimized for online viewers.

Contents

1 Summary

This CTSRD final technical report (FTR) is intended to be a guide for the reader interested in pursuing our work in further depth. It summarizes our work during nine years since the inception of the DARPA I2O CRASH program and its CTSRD project, which began on 24 September 2010. 'CRASH' refers to the DARPA CRASH program, Clean-slate Resilient and Adaptable Secure Hosts. 'CTSRD' is the joint SRI-Cambridge CRASH-worthy Trustworthy Systems R&D project. Greater detail can be found in our published papers itemized in Appendix A, and in reports whose abstracts are included in Appendix B. Appendix C gives a list of PhDs that have resulted or are in progress relating to CTSRD. Appendix D acknowledges the contributions of many people who have helped make CTSRD successful. Finally, Appendix E lists symbols, abbreviations, and acronyms used here. The report's bibliography provides references for additional relevant items (in particular, some that are not generated by the CTSRD team) that not noted in Appendices A or B.

The CTSRD project has performed research and development pursuing clean-slate highly trustworthy computer systems with potential use in a wide variety of critical applications that could require high-end security, reliability, human safety, and other stringent requirements for predictable behavior. We have developed specifications for a family of highly trustworthy computer systems, the CHERI (Capability Hardware Enhanced RISC Instructions) hardware-software architecture (Section 5.2) and its CHERI hardware Instruction-Set Architecture (ISA, Section 5.1), with hardware implementations in Field-Programmable Gate Arrays (FPGAs) and simulation. We have also developed operating systems, programming language extensions, compilers, and applications that demonstrate the efficacy, performance, and enormous potential of the hardware for a collection of illustrative uses and applications (Section 5.4). The CHERI Programmer's Guide (Section 5.5 is intended to provide considerable guidance to the development of software using the CHERI hardware.

CTSRD initiated a deeper study of how to more securely integrate microcontrollers and active devices into total-system trustworthiness, where today's hardware has many undocumented and proprietary microdevices with direct memory access to the main processors (Section 5.7). That work is continuing under the companion DARPA MTO SSITH ECATS project (Extending the CHERI instruction-set Architecture for Trustworthiness in SSITH).

Early on, we developed useful tools to aid in design and development, such as TESLA (Section 5.8) and SOAAP (Section 5.9). Furthermore, CTSRD has engaged in efforts at technology transfer that are elaborated in Section 5.10, which are also continuing under the ECATS project. Future versions of CTSRD reports are likely to be refined and maintained under other projects.

The CHERI ISAs are formally specified. CTSRD has also significantly advanced the applicability of formal methods to rigorous analysis of our specifications, and clearly suggests the feasibility of formal analysis of our low-level software such as microkernels (Section 5.11). That work is now continuing under the companion DARPA MTO CHERI Instruction-set architecture Formal Verification (CIFV) project.

2 Introduction

The CTSRD project was conceived within the DARPA I2O CRASH program as an effort to innovate with relatively few constraints on what might be achieved in new research and realistic development, with a keen eye toward real-world use and extensive opportunities for transitioning our technology into practice. CTSRD is a joint project between the SRI International Computer Science Laboratory (CSL) and the University of Cambridge Computer Science and Technology Department (CST). For simplicity in this report, SRI International's CSL is referred to simply as "SRI", and the University of Cambridge CST Department is referred to simply as "Cambridge".

The CHERI hardware instruction-set architectures that we have designed provide the foundations for a family of computer systems and networks capable of potentially unprecedented trustworthiness. The ISAs are capability-based, where a capability is in essence a nonforgeable and nonbypassable pointer to computer virtual memory, or in some cases complex objects: the capability mechanism thus supports hardware-protected access to dynamically and statically program-defined object capabilities as well as virtual memory. Each capability includes nonalterable access permissions that are specifically relevant to what is permitted to be accessed.

The CHERI ISA's fine-grained access controls enable both coarse-grained and fine-grained compartmentalization – for example, the former among applications, and the latter within individual applications and within operating systems. CHERI's hybrid architecture enables potentially untrustworthy compartmentalized applications (either object code or CHERI-compiled source code) to coexist without being able to compromise highly trustworthy system software, other applications, and other users.

In addition to new conceptual hardware, the CTSRD project also has pursued operating systems and compilers that could take significant advantage of the hardware, tools to aid in developing systems and applications, and more. Although a primary goal was to eliminate many of the common programming errors associated with C and C++, the results have considerably transcended that goal. The CTSRD results described in this FTR also spawned additional projects and funding that have enabled us to go much farther than what could be done under CTSRD alone.

This final report summarizes the goals and results of our CHERI-related work since 2010, primarily with respect to CTSRD while also noting related funding wherever appropriate. All references cited here with square brackets are enumerated in the bibliography.

3 Methods, Assumptions, and Procedures

- **Methods** Proactive adherence to a set of fundamental principles (See [15] for a preliminary discussion of how the CHERI total-system architecture adheres to these principles), modular abstraction with strong encapsulation (e.g., hiding of internal state information), least privilege, formal specifications of hardware instruction sets, formal analysis of ISAs, a total-system hybrid architecture that enables predictably depend-

able coexistence of potentially untrustworthy legacy software with extremely trustworthy software, selection of skilled personnel with extraordinarily diverse backgrounds encompassing security, hardware, software, networks, system engineering, and awareness of the literature. That underlying well-established principles have been followed pervasively in the design and development of the CHERI ISAs and operating systems is perhaps the most important factor in the CHERI architectures. In particular, the principles apply to the ISAs and compilers' ability to constructively use any CHERI hardware that adheres to the principled ISAs. Our explicit intent that CHERI should be especially useful for C and C++ programs (which are typically fraught with security problems) has also been a strong motivational force, and has resulted in the CHERI-LLVM compiler providing fundamentally stronger security.

- **Assumptions** We have made relatively few oversimplifying assumptions regarding the basic hardware instruction-set architecture. That is, the potential applicability of our work is extremely broad, encompassing desktops, laptops, mobile devices, servers, multicore hardware, embedded systems, and a wide range of potential applications. Our threat model includes a broad range of would-be attacks, insider misuse, and programming errors, and even hardware weaknesses such as inadequately protected direct memory access (DMA) failures. Although we have sublimated certain concerns relating to covert timing channels and other out-of-band signaling, somewhat surprisingly the CHERI architecture can provide some remediation for speculative execution (e.g., out-of-order anticipatory pre-fetching) and DMA misuse. As a consequence, there are essentially no limitations on software.

- **Procedures** Formal specifications of ISAs, formal generation of test cases, formally based testing and rigorous formal analysis of hardware and software (now under the companion DARPA MTO CIFV project), regression testing, real-time detection of riskful events (TESLA), a tool for identifying and analyzing riskful dependencies that is particularly useful for establishing appropriate compartmentalizations (SOAAP), extensive use of valuable principles throughout design, specification, implementation, and analysis of the ability to remediate specific types of vulnerabilities (which is being evaluated more stringently in the ongoing ECATS project). Throughout the CHERI development, we have carried out our hardware-software-model co-design: rigorous specification supported by proof, elaboration of architectural decisions into the microarchitecture on FGPA, and exploration of the software implications through the compiler, operating system, and applications. We performed an extensive, multi-year design cycle to iterate from a conventional RISC instruction set to the CHERI ISA, measuring and improving dimensions that include software performance, software compatibility, microarchitectural viability, and security.

4 Site Descriptions

SRI International (the prime contractor, a not-for-profit research institute) is headquartered in Menlo Park, California. The Computer Science Laboratory is centered mostly in Menlo Park, although it has staff in its Washington DC office in Rosslyn VA, and its New York City office, and a small lab in San Luis Obispo CA, with a few outliers or single-person offices.

The University of Cambridge Department of Computer Science and Technology (formerly the Computer Laboratory until recently) is located in the William Gates building, 15 JJ Thomson Avenue, Cambridge CB3 OFD, UK.

Both SRI and U.Cambridge have long histories relating to relevant theory and practice, systems-related computer science and engineering, the development of innovative trustworthy system prototypes, and the use of formal methods applied to software and more recently hardware.

5 Results and Discussion

CTSRD has produced five primary results:

The CHERI Instruction-Set Architecture (ISA) Version 7 [20] of our ISA report is the primary guide to the abstract CHERI protection model, instantiation in architecture, design rationale, and so on.

The CHERI-MIPS prototype processor Our CHERI-MIPS processor prototype, based on our baseline BERI MIPS CPU, implements the CHERI-MIPS ISA in cycle-accurate simulation and on FPGA. The prototype runs the complete CHERI prototype software stack, and allows us to evaluate both microarchitectural realism of our approaches as well as practical performance behavior through software benchmarking.

The CHERI Programmer's Guide This report provides a detailed reference to how software can use the CHERI ISA, anc includes low-level ABIs, compiler and linker behavior, debugger support, C/C++ language implications, OS support, and applications.

The CHERI prototype software stack The CHERI software stack includes CHERI Clang and LLVM, the CheriBSD operating system, and applications ported to run over them. Our design goals including illustrating both incremental adoption paths through modest modifications to existing software stacks, but also heavy use of CHERI primitives – e.g., through "pure-capability processes" that utilize CHERI capabilities in all aspects of their implementation.

The CHERI research papers and technical reports A series of top-tier research publications at venues such as ISCA, IEEE SSP, ASPLOS, MICRO, PLDI, POPL, and ACM CCS explore key design choices and their evaluations in hardware and software. We have also written additional technical reports covering topics such as the interaction between CHERI and superscalar side-channel attacks, and a shorter introductory

technical report to introduce readers to the overall CHERI program of work across hardware, software, and formal methods.

The term "CHERI" refers informally to a number of aspects of our approach, including (all parenthetical chapter numbers refer to the CHERI ISA specification):

- **The CHERI Protection Model** ([20], Chapter 2) is a capability-based model for hardware-software protection of virtual memory, typed objects, and – in a special corner case – physically addressed memory for input-output direct memory access.

- **The CHERI Instruction-Set Architecture** is a formally specified set of extensions that implement the CHERI protection model within specific existing instruction-set architectures (ISAs). As of CHERI ISAv7 [20], this includes a mature specification of CHERI-MIPS (extending 64-bit MIPS), a draft specification of CHERI-RISC-V (extending 32-bit and 64-bit RISC-V), and a sketch description of CHERI-x86-64 (extending 64-bit x86). In addition to describing specific new CHERI instructions, the CHERI ISA specification also describes how CHERI composes with existing aspects of the architecture, including the register file, CHERI-unaware instructions, virtual memory, exceptions, and so on.

  As our approach to CHERI has evolved, our understanding of the structure of CHERI's capability features – such as the contents of capabilities, whether capabilities exist in their own registers or as extensions to integer registers, and so on – has matured substantially. Each report version contains a detailed change list with respect to prior versions to allow this evolution to be followed by readers. See Section 5.1.

  The CHERI ISA is documented extensively in the Version 7 ISA [20], which updates the previous Version 6 ISA document [21] (mentioned primarily for those who wish to track our progress over the past two years). The CHERI protection model is mapped into the conceptual CHERI hardware architecture ([20], Chapter 3). That conceptual capability architecture is then mapped into the CHERI-MIPS hardware ISA ([20], Chapter 4) and the CHERI-RISC-V ISA ([20], Chapter 5). The CHERI-RISC-V ISA is being developed independently under the MTO ECATS project. In addition, ([20], Chapter 6) provides a rough sketch of what a would-be specification of a CHERI-x86-64 ISA might entail; however, there are no plans to implement such hardware.

  Several papers leading up to the ISA report are found in Section A.1. (We omit mentioning the previous versions of the ISA report, as they are summarized in considerable detail in Appendix A – Version History – of the Version 7 document.)

- **The CHERI Hardware-Software System Architecture** combines the CHERI capability-based hardware with operating systems and compilers that understand the capabilities and make constructive use of them. See Section 5.2.

- **The CHERI Application Binary Interface (CheriABI)** [4, 5] introduces the conceptual notion of abstract capabilities, which are then instantiated as architectural capabilities. CheriABI describes models for run-time linkage, the C runtime, and also OS

behavior with respect to virtual memory. Prototyped as a process model for UNIX, the conceptual approach is usable across a range of system scales and applications, including bare-metal software, embedded operating systems, and full MMU-based systems. See Section 5.3.

- **CHERI Formal Analysis** relates to formal modeling of CHERI ISA variants, and formal proofs that the CHERI ISA satisfies certain critical properties for security and memory safety [16]. The work under CTSRD has been superseded by the spin-off DARPA CHERI Instruction-set architecture Formal Verification (CIFV) project, which is analyzing the ability of CHERI ISAs to satisfy certain necessary critical properties of CHERI hardware. See Section 5.11.

5.1 CHERI Instruction-Set Architecture

The baseline CHERI ISA was first defined beginning in September 2010, with more-or-less annual documents describing its evolution. In its initial manifestation, it was conceived as a capability-based extension to a 64-bit MIPS instruction set. The resulting CHERI-MIPS-64 with 256-bit capabilities later led to CHERI-MIPS-64 with compressed 128-bit capabilities (CHERI-Concentrate [23]) and even an experimental CHERI-MIPS 32 with 64-bit capabilities. Version 7 of the CHERI ISA document [20] has separated the abstract CHERI protection model from the various implementations, including adaptations of the CHERI ISA shared with potential industrial transition partners, as well as a CHERI-RISC-V prototypes that, sketched in CTSRD, is being fully elaborated and implemented in the ECATS project. The Version 7 document also includes efforts funded partially by ECATS and CIFV as well. (That report carefully identifies the contributions of each of the other two projects, as appropriate.) Subsequent to the end of CTRSD, further versions and public releases are expected to be prepared under the auspices of the MTO ECATS project, and to some extent the CIFV project (both noted below).

5.2 The CHERI System Architecture

The original CHERI system instruction-set architecture was conceived as a clean-slate hybrid architecture. Its clean-slate nature resulted from the I2O Program Manager Howard Shrobe asking, if we were to start over, what we would do differently. Its hybrid aspect enables highly secure software to co-exist with legacy code and potential malware, because of the CHERI compartmentalization hardware mechanisms and software implementations. (The notion of a hybrid capability operating system was noted in a 2010 paper by Watson et al. [17]: Capsicum: Practical Capabilities for Unix.)

The overall CHERI total-system architecture is an example of a proactively hierarchically layered design, with formally specified ISAs, a variety of operating systems, compilers, and applications that can make constructive use of the hardware features, all with clear boundaries between layers.

5.3 The CHERI Application Binary Interface (ABI)

The CHERI hardware-software architecture imposes tight constraints on the hardware capability manipulation and its use, including nonforgeability, nonbypassability, monotonicity (inability to increase permissions or bounds), and provenance validity. With the assistance of the compiler, language runtime, and operating system, language-level safety properties – such as spatial safety – can be expressed architecturally. In addition, robust protection is also applied to sub-language implementation, such as internal program linkage, call-stack construction, exception handling, and so on. Architectural capability rules then protect against a variety of language-level vulnerabilities as well as implementation vulnerabilities.

While these rules generally permit the execution of current C and C++ code without significant modification, there are occasions on which the programmer model of pointer properties (for example) may violate rules for capabilities. For example, the architecture maintains provenance validity of capabilities from reset, permitting them to remain valid only if they are held in tagged memory or registers. In practice, operating systems may swap memory pages from DRAM to disk and back, violating architectural provenance validity. The OS kernel is able to maintain the appearance of provenance validity for swapped pages by saving tags when swapping out, and re-deriving capabilities from valid architectural capabilities when swapped back in – maintaining the *abstract capabilities* with which compiler-generated code works.

There are other cases where monotonicity prevents common internal tricks used in memory allocators, such as storing allocator metadata just outside of the allocations bounds. In CHERI, a pointer able to reach that metadata must be re-derived from a stronger capability retained by the allocator, rather than derived from the pointer returned to the allocator via the free function, whose bounds intentionally exclude that metadata!

Our ASPLOS 2019 paper on CheriABI [4] explores these issues in detail, covering topics such as context switching, the C-language runtime, virtual-memory behavior, and debugging. (This paper is item 41 in the itemization of Section A.1.)

An extended version of this paper is in a technical report [5], noted in the reports section: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-932.pdf

5.4 CHERI Operating Systems and Software

In the absence of any explicit CHERI support, it is expected that existing operating systems should run unmodified on CHERI hardware. However, they also will receive no benefit from the presence of CHERI support. Over the course of the project, we developed three operating systems able to make effective use of CHERI:

Deimos An early demonstration microkernel developed against our baseline BERI MIPS CPU prior to hardware MMU support, as well as before we had CHERI compiler support. Produced in the first year of the project, the goal of this OS was to allow us to experiment with single-address-space convergence of conventional RISC and a capability-system model. We demonstrated assembly-language use of capability fea-

tures within an otherwise C source-code base for the purposes if software compartmentalization.

CheriBSD A rich adaptation of the general-purpose open-source FreeBSD operating system to use capabilities. We began by bringing FreeBSD to the BERI MIPS CPU, as it gained MMU support, which was a relatively minor adaptation task – but invaluable in debugging BERI MIPS with larger software corpora. We then proceeded to incrementally adapt FreeBSD, focusing initially on supporting capability use in userspace – modest changes to kernel boot, context switching, virtual memory, signal delivery, and so on – followed by larger-scale changes as CHERI compiler support matured. At the end of the project, CheriBSD supports legacy MIPS binaries, hybrid binaries making selective use of capability features, and pure-capability binaries in the CheriABI process environment. We also have a variant of the kernel that is, itself, compiled to use pure-capability code. Most recently, we have developed temporal memory-safety support for CheriABI, for which we are in the final stages of a research paper. When compiled without CHERI support, CheriBSD supports current architectures and applications as the baseline FreeBSD operating system does today – a key aspect of our hybrid design that we expect would carry over to other future CHERI-enabled operating systems.

CheriOS A single address-space microkernel that relies extensively on CHERI features. This experimental system is intended to understand a more extreme spot in the CHERI software design space, and relies on the availability of capabilities to implement its "nanokernel" services, cross-domain communication via message passing and shared memory, and memory protection including revocation. CheriOS utilizes the conventional MMU still, but only to construct a larger and more flexible virtual address space for the purposes of temporal memory safety.

In addition, as part of our ECATS project, we have adapted the MMU-less FreeRTOS embedded operating system to compile as pure-capability code for CHERI-RISC-V. In mid-September 2019, we are just beginning a port of CheriBSD to the CHERI-RISC-V architecture, having completed adaptations to the RISC-V ISA to support the required functionality. It represents just one example of the close integration and collaboration among the CTSRD, ECATS, and CIFV projects.

The CHERI-LLVM compiler is an augmentation of the LLVM compiler (a common backend for C and C++ compilers) that intimately understands and uses the capability extensions, and effectively helps avoid common vulnerabilities in C/C++ programs and application program memory management. (The CHERI system also supports other programming languages, but is particularly careful about helping overcome these two languages that are inherently weak on memory safety.) We have adapted the LLD compile-time linker to support capabilities, and similarly the FreeBSD run-time linker. This allows dynamically linked applications to utilize capabilities to protect not just global variables themselves, but also the underlying implementation structures prepared by the linker – Global Offset Tables (GOTs). Similarly, Program Linkage Tables (PLTs) used for function calls are protected.

Finally, we have adapted the GDB debugger to support CHERI, allowing inspection of registers, stack tracing, break points, and other common programmer expectations to be met when developing in the CHERI environment.

5.5 The CHERI Programmer's Guide

This document has been updated from its 2018 version, for release with the current Version 7 ISA document, along with this FTR. (In the CTSRD contract, it is referred to by its proposed title, The CHERI User's Guide.) See Section B.4. As noted above, it is likely to continue to be updated beyond the end of CSTRD.

5.6 Introduction to Pure-Capability CHERI C/C++ Programming

This interim working document began in May 2019, in response to requests from potential adopters, and was first distributed at an industrial CHERI Hackathon in London on 13 June 2019. See Section B.5. It remained a standalone document with limited distribution, and has now been incorporated into the 2019 version of the CHERI Programmer's Guide, at which point it no longer exists as a stand-alone document.

5.7 Controlling Direct Memory Access

Our early work on controlling direct memory access from embedded microdevices and input-output began as a seedling task under the CTSRD project, and later has inspired subsequent work under the DARPA MTO ECATS project. Our resulting IOMMU DMA paper [12] received support from both CTSRD and ECATS; it is noted in Section A.1 as item 40. We are now pursuing issues of microcontroller integration for DMA and input-output under the ECATS project, and have begun to explore the rudiments of a capability-based interposer that would supplant the inadequate IOMMUs.

5.8 TESLA

Experienced programmers of complex software systems document and test invariants through extensive use of software assertions. Unfortunately, C language assertions are able to test only invariants that can be evaluated at the instant `assert` is invoked. Checking more complex temporal properties requires programmers to manually instrument code and data structures. This makes checking safety properties (e.g., correct memory allocation protocols, check-before-use, conformance to the TCP state machine, and wall clock timeliness goals) verbose, time-consuming, and error-prone.

TESLA significantly improves on existing approaches by enhancing the C language and runtime to support *temporal assertions*, which are able to reference past and future events. We have prototyped TESLA using a modified clang compiler that inserts runtime instrumentation, and implemented the TESLA runtime in both an operating system kernel and

userspace applications. We explore our prototype's programmability, performance, and correctness through experimentation with the FreeBSD kernel and OpenSSH. We describe TESLA in detail in a paper published at EuroSys 2014 [1]. (See A.1 item 19.)

TESLA has been used productively in the BAE/Cambridge/Memorial University CADETS (Causal, Adaptive, Distributed and Efficient Tracing System) project for DARPA I2O's Transparent Computing program.

5.9 SOAAP

SOAAP presents a new conceptual framework embodied in an LLVM-based tool: the Security-Oriented Analysis of Application Programs (SOAAP) that allows programmers to reason about compartmentalization using source-code annotations (*compartmentalization hypotheses*). Application compartmentalization, a vulnerability mitigation technique employed in programs such as OpenSSH, the open-source Chromium web browser, and its extended proprietary Chrome browser. Compartmentalization decomposes software into isolated components to limit privileges leaked or otherwise available to attackers. However, compartmentalizing applications – and maintaining that compartmentalization – is hindered by *ad hoc* methodologies and significantly increased programming effort. In practice, programmers stumble through (rather than overtly reason about) *compartmentalization spaces* of possible decompositions, unknowingly trading off correctness, security, complexity, and performance. We published a paper describing SOAAP at ACM CCS 2015 [7], as well as an extended technical-report version of the same [8]. (See A.1 item 27.)

SOAAP has been used in some of our CHERI applications efforts to determine how compartmentalization within those applications would be most effective. It was an early attempt to characterize and analyze the relative effectiveness of different compartmentalizations. SOAAP cries out for reconsideration now that the CHERI mechanisms have settled down, and are beginning to be used constructively: Whereas CHERI memory safety is applied with relative easy using the compiler, compartmentalization still requires manual program refactoring. Indeed, efforts in that direction are now ongoing under the ECATS project.

5.10 CHERI Technology Transition

Starting in 2014, SRI International and the University of Cambridge (supported by DARPA) and Arm Limited (self funding) began a long-term joint research project to develop an experimental application of the CHERI protection model to the 64-bit ARMv8-A ISA. The initial transition funds were targeted at creating an initial mapping of the CHERI model into the ARM instruction set, requiring us to address the following challenges:

Architecture-neutral and architecture-specific specifications CHERI was designed as an extension to the 64-bit MIPS ISA, CHERI-MIPS. In this step, we abstracted out the portable protection behavior and specified it independently from its architecture-specific integration.

Merged register file While in CHERI-MIPS, we introduced a new capability register file to hold capability values, in the style of a floating-point unit (which had excessive microarchitectural cost for small designs). In this step, we determined that it was safe to perform this refactoring of the design, as long as intentionality was maintained for all instructions. That implies that an operand must be identified as either an integer or a capability, rather than deciding on its interpretation based on the tag bit dynamically).

Create a 128-bit compressed capability scheme 256-bit capabilities present a substantial data-cache footprint increase for pointer-intensive workloads (such as language runtimes). We analyzed the existing literature of bounds-compression schemes (in which bounds are compressed relative to the pointer value in a fat-pointer scheme), and determined that none were suitable for use in the C/C++ idioms present in large source-code bases. We developed the CHERI-128 scheme, followed later by the CHERI Concentrate scheme (published in IEEE TCS in 2019 [23]). In particular, these support out-of-bounds pointers as they are commonly used in the open-source software we employed for evaluation.

Develop an efficient tag implementation In our earlier work developing CHERI, we employed a simple tag look-aside table in a partitioned region of DRAM, along with a tag cache. Analysis by Arm suggested that this might substantially increase DRAM access rates, which directly impacts device battery life. We developed a hierarchical tag-table and tag-cache approach that addressed this problem, exploiting the observation that tags are not evenly distributed across the physical address space.

Assist Arm with specifying a CHERI-ARM ISA We worked closely with Arm as they specified CHERI extensions to ARMv8-A, improving our design rationale as we encountered areas where choices were poorly explained, and also assisted with analyzing necessary divergences – such as the impact of architectural page tables rather than an architectural TLB.

Assist Arm in adapting CHERI Clang/LLVM We worked closely with Arm as they adapted the CHERI Clang/LLVM compiler suite for use with the experimental CHERI-ARM instruction set, as well as assisting with performance analysis and optimization.

Port our CHERI software stack to CHERI-ARM We ported our CheriBSD software stack to the CHERI-ARM architecture, testing it on an Arm-provided simulator. This allowed us to evaluate a variety of ISA design choices, as well as provide a baseline OS for performance evaluation and as a template for use.

In 2016, we received further funding from DARPA to pursue software-stack transition work, whose tasking included:

CHERI-CFI Implement compiler and OS runtime support for Control-Flow Integrity using CHERI capabilities. While the ISA had always been designed with code pointers being implemented as capabilities, this work completed the implementation that included work on dynamic linking (essential to current software environments).

CHERI-ABI Develop a pure-capability process environment for CheriBSD in which all pointers, implied (e.g., code pointers, GOT pointers, vtable pointers) or explicit (e.g., language-level pointers to code and data) are implemented using capabilities. This also required developing a capability-aware system-call layer including features such as threads and signal delivery. We published a paper on CheriABI at ASPLOS 2019 [4], as well as an extended technical-report version [5].

CHERI-C++ Implement CHERI Clang/LLVM support for the C++ programming language widely used in higher-level applications such as web browsers, mail readers, and office suites. We completed this adaptation using QTWebKit as our motivating application of choice for performance analysis.

CHERI-QEMU Continue developing and supporting Qemu as a fast emulation platform for CHERI-MIPS, to improve software development times and act as a potential foundation for a future CHERI-ARM emulator.

CHERI-DEBUG Implement CHERI support in the debugger, including extending CheriBSD to support capabilities via its `ptrace(2)` interface, the GDB debugger, and the LLDB debugger.

CHERI-KERNEL Implement pure-capability support for the kernel implementation itself, in the style of CheriABI support for userspace. We have completed this implementation, providing fine-grained memory protection for the UNIX kernel.

Throughout this transition work, we have collaborated closely with Arm, including hosting multiple Arm employees as long-term visitors at Cambridge's William Gates Building. Throughout the project, we met weekly with members of Arm's multiple teams, and held monthly CHERI-ARM steering-group meetings with members of Arm's technical leadership including architecture, development tools, and research groups. We attended multiple joint meetings with Arm customers, assisting with briefing the CHERI approach and design, engaging with customer concerns, and contributing to software analyses to help respond to concerns.

In January 2019, Arm wrote a blog article stating that they had been working with the CHERI team to explore a possible integration of CHERI into the ARM architecture. In September 2019, shortly before the end of the CTSRD contract, Arm announced a forthcoming demonstrator CPU, System-on-Chip, and development board based on an integration of CHERI with ARMv8-A. This demonstrator will be partially funded by Arm, and partially by the UK government via InnovateUK. The UK's EPSRC and ESRC funding bodies have announced solicitations for UK-based academic proposals relating to the CHERI technology and CHERI deployment. Future calls from InnovateUK will allow UK companies to bid for similar funds.

At the 26 September 2019 UKRI ISCF Digital Security by Design Challenge Collaborators' Workshop, talks were given by Robert Watson (Cambridge), Richard Grisenthwaite (Arm), and Manuel Costa (Microsoft) on CHERI and its potential applications, the forthcoming demonstration system (*Morello*), and potential avenues for future research to validate

and build on the prototype. While transition to large-scale industrial use is not yet complete, this provides a significant step in that direction.

5.11 Formal Methods

The formal methods work and draft report done originally under CTSRD have been supplanted completely by the recent CIFV project, with Peter Sewell and Prashanth Mundkur as additional investigators). It is mentioned here because of its relevance to the development of trustworthy hardware.

CIFV (funded by I2O but managed by MTO) began in February 2018, and the final CIFV report is due in February 2021, with a preliminary report in February 2020. Formal models and specifications for CHERI several variants (CHERI-MIPS and CHERI-RISC-V) are in the CIFV github repository, along with some early proofs. Peter Sewell's team at University of Cambridge has also developed a formal model for ARMv8-A (with Arm, and under support from the UK EPSRC REMS project), which is also being formally analyzed.

In July 2019, we submitted our paper, *Rigorous Engineering for Hardware Security: Formal Modelling and Proof in the CHERI Design and Implementation Process*, to IEEE SSP 2020. This paper was supported in part by the DARPA MTO CIFV project and UK-funded REMS project. This is item A.1 44 in Section A.1. The draft text has been released, in the interim, as a technical report, similarly entitled *Rigorous engineering for hardware security: formal modelling and proof in the CHERI design and implementation process* [16].

Ideally, subsequent to the end of CTSRD efforts and inspired by CIFV, formal analysis could potentially extend further down into CHERI microarchitecture, and further upward into operating-system kernels, controlled sharing and isolation provided by application compartmentalization, and properties that could be enforced by the CHERI-LLVM compiler. However, such efforts are out of scope for the existing CIFV funding, as they were for the concluding CTSRD project.

Supported in part by the DARPA MTO CIFV project and UK-funded REMS project, we submitted our paper, *ISA Semantics for ARMv8-A, RISC-V, and CHERI-MIPS* at POPL 2019 [2]. This paper describes the Sail specification of the full architectures, including systems features such as exceptions and virtual memory, for the ARMv8-A, RISC-V, and CHERI-MIPS ISAs. This is item A.1 45 in Section A.1.

5.12 Summary of Final CTSRD Technical Deliverables

- A006R: CTSRD Architecture Document – The CHERI ISA Report Version 7
- A010: CHERI Programmer's Guide (originally the User's Guide)
- A011, A011R: Executable software – source files and packaging requirements (CHERI Clang/LLVM, CheriBSD, etc. – itemized on the next page)
- A012 Commercial Off-the-shelf Manual and Associated Supplemental Data – There are no COTS products, and thus none to deliver.
- A013: Draft Final Technical Report (You are reading it now; it will be reviewed and morphed into the publicly releasable A013 Final Technical Report.)
- A014: Final Trusted Hardware – This is completely described in A006R.
- A015+: CHERI Instruction-set Architecture Report – This is included as an integral part of A006R.
- A016+: Final Prototype – This is covered by the CHERI Prototype ISA Version 7 (CDRL A006R).
- A017: Library Case Study – This deliverable is Appendix A Section A.3 in the CHERI Programmer's Guide report (A010). That appendix will eventually become an integral part of the body of the report once the compartmentalization material in the existing Appendix A is made consistent with the rest of the A010 report and with A006R. (Some refinements have been made during the government review and approval cycle for this FTR, others can be expected later, under other auspices. However, the next version of this report is likely to take place in the future.))

As noted above, some of the delivered items actually fulfill multiple CDRLs. In particular, A006R is also in fulfillment of CTSRD CDRLs A014, A015+ and A016+; An appendix of A010 is also in fulfillment of A017. The CHERI ISA report (A013) and the CHERI Programmer's Guide (A010) are living documents that will continue to be updated under other auspices subsequent to the end of CTSRD. (The A010 and A013 deliverables are consistently up-to-date as of 30 September 2019.)

The contractually required software deliverables (A011R) are provided as tarballs:

204M CheriBSD (pure-capability): 20190930-ctsrd-cheribsd-purecap-kernel.tbz
204M CheriBSD (hybrid kernel): 20190930-ctsrd-cheribsd.tbz
464K CHERI build: 20190930-ctsrd-cheribuild.tbz
45M CHERI debugger: 20190930-ctsrd-gdb.tbz
95M CHERI Clang/LLVM: 20190930-ctsrd-llvm-project.tbz
16M CHERI QEMU: 20190930-ctsrd-qemu.tbz

5.13 Potential Further CHERI-related Efforts

There is still more work that could productively be done, not just after ECATS and CIFV end, but also sooner. Here are some topic areas that would benefit from further work:

- Quantitative CHERI-ISA optimizations
- Tag tables vs. native DRAM tags
- Superscalar microarchitectures
- Non-volatile memory
- C++ compilation to CHERI
- Compiler optimizations: performance and soundness
- CHERI and ISO C/POSIX APIs
- Growing the software corpus
- Toolchain: linker, debugger, ...
- Exploring efficiency and security of compartmentalization, via tools, program annotations, and compilers as well as sandboxing frameworks within CHERI system software
- Safe native-code interfaces (e.g., JNI and other FFIs)
- Safe inter-language interoperability
- C-language garbage collection and revocation
- Accelerating and more robust managed languages
- Formal proofs of ISA properties (furthering CIFV)
- Formal proofs of software properties, including what LLVM can add
- Categorical proofs of vulnerability elimination
- Verified hardware implementations
- Microarchitectural assurance (ISA consistency with ASICs and production HW)
- Supply-chain integrity and assurance
- Pointer-based security analysis from traces
- Alternatives to IOMMUs for embedded microcontrollers and I/O (beyond ECATS)

- Architectural alternatives of CHERI ISAs for use with memory-safe languages: For example, how might CHERI hardware be simplified to run the seL4 hypervisor with memory-safe languages (e.g., Rust, OCaml)?
- MMU-free CHERI microkernel
- MMU-free HW designs for the so-called Internet of Things
- Other microarchitectural optimization opportunities for special-purpose systems and from exposed software semantics (e.g., speculative execution)
- Subsequent additional tech transition efforts and new applications

Some of these were contemplated under CTSRD, but not pursued. Others have been contemplated under ECATS and CIFV, but cannot be developed with the existing resources, as they are most likely not sufficiently in scope for either of those two ongoing projects. However, it is clear that more work could be done, particularly related to developing applications and system software for CHERI, and some advanced exploits such as DMA attacks and speculative execution that may not be adequately addressed by ECATS.

The advent of the Morello demonstrator processor, SoC, and board from Arm raise a host of further research topics relating to evaluating design choices in the SoC – number of capability registers, tagging mechanism, and so on. Further, the new version of the ARMv8-A instruction-set variant to be used in Morello will pick up more recent CHERI features, such as those relating to temporal memory safety, which can now be evaluated at a much larger scale and greater realism.

6 Conclusions

Our I2O CRASH CTSRD project is now at the end of its nine-year span. However, this is not the end for CHERI – and could perhaps be just the beginning of a new era of substantial realizations, future work, and continuing technology transfer. In conjunction with the two ongoing companion DARPA projects (MTO SSITH ECATS and MTO CIFV), the CHERI hardware-software system architecture has attained considerable maturity and enabled potential trustworthy applications. It has also opened up various opportunities for future extensions and new uses.

We are extraordinarily grateful to our DARPA program and office managers, whose foresight and wisdom allowed us to continue what started as a four-year project and enabled us to enormously enhance the elaboration of the CHERI system research and development. Our technical team was truly extraordinary, with relevant expertise and dedication. Our acknowledgments of these individuals and others who also contributed are listed in Appendix D.

The two ongoing DARPA ECATS and CIFV projects are both expected to continue into early 2021. In particular, pertinent related work is emerging from ECATS – for example, as relates to smaller 32-bit architectures extended with CHERI. The CIFV project is developing formal analyses of several CHERI variant ISAs, including CHERI-MIPS and CHERI-RISC-V, as well as a mechanically extracted ARMv8-A model derived from Arm's own ASL specification language. The ECATS and CIFV progress and results are being documented independently – although the basic CHERI ISA report will continue to be revised as a joint I2O-MTO report, subsequent to the conclusion of CTSRD.

The newly announced effort to develop the experimental Arm Morello CPU architecture, SoC, and board are a source of particular excitement. As the high-level details of this project have only been announced in the final week of our CTSRD contract, we are not able to provide more than a summary. However, the opportunity for an experimental deployment in a matured multicore superscalar design will allow the CHERI approach to be evaluated in hardware and software with a much greater degree of realism than has been achievable within the constraints of our initial research projects.

A Publications and Presentations

The items listed here are chronological, to reflect historical continuity. Each item received all or at least partial funding from the CRASH CTSRD project. Items without specific URLs included here can generally be found on the CTSRD project website:
http://www.cl.cam.ac.uk/research/security/ctsrd/
The CHERI-specific papers and reports are at
http://www.cl.cam.ac.uk/research/security/ctsrd/cheri/

Because of the extensive lists in this section, and the online accessibility of papers and reports, we have chosen not to append any of these items to this report – most notably the lengthy CHERI ISA document Version 7. However, subsequent versions of that document after the end of CSTRD are likely to continue to appear from time to time, perhaps under ECATS or other auspices.

A.1 Journal Articles, Conference Papers, and Book Chapters

For many of the listed publications, we also include a short abstract or other annotation. This list uses simple cardinal numbering, and when referred to elsewhere in this report are noted as A.1 items.

As noted earlier, all references cited in this report with square brackets are enumerated separately in the bibliography, to avoid confusion. (Some of the cited references also refer to highly relevant publications that are not attributable to CHERI-related authors.)

1. Peter G. Neumann and Robert N. M. Watson, "Capabilities Revisited: A Holistic Approach to Bottom-to-Top Assurance of Trustworthy Systems", Fourth Layered Assurance Workshop (in association with ACSAC 2010), Austin, Texas, 6-7 December 2010. (This was the first paper resulting from the CRASH CTSRD project.)
   http://www.csl.sri.com/neumann/law10.pdf

   Abstract: Long active in computer security, SRI and U.Cambridge have jointly begun a new total-system effort to develop a hierarchically layered high-assurance strongly typed capability-based system. While capabilities have long been proposed as a mechanism for mapping language structure and security policy into the hardware protection mechanism, they have seen relatively little use in general-purpose computing. A confluence of events has created the opportunity for new research, and perhaps technology transfer: soft core FPGAs, increased risk of attack even in consumer environments, and a renewed interest in revising the hardware-software interface. Capability Hardware Enhanced RISC Instructions (CHERI) will blend traditional RISC CPU instructions with new capability facilities, offering the promise of hybrid software designs easing incremental adoption. This paper represents an early-stage description of the approach and goals.

2. Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Jonathan Anderson, Ross Anderson, Nirav Dave, Ben Laurie, Simon W. Moore, Steven J. Murdoch, Philip

Paeps, Michael Roe, and Hassen Saidi, "CHERI: A Research Platform Deconflating Hardware Virtualization and Protection", RESoLVE workshop associated with ASPLOS, London, 5–7 March 2012. (This paper details subsequent progress on the development of the hardware architecture.)
http://www.csl.sri.com/neumann/2012resolve-cheri.pdf

Abstract: Contemporary CPU architectures conflate virtualization and protection, imposing virtualization-related performance, programmability, and debuggability penalties on software requiring fine-grained protection. First observed in micro-kernel research, these problems are increasingly apparent in recent attempts to mitigate software vulnerabilities through application compartmentalization. Capability Hardware Enhanced RISC Instructions (CHERI) extend RISC ISAs to support greater software compartmentalization. CHERI's *hybrid capability model* provides fine-grained compartmentalization within address spaces while maintaining software backward compatibility, which will allow the incremental deployment of fine-grained compartmentalization in both our most trusted and least trustworthy C-language software stacks. We have implemented a 64-bit MIPS research soft core, BERI, as well as a capability coprocessor, and begun adapting commodity software packages (FreeBSD and Chromium) to execute on the platform.

3. Myron King, Nirav Dave, and Arvind, "Automatic Generation of Hardware/Software Interfaces", ASPLOS 2012, London, March 2012. (This paper is based on Nirav Dave's MIT PhD thesis, for which Arvind was Nirav's thesis professor, and Myron did some of the implementation for the compiler for Nirav's BCL language extension of Bluespec's BSV.)

4. Khilan Gudka, Robert N. M. Watson, Steven Hand, Ben Laurie, and Anil Madhavapeddy. "Exploring compartmentalisation hypotheses with SOAAP", Adaptive Host and Network Security (AHANS 2012) Workshop, September 2012.
http://www.cl.cam.ac.uk/research/security/ctsrd/

Abstract: Application compartmentalization decomposes software into sandboxed components in order to mitigate security vulnerabilities, and has proven effective in limiting the impact of compromise. However, experience has shown that adapting existing C-language software is difficult, often leading to problems with correctness, performance, complexity, and most critically, security. Security-Oriented Analysis of Application Programs (SOAAP) is an in-progress research project into new semi-automated techniques to support compartmentalisation. SOAAP employs a variety of static and dynamic approaches, driven by source-code annotations termed compartmentalisation hypotheses, to help programmers evaluate strategies for compartmentalising existing software.

5. Peter G. Neumann (with Jeremy Epstein and Dan Thomsen) guest edited and coauthored the introduction to a special issue of IEEE Security and Privacy, November-December 2012. This issue included an invited article by Howard Shrobe and Daniel

Adams, "Suppose We Got a Do-Over: A Revolution for Secure Computing". All of the papers selected for this issue were intentionally highly relevant to clean-slate architectures and the CRASH program.

6. Peter G. Neumann, Inside Risks: "The Foresight Saga, Redux", Communications of the ACM, 55, 10, October 2012. (This article stresses the importance of long-term thinking.)
   http://www.csl.sri.com/neumann/cacm228.pdf

7. Peter G. Neumann, Inside Risks: "More Sight on Foresight", Communications of the ACM, 56, 2, February 2013, pages 23–25. (This article reflects on elections and natural disasters as examples of the vital need for more long-term thinking.)
   http://www.csl.sri.com/neumann/cacm229.pdf

8. Robert N.M. Watson, "A Decade of OS Access-Control Extensibility: Open-source Security Foundations for mobile and embedded devices", Communications of the ACM, 56, 2, February 2013, pages 52–63. (This is an important contribution to the general literature.)

   Abstract: To discuss operating-system security is to marvel at the diversity of deployed access-control models: Unix and Windows NT multiuser security, Type Enforcement in SELinux, anti-malware products, app sandboxing in Apple OS X, Apple iOS, and Google Android, and application-facing systems such as Capsicum in FreeBSD. This diversity is the result of a stunning transition from the narrow 1990s Unix and NT status quo to security localization-the adaptation of operating-system security models to site-local or product-specific requirements. This transition was motivated by three changes: the advent of ubiquitous Internet connectivity; a migration from dedicated embedded operating systems to general-purpose ones in search of more sophisticated software stacks; and widespread movement from multiuser computing toward single-user devices with complex application models. The transition was facilitated by extensible access-control frameworks, which allow operating-system kernels to be more easily adapted to new security requirements.

9. Jon Woodruff, Simon Moore and Robert Watson, "Memory Segmentation to Support Secure Applications, CEUR Workshop: Doctoral Symposium on Engineering Secure Software and Systems (ESSoS), Paris, France, 26–27 February 2013. Sponsored by ACM SIGSOFT, IEEE Computer Society, NESSOS, INRIA, LNCS.
   https://distrinet.cs.kuleuven.be/events/essos/2013

10. Robert N. M. Watson, Steven J. Murdoch, Khilan Gudka, Jonathan Anderson, Peter G. Neumann, and Ben Laurie. "Toward a theory of application compartmentalisation", Security Protocols XXI, 21st International Workshop, Sidney Sussex College, Cambridge UK, 18-20 March 2013, published subsequently as Springer Verlag LNCS 8263, pp. 19–27, with following transcript of discussion, pp. 28–38.

Abstract: Application compartmentalisation decomposes software applications into sandboxed components, each delegated only the rights it requires to operate. Compartmentalisation is seeing increased deployment in vulnerability mitigation, motivated informally by appeal to the principle of least privilege. Drawing a comparison with capability systems, we consider how a distributed system interpretation supports an argument that compartmentalisation improves application security.

11. William R. Harris (University of Wisconsin, Madison), Somesh Jha (University of Wisconsin, Madison), Thomas Reps (University of Wisconsin, Madison), Jonathan Anderson (University of Cambridge), and Robert N. M. Watson (University of Cambridge), "Declarative, Temporal, and Practical Programming with Capabilities", IEEE Symposium on Security and Privacy ("Oakland"), May, 2013. (This was the first publication resulting from our collaboration with other CRASH performers.)
http://research.cs.wisc.edu/wpis/papers/oakland13.pdf

12. Richard Uhler and Nirav Dave, "Smten: Automatic Translation of High-Level Symbolic Computations into SMT Queries", 25th International Conference on Computer-Aided Verification, Saint Petersburg, Russia, 13–19 July 2013.

    Smten provides user-focused abstractions for SMT queries, allowing fast modular development of formal tools. It was developed to be part of a framework in which we were developing our Bluespec-level verification tools. Smten embeds the SRI formal analysis tools – SMT queries, SMT solvers, Yices – into the Bluespec to Verilog development process. However, it has been superseded by other approaches.
    http://people.csail.mit.edu/ruhler/smten-cav13.pdf

13. Muralidaran Vijayaraghavan, Nirav Dave and Arvind, "Modular Compilation of Guarded Atomic Actions", 11th ACM-IEEE International Conference on Formal Methods and Models for Co-Design (MemoCODE 2013), Portland, Oregon, 18-20 October 2013.
    http://people.csail.mit.edu/ndave/Research/bsvmodular.pdf

    Abstract: Over the last decade, Bluespec, a hardware description language of guarded atomic actions has been used to describe rapidly modifiable, modular, no-compromise hardware designs and generate circuits from them. While the language itself supports significant modularity, the compiler compiles a module with other modules as parameters by in-lining or flattening the module. This forces the user to either suffer large compile times or to change the modular structure of the design. In this paper we propose a new modular compilation scheme which supports compilation of modules with interface methods as parameters and preserves Bluespec's one-rule-at-a-time semantic model. This compilation process inherently requires the distributed scheduling of rules.

14. A. Theodore Markettos, Jonathan Woodruff, Robert N. M. Watson, Bjoern A. Zeeb, Brooks Davis, Simon W Moore, "The BERIpad tablet: open-source construction", CPU, OS and applications, Proceedings of 2013 FPGA Workshop and Design Contest, 1–3 November 2013, Southeast University, Nanjing, China.

Abstract: We present a full desktop computer system on a portable FPGA tablet. We have designed BERI, a 64-bit MIPS R4000-style soft processor in Bluespec SystemVerilog. The processor is implemented in a system-on-chip on an Altera Stratix IV FPGA on a Terasic DE4 FPGA board that provides a full motherboard of peripherals. We run FreeBSD providing a multiuser UNIX-based OS with access to a full range of general-purpose applications. We have a thorough test suite that verifies the processor in continuous integration. We have open-sourced the complete stack at beri-cpu.org including processor, system-on-chip, physical design and OS components. We relate some of our experiences of applying techniques from successful opensource software projects on the design of open-source hardware.

15. David Chisnall, "Smalltalk in a C world", Science of Computer Programming, 18 November 2013, ISSN 0167-6423.
http://dx.doi.org/10.1016/j.scico.2013.10.013

    Abstract: Smalltalk, in spite of myriad advantages in terms of ease of development, has been largely eclipsed by lower-level languages like C, which has become the lingua franca on modern systems. This paper introduces the Pragmatic Smalltalk compiler, which provides a dialect of Smalltalk that is designed from the ground up for close interoperability with C libraries. We describe how high-level Smalltalk language features are lowered to allow execution without a virtual machine, allowing Smalltalk and C code to be mixed in the same program without hard boundaries between the two. This allows incremental deployment of Smalltalk code in programs and libraries along with heavily optimised lowlevel C and assembly routines for performance critical segments.

    syntehs

16. David Chisnall, "The Challenge of Cross-Language Interoperability: Interfacing between languages is becoming more important", Communications of the ACM, 56, 12, 50–62, December 2013.

17. David Chisnall, "LLVM in the FreeBSD Toolchain", Proceedings of AsiaBSDCon 2014, Tokyo, Japan, March 13–16, 2014.

    Abstract: FreeBSD 10 shipped with Clang, based on LLVM, as the system compiler for x86 and ARMv6+ platforms. This was the first FreeBSD release not to include the GNU compiler since the project's beginning. Although the replacement of the C compiler is the most obvious user-visible change, the inclusion of LLVM provides opportunities for other improvements.

18. Brooks Davis, Robert Norton, Jonathan Woodruff, and Robert N. M. Watson. "How FreeBSD Boots: a soft-core MIPS perspective", Proceedings of AsiaBSDCon 2014, Tokyo, Japan, March 13–16 2014.

    Abstract: We have implemented a soft-core, multi-threaded, 64-bit MIPS R4000-style CPU called BERI to support research on the hardware/software interface. We have

ported FreeBSD to this platform including support for multi-threaded symmetric multiprocessing. This paper describes the process by which a BERI system boots from CPU startup through the boot loaders, hand off to the kernel, and enabling secondary CPU threads. Historically, the process of booting FreeBSD has been documented from a user perspective or at a fairly high level. This paper aims to improve the documentation of the low level boot process for developers aiming to port FreeBSD to new targets.

19. Jonathan Anderson, Robert N. M. Watson, David Chisnall, Khilan Gudka, Brooks Davis, and Ilias Marinos. TESLA: Temporally Enhanced System Logic Assertions, Proceedings of the 2014 European Conference on Computer Systems (EuroSys 2014), Amsterdam, The Netherlands, April 14–16 2014.
https://www.cl.cam.ac.uk/research/security/ctsrd/

    Abstract: Large, complex, rapidly evolving pieces of software such as operating systems are notoriously difficult to prove correct. Instead, OS developers use techniques such as assertions to describe the expected behaviour of their systems and check actual behaviour through testing. However, many dynamic safety properties cannot be validated this way because they are temporal in nature, they depend on events in the past or the future and are not easily expressed in assertions.

    TESLA is a description, analysis, and validation tool that allows systems programmers to describe expected temporal behaviour in low-level languages such as C. *Temporal assertions* can span the interfaces between libraries and even languages. TESLA exposes run-time behaviour using program instrumentation, illuminating coverage of complex state machines and detecting violations of specifications.

    We evaluate TESLA by applying it to complex software, including an OpenSSL security API, the FreeBSD kernel access-control framework, and GNUstep's rendering engine. With performance that allows "always-on" availability, we demonstrate that existing systems can benefit from a richer dynamic analysis without being re-written for amenability to a complete formal analysis.

20. Jonathan Woodruff, Robert N. M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, and Michael Roe. The CHERI capability model: Revisiting RISC in an age of risk, Proceedings of the 41st International Symposium on Computer Architecture (ISCA 2014), Minneapolis, MN, USA, June 14–16, 2014. (This paper received an "honorable mention" in the Guest Editor piece for the Micro Top Picks edition.)
https://www.cl.cam.ac.uk/research/security/ctsrd/

    Abstract: Motivated by contemporary security challenges, we re-evaluate and refine capability-based addressing for the RISC era. We present CHERI, a hybrid capability model that extends the 64-bit MIPS ISA with byte-granularity memory protection. We demonstrate that CHERI enables language memory model enforcement and fault isolation in hardware rather than software, and that the CHERI mechanisms are easily adopted by existing programs for efficient in-program memory safety.

In contrast to past capability models, CHERI complements, rather than replaces, the ubiquitous page-based protection mechanism, providing a migration path towards deconflating data-structure protection and OS memory management. Furthermore, CHERI adheres to a strict RISC philosophy: it maintains a load-store architecture and requires only single-cycle instructions, and supplies protection primitives to the compiler, language runtime, and operating system.

We demonstrate a mature FPGA implementation that runs the FreeBSD operating system with a full range of software and an open-source application suite compiled with an extended LLVM to use CHERI memory protection. A limit study compares published memory safety mechanisms in terms of instruction count and memory overheads. The study illustrates that CHERI is performance-competitive even while providing assurance and greater flexibility with simpler hardware.

21. Ilias Marinos, Robert N. M. Watson, and Mark Handley, "Network Stack Specialization for Performance", Proceedings of ACM SIGCOMM 2014 Conference (SIGCOMM'14), Chicago, IL, USA, August 17–22, 2014.

22. Richard Uhler and Nirav Dave, "Smten with Satisfiability-based Search", OOPSLA 2014, Portland, Oregon, 20–24 October 2014.

    Abstract: Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) have been used in solving a wide variety of important and challenging problems, including automatic test generation, model checking, and program synthesis. For these applications to scale to larger problem instances, developers cannot rely solely on the sophistication of SAT and SMT solvers to efficiently solve their queries; they must also optimize their own orchestration and construction of queries. We present Smten, a high-level language for orchestrating and constructing satisfiability-based search queries. We show that applications developed using Smten require significantly fewer lines of code and less developer effort to achieve results comparable to standard SMT-based tools.

23. Brooks Davis, Robert Norton, Jonathan Woodruff, Robert N. M. Watson, "Bringing up MIPS", FreeBSD Journal, The FreeBSD Foundation, January-February 2015.

24. David Chisnall, Colin Rothwell, Brooks Davis, Robert N.M. Watson, Jonathan Woodruff, Simon W. Moore, Peter G. Neumann, and Michael Roe, "Beyond the PDP-11: Architectural Support for a Memory-Safe C Abstract Machine", 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2015), Istanbul, Turkey, 14–18 March 2015. (Best Presentation Award to David Chisnall!)

    Abstract: We propose a new memory-safe interpretation of the C abstract machine that provides stronger protection to benefit security and debugging. Despite ambiguities in the specification intended to provide implementation flexibility, contemporary implementations of C have converged on a memory model similar to the PDP-11,

the original target for C. This model lacks support for memory safety despite well-documented impacts on security and reliability.

Attempts to change this model are often hampered by assumptions embedded in a large body of existing C code, dating back to the memory model exposed by the original C compiler for the PDP-11. Our experience with attempting to implement a memory-safe variant of C on the CHERI experimental microprocessor led us to identify a number of problematic idioms. We describe these as well as their interaction with existing memory safety schemes and the assumptions that they make beyond the requirements of the C specification. Finally, we refine the CHERI ISA and abstract model for C, by combining elements of the CHERI capability model and fat pointers, and present a softcore CPU that implements a C abstract machine that can run legacy C code with strong memory protection guarantees.

25. Robert N. M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera, "CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization", IEEE Symposium on Security and Privacy, San Jose, CA, May 18-20, 2015.

    Abstract: CHERI extends a conventional RISC Instruction-Set Architecture, compiler, and operating system to support fine-grained, capability-based memory protection to mitigate memory-related vulnerabilities in C-language TCBs. We describe how CHERI capabilities can also underpin a hardware-software object-capability model for application compartmentalization that can mitigate broader classes of attack. Prototyped as an extension to the open-source 64-bit BERI RISC FPGA softcore processor, FreeBSD operating system, and LLVM compiler, we demonstrate multiple orders-of-magnitude improvement in scalability, simplified programmability, and resulting tangible security benefits as compared to compartmentalization based on pure Memory-Management Unit (MMU) designs. We evaluate incrementally deployable CHERI-based compartmentalization using several real-world UNIX libraries and applications.

26. Muralidaran Vijayaraghavan, Nirav Dave, and Arvind, "Modular Deductive Verification of Multiprocessor Hardware Designs", 27th International Conference on Computer Aided Verification (CAV 2015), San Francisco, 18-24 July 2015.

27. Khilan Gudka (University of Cambridge), Robert N. M. Watson, Jonathan Anderson, David Chisnall, Brooks Davis. Ben Laurie (Google UK Ltd.), Ilias Marinos, Peter G. Neumann, Alex Richardson, "Clean Application Compartmentalization with SOAAP", 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS 2015), 12-16 October 2015, Denver, Colorado.
    https://ccs2015.cs.ucsb.edu/hc/paper.php/97?cap=097abFcxmySsppc

    Abstract: Application compartmentalization, a vulnerability mitigation technique employed in programs such as OpenSSH and the Chromium web browser, decomposes software into isolated components to limit privileges leaked or otherwise available to

attackers. However, compartmentalizing applications – and maintaining that compartmentalization – is hindered by *ad hoc* methodologies and significantly increased programming effort. In practice, programmers stumble through (rather than overtly reason about) *compartmentalization spaces* of possible decompositions, unknowingly trading off correctness, security, complexity, and performance. We present a new conceptual framework embodied in an LLVM-based tool: the Security-Oriented Analysis of Application Programs (SOAAP) that allows programmers to reason about compartmentalization using source-code annotations (*compartmentalization hypotheses*). We demonstrate considerable benefit when creating new compartmentalizations for complex applications, and analyze existing compartmentalized applications to discover design faults and maintenance issues arising from application evolution.

28. Matthew Naylor and Simon Moore, "Bluecheck: A Generic Synthesizable Test Bench", ACM-IEEE MEMOCODE, 21–23 September 2015, Austin, Texas.

    Abstract: Writing test benches is one of the most frequently performed tasks in the hardware development process. The ability to reuse common test bench features is therefore key to productivity. In this paper, we present a generic test bench, parameterised by a specification of correctness, which can be used to test any design. Our test bench provides several important features, including automatic test-sequence generation and shrinking of counter-examples, and is fully synthesisable, allowing rigorous testing on FPGA as well as in simulation. The approach is easy to use, cheap to implement, and encourages the formal specification of hardware components through the reward of automatic testing and simple failure cases.

29. Brooks Davis, Building a Foundation for Secure Trusted Computing Bases, FreeBSD Journal, March-April 2016.

    Abstract: BSD operating systems have been around since the 1980s, and the history of UNIX extends all the way back to 1969, but despite orders of magnitude growth in performance, storage, and memory capacity, we still use CPUs with computing models that are remarkably similar to the PDP-11 on which the early versions of UNIX were run. We have a flat virtual address space (now 64 bits instead of 16), TLB-based process virtualization of memory, and permissions at page granularity.

30. Robert N. M. Watson, Simon W. Moore, and Peter G. Neumann, CHERI: a hardware-software system to support the principle of least privilege, ERCIM News, The European Research Consortium for Informatics and Mathematics, June 2016. (Subtitle: The CHERI hardware-software system has the potential to provide unprecedented security, reliability, assurance, ease of programmability, and compatibility.) This article provides a short summary of our clean-slate hardware-software co-design for the CHERI system, published in a journal that has frequent articles on trustworthiness, safety, security, reliability, and related topics.
    http://ercim-news.ercim.eu/en106

Abstract: The CHERI hardware-software system has the potential to provide unprecedented security, reliability, assurance, ease of programmability, and compatibility.

31. Robert N. M. Watson, Robert Norton, Jonathan Woodruff, Alexandre Joannou, Simon W. Moore, Peter G. Neumann, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, A. Theodore Markettos, Ed Maste, Steven J. Murdoch, Michael Roe, Colin Rothwell, Stacey Son, and Munraj Vadera, Fast Protection-Domain Crossing in the CHERI Capability-System Architecture, special Issue of IEEE Micro journal, vol. 36, no. 5, pp. 38–49, Sept-Oct 2016, (This paper is an extension and refinement of the 2015 paper for the IEEE Symposium on Security and Privacy.)
https://www.cl.cam.ac.uk/research/security/ctsrd/
https://www.repository.cam.ac.uk/handle/1810/257042

    Abstract: Capability Hardware Enhanced RISC Instructions (CHERI) supplement the conventional Memory Management Unit (MMU) with Instruction-Set Architecture (ISA) extensions that implement an in-address-space capability-system model. CHERI capabilities can also underpin a hardware-software object-capability model for scalable application compartmentalization that can mitigate broader classes of attack. This paper describes ISA additions to CHERI that support fast protection-domain switching, not only in terms of low cycle count, but also efficient memory sharing with mutual distrust. We propose ISA support for sealed capabilities, hardware-assisted checking during protection-domain switching, a lightweight capability flow-control model, and fast register clearing – while retaining the flexibility of a software-defined protection-domain transition model. We validate this approach through a full-system experimental design including ISA extensions, FPGA prototype (implemented in Bluespec SystemVerilog), and software stack including OS (based on FreeBSD), compiler (based on LLVM), software compartmentalization model, and open-source applications.

32. David Chisnall, Brooks Davis, Khilan Gudka, David Brazdil, Alexandre Joannou, Jonathan Woodruff, A. Theodore Markettos, J. Edward Maste, Robert Norton, Stacey Son, Michael Roe, Simon W. Moore, Ben Laurie, Peter G. Neumann, and Robert N. M. Watson, CHERI JNI: Sinking the Java security model into the C, Proceedings of the 22nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2017). Xi'an, China, April 8–12, 2017.

    Abstract: We characterize the cache behavior of an in-memory tag table and demonstrate that an optimized implementation can typically achieve a near-zero memory traffic overhead. Both industry and academia have repeatedly demonstrated tagged memory as a key mechanism to enable enforcement of powerful security invariants, including capabilities, pointer integrity, watchpoints, and information-flow tracking. A single-bit tag shadowspace is the most commonly proposed requirement, as one bit is the minimum metadata needed to distinguish between an untyped data word and any number of new hardware-enforced types. We survey various tag shadowspace approaches and identify their common requirements and positive features of their

implementations. To avoid non-standard memory widths, we identify the most practical implementation for tag storage to be an in-memory table managed next to the DRAM controller. We characterize the caching performance of such a tag table and demonstrate a DRAM traffic overhead below 5% for the vast majority of applications. We identify spatial locality on a page scale as the primary factor that enables surprisingly high table cache-ability. We then demonstrate tag-table compression for a set of common applications. A hierarchical structure with elegantly simple optimizations reduces DRAM traffic overhead to below 1% for most applications. These insights and optimizations pave the way for commercial applications making use of single-bit tags stored in commodity memory.

33. Brooks Davis, Everything you ever wanted to know about "hello, world"* (*but where afraid to ask), invited paper and talk for AsiaBSDCon 2017, Tokyo, 9-12 March 2017. This paper illustrates the somewhat surprising complexity inherent in a very simple program.

    Abstract: The first example in the classic book "The C Programming Language" by Kernighan and Ritchie[10] is in fact a remarkably complete test of the C programming language; it prints the text "hello, world" on a single line and exits. It's seemingly simple and straightforward (once the programmer understands that `\n` means *newline*). In reality though, it compiles to over 550KiB on MIPS64! This paper provides a guided tour of a slightly more complex program, where `printf` is called with multiple arguments. Along the way from the initial processes' call to `exec` to the final `_exit`, we'll tour the program loading code in the kernel, the basics of system-call implementation, the implementation of the memory allocator, and of course `printf`. We'll also touch on localization, a little on threading support, and a brief overview of the dynamic linker.

34. Brooks Davis, CheriBSD: A Research Fork of FreeBSD, AsiaBSDCon 2017, Tokyo, 9-12 March 2017.

    Abstract: CheriBSD is a fork of FreeBSD to support the CHERI research CPU. We have extended the kernel to provide support for CHERI memory capabilities as well as modifying applications and libraries including `tcpdump`, `libmagic`, and `zlib` to take advantage of these capabilities for improved memory safety and compartmentalization. We have also developed custom demo applications and deployment infrastructure for our table demo platform. This paper discusses the challenges facing a long-running public fork of FreeBSD.

    These challenges include keeping up with FreeBSD-CURRENT, tools and strategies for using `git`, and the difficulties – and value – of upstreaming improvements. We also cover our internal and external release process and the products we produce. CheriBSD targets a research environment, but lessons learned apply to many environments building products or services on customized versions of FreeBSD.

35. Alexandre Joannou, Jonathan Woodruff, Robert Kovacsics, Simon W. Moore, Alex Bradbury, Hongyan Xia, Robert N. M. Watson, David Chisnall, Michael Roe, Brooks

Davis, Edward Napierala, John Baldwin, Khilan Gudka, Peter G. Neumann, Alfredo Mazzinghi, Alexander Richardson, Stacey Son, A. Theodore Markettos, Efficient Tagged Memory, International Conference on Computer Design, ICCD 2017, Boston, 5-8 November 2017.

Abstract: We characterize the cache behavior of an in-memory tag table and demonstrate that an optimized implementation can typically achieve a near-zero memory traffic overhead. Both industry and academia have repeatedly demonstrated tagged memory as a key mechanism to enable enforcement of powerful security invariants, including capabilities, pointer integrity, watchpoints, and information-flow tracking. A single-bit tag shadowspace is the most commonly proposed requirement, as one bit is the minimum metadata needed to distinguish between an untyped data word and any number of new hardware-enforced types. We survey various tag shadowspace approaches and identify their common requirements and positive features of their implementations. To avoid non-standard memory widths, we identify the most practical implementation for tag storage to be an in-memory table managed next to the DRAM controller. We characterize the caching performance of such a tag table and demonstrate a DRAM traffic overhead below 5on a page scale as the primary factor that enables surprisingly high table cache-ability. We then demonstrate tag-table compression for a set of common applications. A hierarchical structure with elegantly simple optimizations reduces DRAM traffic overhead to below 1insights and optimizations pave the way for commercial applications making use of single-bit tags stored in commodity memory.

36. Robert N. M. Watson, Peter G. Neumann, and Simon W. Moore, Balancing Disruption and Deployability in the CHERI Instruction-Set Architecture (ISA), Chapter 5 in *New Solutions for Cybersecurity*, Howie Shrobe, David Shrier, Alex Pentland, eds., MIT Press/Connection Science: Cambridge Mass., January 2018. (There is no abstract; see the next item.)

37. Peter G. Neumann, Fundamental Trustworthiness Principles, Chapter 6 in *New Solutions for Cybersecurity*, Howie Shrobe, David Shrier, Alex Pentland, eds., MIT Press/Connection Science: Cambridge Mass., January 2018.

    Original abstract (which the editors excised – because *no chapter in this book has an abstract!*): Enormous benefits can result from basing requirements, architectures, implementations, and operational practices on well-defined and well-understood generally accepted principles. Furthermore, any set of principles is by itself clearly incomplete. However, considerable experience, understanding, and foresight are needed to use such principles productively.

    In this chapter, we itemize, review, and interpret various design and development principles that – if properly understood and applied – can advance predictable composability of components, total-system trustworthiness, high assurance, and other attributes of computer systems and networks. We consider the relative applicability of those

principles, as well as some of the problems they may introduce. We also examine the pervasive way in which these design and development principles have inspired and motivated the prototype CHERI architecture. (The chapter of course refers to the previous chapter, noted in the previous item.)

38. Alfredo Mazzinghi, Ripduman Sohan, and Robert N. M. Watson, Pointer Provenance in a Capability Architecture, Proceedings of the 10th USENIX Workshop on the Theory and Practice of Provenance (TaPP'18), London, July 2018.

    Abstract: We design and implement a framework for tracking pointer provenance, using our CHERI fat-pointer capability architecture to facilitate analysis of security implications of program pointer flows in both user and privileged code, with minimal instrumentation. CHERI enforces pointer provenance validity at the architectural level, in the presence of complex pointer arithmetic and type casting. CHERI present new opportunities for provenance research: we discuss use cases and highlight lessons and open questions from our work.

39. Hongyan Xia, Jonathan Woodruff, Hadrien Barral, Lawrence Esswood, Alexandre Joannou, Robert Kovacsics, David Chisnall, Michael Roe, Brooks Davis, John Baldwin, Khilan Gudka, Peter G. Neumann, Alex Richardson, Edward Napierala, Simon W. Moore and Robert N. M. Watson, CheriRTOS: A Capability Model for Embedded Devices, International Conference on Computer Design, ICCD, Orlando FL, 7-10 October 2018.

    Abstract: Embedded systems are deployed ubiquitously among various sectors including automotive, medical, robotics and avionics. As these devices become increasingly connected, the attack surface also increases tremendously; new mechanisms must be deployed to defend against more sophisticated attacks while not violating resource constraints. In this paper we present CheriRTOS on CHERI-64, a hardware-software platform atop Capability Hardware Enhanced RISC Instructions (CHERI) for embedded systems.

40. A. Theodore Markettos, Colin Rothwell, Brett F. Gutstein, Allison Pearce, Peter G. Neumann, Simon W. Moore, and Robert N. M. Watson, Thunderclap: Exploring Vulnerabilities in Operating-System IOMMU Protection via DMA from Untrustworthy Peripherals, Network and Distributed Systems Security (NDSS 2019), San Diego CA, 24-27 February 2019.

    Abstract: Direct Memory Access (DMA) attacks have been known for many years: DMA-enabled I/O peripherals have complete access to the state of a computer and can fully compromise it including reading and writing all of system memory. With the popularity of Thunderbolt 3 over USB Type-C and smart internal devices, opportunities for these attacks to be performed casually with only seconds of physical access to a computer have greatly broadened. In response, commodity hardware and operating-system (OS) vendors have incorporated support for Input-Output Memory Management Units (IOMMUs), which impose memory protection on DMA, and are

widely believed to protect against DMA attacks. We investigate the state-of-the-art in IOMMU protection across OSes using a novel *I/O-security research platform*, and find that current protections fall short when faced with a functional network peripheral that uses its complex interactions with the OS for ill intent. We describe vulnerabilities in macOS, FreeBSD, and Linux, which notionally utilize IOMMUs to protect against DMA attackers. Windows uses the IOMMU only in limited cases. and it remains vulnerable. Using Thunderclap, an open-sourced FPGA research platform that we built, we explore new classes of OS vulnerability arising from inadequate use of the IOMMU. The complex vulnerability space for IOMMU-exposed shared memory available to DMA-enabled peripherals allows attackers to extract private data (sniffing cleartext VPN traffic) and hijack kernel control flow (launching a root shell) in seconds using devices such as USB-C projectors or power adapters. We have worked closely with OS vendors to remedy these vulnerability classes, and they have now shipped substantial feature improvements and mitigations as a result of our work. (This paper had support from CTSRD and ECATS.)

41. Brooks Davis, Robert N. M. Watson, Alexander Richardson, Peter G. Neumann, Simon W. Moore, John Baldwin, David Chisnall, James Clarke, Nathaniel Wesley Filardo, Khilan Gudka, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, J. Edward Maste, Alfredo Mazzinghi, Edward Tomasz Napierala, Robert M. Norton, Michael Roe, Peter Sewell, Stacey Son, and Jonathan Woodruff. CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment. In Proceedings of 2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS 2019). Providence, RI, USA, 13-17 April 2019. (This paper had support from CTSRD and ECATS. It received the Best Paper Award.)

    Abstract: The CHERI architecture allows pointers to be implemented as capabilities (rather than integer virtual addresses) in a manner that is compatible with, and strengthens, the semantics of the C language. In addition to the spatial protections offered by conventional fat pointers, CHERI capabilities offer strong integrity, enforced provenance validity, and access monotonicity. The stronger guarantees of these architectural capabilities must be reconciled with the real-world behavior of operating systems, run-time environments, and applications. When the process model, user-kernel interactions, dynamic linking, and memory management are all considered, we observe that simple derivation of architectural capabilities is insufficient to describe appropriate access to memory. We bridge this conceptual gap with a notional *abstract capability* that describes the accesses that should be allowed at a given point in execution, whether in the kernel or userspace. To investigate this notion at scale, we describe the first adaptation of a full C-language operating system (FreeBSD) with an enterprise database (PostgreSQL) for complete spatial and referential memory safety. We show that awareness of abstract capabilities, coupled with CHERI architectural capabilities, can provide more complete protection, strong compatibility, and acceptable performance overhead compared with the pre-CHERI baseline and software-only approaches. Our observations also have potentially significant implications for other

mitigation techniques.

42. Jonathan Woodruff, Alexandre Joannou, Hongyan Xia, Anthony Fox, Robert Norton, David Chisnall, Brooks Davis, Khilan Gudka, Nathaniel W. Filardo, A. Theodore Markettos, Michael Roe, Peter G. Neumann, Robert N. M. Watson, and Simon W. Moore, CHERI Concentrate: Practical Compressed Capabilities. IEEE Transactions on Computing, April 2019.

    Abstract: We present CHERI Concentrate, a new fat-pointer compression scheme applied to CHERI, the most developed capability-pointer system at present. Capability fat-pointers are a primary candidate for enforcing fine-grained and non-bypassable security properties in future computer systems, although increased pointer size can severely affect performance. Thus, several proposals for capability compression have been suggested that do not support legacy instruction sets, ignore features critical to the existing software base, and also introduce design inefficiencies to RISC-style processor pipelines. CHERI Concentrate improves on the state-of-the-art region-encoding efficiency, solves important pipeline problems, and eases semantic restrictions of compressed encoding, allowing it to protect a full legacy software stack. We present the first quantitative analysis of compiled capability code, which we use to guide the design of the encoding format. We analyze and extend logic from the open-source CHERI prototype processor design on FPGA to demonstrate encoding efficiency, minimize delay of pointer arithmetic, and eliminate additional load-to-use delay. To verify correctness of our proposed high-performance logic, we present a HOL4 machine-checked proof of the decode and pointer-modify operations. Finally, we measure a 50% to 75% reduction in L2 misses for many compiled C-language benchmarks running under a commodity operating system using compressed 128-bit and 64-bit formats, demonstrating both compatibility with and increased performance over the uncompressed, 256-bit format. (This paper had support from CTSRD and ECATS.)

43. Hongyan Xia, Jonathan Woodruff, San Ainsworth, Nathaniel W. Filardo, Peter G. Neumann, Simon W. Moore, Robert N. M. Watson, and Timothy M. Jones, CHERIvoke: Characterizing Pointer Revocation Using CHERI Capabilities for Temporal Memory Safety, IEEE Micro conference, October 2019.

    Abstract: A lack of temporal safety in low-level languages has led to an epidemic of use-after-free exploits. These have surpassed in number and severity even the infamous buffer-overflow exploits violating spatial safety. Capability addressing can directly enforce *spatial* safety for the C language by enforcing bounds on pointers and by rendering pointers unforgeable. Nevertheless, an efficient solution for strong *temporal* memory safety remains elusive.

    CHERI is an architectural extension to provide hardware capability addressing that is seeing significant commercial and open-source interest. We show that CHERI capabilities can be used as a foundation to enable low-cost heap temporal safety by facilitating out-of-date pointer revocation, as capabilities enable precise and efficient

identification and invalidation of pointers, even when using unsafe languages such as C. We develop CHERIvoke, a technique for deterministic and fast sweeping revocation to enforce temporal safety on CHERI systems. CHERIvoke quarantines freed data before periodically using a small shadow map to revoke all dangling pointers in a single sweep of memory, and provides a tunable tradeoff between performance and heap growth.

We evaluate the performance of such a system for high-performance processors, and further analytically examine its primary overheads. When configured with a heap-size overhead of 25%, we find that CHERIvoke achieves an average execution-time overhead of under 5%, far below the overheads associated with traditional garbage collection, revocation, or page-table systems.

44. Kyndylan Nienhuis, Alexandre Joannou, Anthony Fox, Michael Roe, Brian Campbell, Matthew Naylor, Robert M. Norton, Simon W. Moore, Peter G. Neumann, Ian Stark, Robert N. M. Watson, and Peter Sewell, Rigorous Engineering for Hardware Security: Formal Modelling and Proof in the CHERI Design and Implementation Process. Submitted to the July batch of papers for IEEE 41st Symposium on Security and Privacy, May 2020. (SSP now has monthly rolling submissions.) It has received a "Revise and Resubmit" response, which is encouraging (with only one mildly negative reviewer, and minor points to be fixed.) If ultimately accepted, it will appear online in its near-final form, well before the the final version would be presented at SSP 2020 in May 2020.)

    Authors span Arm, Cambridge, SRI, and the University of Edinburgh, with support primarily under the UK REMS project – which is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 789108), with some additional support from CTSRD and CIFV. This is truly a joint and highly synergistic effort, with real practical implications. It is also providing the formal basis for CIFV's analysis of several CHERI variants other than CHERI-MIPS. This paper received two positive reviews and one marginal negative one in October 2019, and received an evaluation of please "revise and resubmit." It has been revised and resubmitted in October 2019.

    Abstract: This paper describes our machine-checked proofs of various security properties of the entire CHERI-MIPS ISA, together with discussion of how we've used formal ISA models in lightweight ways during the CHERI-MIPS design and implementation process. (CHERI-MIPS has been developed under CTSRD. It was formally specified originally in L3, and now in the Cambridge SAIL specification language for use by the Cambridge formal analysis tools.)

45. Ben Simner, Shaked Flur, Christopher Pulte, Alasdair Armstrong, Jean Pichon-Pharabod, Luc Maranget, and Peter Sewell, ARMv8-A System Semantics: Instruction Fetch in Relaxed Architectures.

    Authors are all at U.Cambridge, except for Luc Maranget – who is now at INRIA in Paris. This paper originated from previously ongoing work under Peter Sewell's REMS project at U.Cambridge, mostly independently of the three SRI-Cambridge

DARPA projects noted here. However, it is highly relevant to CTSRD – as it focuses on formal specifications for ARM-v7-A, and thus is included here because of its seminal relevance to the work related to the CHERI-ARM-v8 that has emerged in connection with CTSRD, and because it provides a useful starting point for CIFV. (Infortunately, it was rejected during October 2019, and will presumably be repurposed for another venue.)

Draft abstract: This paper establishes rigorous semantics for instruction fetch and icache maintenance (in the concurrent setting) for ARMv8-A, in consultation with the Arm chief architect and others. Accepted at POPL. This was partly funded by CIFV – it exploits the Sail-to-SMT backend originally developed to quickly check properties of CHERI Sail models, to evaluate an axiomatic semantics for instruction fetch on concurrent tests. This is a first step in establishing rigorous concurrent *system* semantics; previous work has covered only *user-mode* concurrency.

A.2 Other Relevant Public Items

- Robert Watson, IEEE Spectrum Techwise Conversation podcast interview, recorded 26 December 2012, explores the argument for clean-slate design and the nature of current attacker-defender asymmetry.
  http://spectrum.ieee.org/podcast/computing/software/computers-its-time-to-start-ove

- Portrait: Robert Watson, Research on the Hardware-Software Interface.
  http://queue.acm.org/detail_video.cfm?id=2382552

- Peter Neumann, Minnesota Public Radio's The Daily Circuit, 27 December 2012, discussing the inadequacy of passwords and the need for trustworthy systems.
  http://minnesota.publicradio.org/display/web/2012/12/27/daily-circuit-cyber-securit

- Peter Neumann delivered the 2013 Elliott Organick Memorial Lectures at the University of Utah in March 2013. The slides for the first lecture, A Personal History of Layered Trustworthiness, are online.
  http://www.csl.sri.com/neumann/utah13+x4.pdf
  The slides for the second lecture used some of these slides plus previously released material from the SRI/U.Cambridge presentations at the fall 2012 PI meetings.

- Peter Neumann received the 2013 Computing Research Association Distinguished Contributions Award at the ACM Awards Banquet in San Francisco. 15 June 2013.

- Peter Neumann was interviewed for the Computer Security History Project: http://conservancy.umn.edu/handle/11299/162377 (click on 'Peter Neumann'), The entire collection of interviews is worth visiting:
  http://www.cbi.umn.edu/oh/index.html

- Peter G Neumann, Sean Peisert, and Marv Schaefer, "The IEEE Symposium on Security and Privacy in Retrospect," IEEE Security and Privacy, invited article, June 2014, pages 2–4 (introduction to the special issue on best SSP papers from 2013).

- Robert Watson was interviewed for The Economist's June 2014 Technology Quarterly, explaining how compartmentalized software designs could have mitigated vulnerabilities such as Heartbleed. https://www.economist.com/technology-quarterly/2014/06/05/hiding-from-big-data

- Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Peter G. Neumann, Susan Landau, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael Specter, Daniel J. Weitzner, "Keys Under Doormats: Mandating insecurity by requiring government access to all data and communications.
  http://dspace.mit.edu/handle/1721.1/97690
  The complete version of Keys Under Doormats was published in the (fully open-access) Journal of Cybersecurity, vol 1 no 1, Oxford University Press, November 2015.
  http://cybersecurity.oxfordjournals.org/content/1/1/69
  This article won the 2015 J.D. Falk Award from the Messaging Malware Mobile Anti-Abuse Working Group (M3AAWG) at their annual meeting in Atlanta, 20-22 Oct 2015: "The M3AAWG J.D. Falk Award seeks to recognize people who are committed to making a better online world... The award seeks to recognize efforts for a particularly meritorious item of work... The recipient must also embody the spirit of J.D.'s volunteerism and community building. The J.D. Falk Award winners have a vigilant eye on the broader perspective of Internet systems and communities and call upon thoughtful humor when things get tough." (Peter Neumann contributed a three-minute video that was shown at the award ceremony.) The article's authors also subsequently received the Electronic Freedom Foundation Pioneer Award for 2016.

  It is of considerable interest to cryptographic uses of CHERI-based systems because of the need for highly trustworthy systems on which to base nonsubvertible cryptographic implementations.

- Robert Watson (and other SSITH PIs) were interviewed for an article appearing in the 11 August 2018 issue of New Scientist, *Uncrackable computer chips stop malicious bugs attacking your computer*. The article described ongoing research involving architectural security, including CHERI. https://www.newscientist.com/article/mg23931900-300-uncrackable-computer-chips-stop-malicious-bugs-attacking-your-comput

- Peter Neumann keynoted the first DARPA-sponsored seL4 Summit in November 2018, summarizing the CHERI-related work to date, and considering the potentials of developing a CHERI-RISC-V enhanced seL4, to provide added trustworthiness to seL4 from the CHERI hardware, as well as to provide greater assurance for user code.

- Leah Hoffman interviewed Peter Neumann, Promoting Common Sense, Reality, Dependable Engineering, Communications of the ACM 61, 12, 128-127 (sic), December 2018.

- The IOMMU direct-memory access paper received considerable attention, including coverage in The Register, ZDNet, and the BBC. Thunderclap now has its own website, where further information can be found: http://thunderclap.io

  It is evident that our I20/MTO Thunderclap work has significantly influenced the evolving USB 4 specification. The revised version (which was just released in early September 2019) incorporates substantial parts of our Thunderclap recommendations. Like Thunderbolt, it also addresses PCE Express DMA traffic. Nevertheless, the revised USB 4 spec still appears to be potentially vulnerable to Thunderclap-style attacks.

  In that our industry collaborators have been on the standards committee and long been aware of our Thunderclap work, they were able to make direct inputs into the standardization process, and also inform other industry players of our threat model. DARPA might now wish to consider our Thunderclap work as a "BIG WIN* for DARPA", as the new USB 4 standard mandates our primary recommendations for hardening it against malicious Thunderbolt devices, and repeats our observation that careful design of operating system software and device drivers is still essential (despite the new standard).

- Peter Neumann participated in the fortieth Symposium on Security and Privacy, 20-22 May 2019. He served on a retrospective-futurist SSP panel on 21 May 2019, with Dorothy Denning, Deb Frincke, and Dick Kemmerer (all panelists were program-committee chairs during the 1980s). He also was on an awards committee that selected the most influential SSP papers from 1980 to 1994, whose authors were honored as part of the 40th-anniversary celebration. The session is captured online. (Peter is the only one present at the first SSP in 1980 who still attends – having been registered at nine of the first ten, each of the most recent 10, and many in between.)

B Abstracts of CTSRD Reports

B.1 An Introduction to CHERI

https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-941.pdf

Robert N. M. Watson, Simon W. Moore, Peter Sewell, and Peter G. Neumann.

This technical report [19] presents a high-level summary that had been requested by ICSF, and is also useful for some of our prospective adopters:

CHERI (Capability Hardware Enhanced RISC Instructions) extends conventional processor Instruction-Set Architectures (ISAs) with *architectural capabilities* to enable fine-grained memory protection and highly scalable software compartmentalization. CHERI's hybrid capability-system approach allows architectural capabilities to be integrated cleanly with contemporary RISC architectures and microarchitectures, as well as with MMU-based C/C++-language software stacks.

CHERI's capabilities are unforgeable tokens of authority, which can be used to implement both explicit pointers (those declared in the language) and implied pointers (those used by the runtime and generated code) in C and C++. When used for C/C++ memory protection, CHERI directly mitigates a broad range of known vulnerability types and exploit techniques. Support for more scalable software compartmentalization facilitates software mitigation techniques such as sandboxing, which also defend against future (currently unknown) vulnerability classes and exploit techniques.

We have developed, evaluated, and demonstrated this approach through hardware-software prototypes, including multiple CPU prototypes, and a full software stack. This stack includes an adapted version of the Clang/LLVM compiler suite with support for capability-based C/C++, and a full UNIX-style OS (CheriBSD, based on FreeBSD) implementing spatial, referential, and temporal memory safety. Formal modeling and verification allow us to make strong claims about the security properties of CHERI-enabled architectures.

This report is a high-level introduction to CHERI: Capability Hardware Enhanced RISC Instructions. The report describes our architectural approach, CHERI's key microarchitectural implications, our approach to formal modeling and proof, the CHERI software model, our software-stack prototypes, further reading, and potential areas of future research.

B.2 CHERI Instruction-Set Architecture (Version 7), June 2019

Full title: Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 7) https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-927.pdf

Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin, David Chisnall, Brooks Davis, Nathaniel Wesley Filardo, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Simon W. Moore, Steven J. Murdoch, Kyndylan Nienhuis, Robert Norton, Alex Richardson, Peter Rugg, Peter Sewell, Stacey Son, and Hongyan Xia. (Authorship includes current and past primary contributors, most of whom are still active.)

This technical report [20] describes CHERI ISAv7, the seventh version of the Capability Hardware Enhanced RISC Instructions (CHERI) Instruction-Set Architecture (ISA) being developed by SRI International and the University of Cambridge. This design captures seven years of research, development, experimentation, refinement, formal analysis, and validation through hardware and software implementation. CHERI ISAv7 is a substantial enhancement to prior ISA versions. We differentiate an architecture-neutral protection model versus architecture-specific instantiations in 64-bit CHERI-MIPS, 64-bit and 32-bit CHERI-RISC-V, and conceptually x86_64. CHERI-MIPS supports special-purpose capability registers. CHERI-RISC-V is substantially elaborated. A new compartment-ID register assists in resisting microarchitectural side-channel attacks. Experimental features include linear capabilities, capability coloring, temporal memory safety, and 64-bit capabilities for 32-bit architectures. (See a related short paper by Aaron Lippeveldts in Belgium, Linear capabilities for CHERI: an exploration of the design space [11], proposing an ISA extension for CHERI.)

CHERI is a *hybrid capability-system architecture* that adds new capability-system primitives to a commodity 64-bit RISC hardware ISA, enabling software to efficiently implement *fine-grained memory protection* and *scalable software compartmentalization*. Design goals have included incremental adoptability within current ISAs and software stacks, low performance overhead for memory protection, significant performance improvements for software compartmentalization, formal grounding, and programmer-friendly underpinnings. We have focused on providing strong and efficient architectural foundations for the principles of *least privilege* and *intentional use* in the execution of software at multiple levels of abstraction, preventing and mitigating vulnerabilities.

The CHERI system architecture purposefully addresses known performance and robustness gaps in commodity ISAs that hinder the adoption of more secure programming models centered around the principle of least privilege. To this end, CHERI blends traditional paged virtual memory with an in-address-space capability model that includes capability registers, capability instructions, and tagged memory. CHERI builds on the C-language fat-pointer literature: its capabilities can describe fine-grained regions of memory, and can be substituted for data or code pointers in generated code, protecting data and also improving control-flow robustness. Strong capability integrity and monotonicity properties allow the CHERI model to express a variety of protection properties, from enforcing valid C-language pointer provenance and bounds checking to implementing the isolation and controlled communication structures required for software compartmentalization.

CHERI's hybrid capability-system approach, inspired by the Capsicum security model, allows incremental adoption of capability-oriented design: software implementations that are more robust and resilient can be deployed where they are most needed, while leaving less critical software largely unmodified, but nevertheless suitably constrained to be incapable of having adverse effects. Potential deployment scenarios include low-level software Trusted Computing Bases (TCBs) such as separation kernels, hypervisors, and operating-system kernels, as well as userspace TCBs such as language runtimes and web browsers. We also see potential early-use scenarios around particularly high-risk software libraries (such as

data compression, protocol parsing, and image processing), which are concentrations of both complex and historically vulnerability-prone code exposed to untrustworthy data sources, while leaving containing applications unchanged.

The preparation of this document was funded at SRI and Cambridge solely by CTSRD until mid-2018, when MTO ECATS support was also used to fund some of the authors.

The report grew steadily throughout the CTSRD project. The released Version 7 has 543 pages in letter format, and 496 pages in A4 format. It is likely to continue to grow.

The chapters are enumerated as follows:

Version 7, Chapter 1: Introduction (excerpted)

CHERI (Capability Hardware Enhanced RISC Instructions) extends commodity RISC Instruction-Set Architectures (ISAs) with new capability-based primitives that improve software robustness to security vulnerabilities. The CHERI model is motivated by the *principle of least privilege*, which argues that greater security can be obtained by minimizing the privileges accessible to running software. A second guiding principle is the *principle of intentional use*, which argues that, where many privileges are available to a piece of software, the privilege to use should be explicitly named rather than implicitly selected. While CHERI does not prevent the expression of vulnerable software designs, it provides strong *vulnerability mitigation*: attackers have a more limited vocabulary for attacks, and should a vulnerability be successfully exploited, they gain fewer rights, and have reduced access to further attack surfaces. CHERI allows software privilege to be minimized at two granularities:

- **Fine-grained code protection:** CHERI supports *fine-grain protection* and *intentional use* through in-address-space *memory capabilities*, which replace integer virtual-address representations of code and data pointers. The aim here is to minimize the rights available to be exercised on an instruction-by-instruction basis, limiting the scope of damage from inevitable software bugs. CHERI capabilities protect the integrity and valid provenance of pointers themselves, as well as allowing fine-grained protection of the in-memory data and code that pointers refer to. These protection policies can, to a large extent, be based on information already present in program descriptions – e.g., from C-language types, memory allocators, and run-time linking. This application of least privilege and intentional use provides strong protection against a broad range of memory- and pointer-based vulnerabilities and exploit techniques – buffer overflows, format-string attacks, pointer injection, data-pointer-corruption attacks, control-flow attacks, and so on. Many of these goals can be achieved through code recompilation on CHERI.

- **Secure encapsulation:** At a coarser granularity, CHERI also supports *secure encapsulation* and *intentional use* through the robust and efficient implementation of highly scalable in-address-space *software compartmentalization* using *object capabilities*. The aim here is to minimize the set of rights available to larger isolated software components, building on efficient architectural support for strong software encapsulation. These protections are grounded in explicit descriptions of isolation and communication provided by software authors, such as through explicit software sandboxing. This application of least privilege and intentional use provides strong mitigation of application-level vulnerabilities, such as logical errors, downloaded malicious code, or software Trojans inserted in the software supply chain.

  Effective software compartmentalization depends on explicit software structure, and can require significant code change. Where compartmentalization already exists in software, CHERI can be used to significantly improve compartmentalization performance and granularity. Where that structure is not yet present, CHERI can improve the adoption path for compartmantalization due to supporting in-address-space compartmentalization models.

CHERI is designed to support incremental adoption within current security-critical, C-language *Trusted Computing Bases (TCBs)*: operating-system (OS) kernels, key system libraries and services, language runtimes supporting higher-level type-safe languages, and applications such as web browsers and office suites. While CHERI builds on many historic ideas about capability systems (see Chapter 12), one of the key contributions of this work is CHERI's *hybrid capability-system architecture*. In this context, *hybrid* refers to combining aspects from conventional architectures, system software, and language/compiler choices with capability-oriented design. Key forms of hybridization in the CHERI design include:

A RISC capability system A capability-system model is blended with a conventional RISC user-mode architecture without disrupting the majority of key RISC design choices.

An MMU-enabled capability system A capability-system model is cleanly and usefully composed with conventional ring-based privilege and virtual memory based on MMUs (Memory Management Units).

A C-language capability system CHERI can be targeted by a C/C++-language compiler with strong compatibility, performance, and protection properties.

Hybrid system software CHERI supports a range of OS models including conventional MMU-based virtual-memory designs, hybridized designs that host capability-based software within multiple virtual address spaces, and pure single-address-space capability systems.

Incremental adoptability Within pieces of software, capability-aware design can be disregarded, partially adopted, or fully adopted with useful and predictable semantics. This allows incremental adoption within large software bases, from OS kernels to application programs.

We hope that these hybrid aspects of the design will support gradual deployment of CHERI features in existing software, rather than obliging a clean-slate software design, thereby offering a more gentle hardware-software adoption path.

In the remainder of this chapter, we describe our high-level design goals for CHERI, the notion that CHERI is an architecture-neutral protection model with architecture-specific mappings (such as CHERI-MIPS and CHERI-RISC-V), an introduction to the CHERI-MIPS concrete instantiation, a brief version history, an outline of the remainder of this report, and our publications to date on CHERI. A more detailed discussion of our research methodology, including motivations, threat model, and evolving approach from ISA-centered prototyping to a broader architecture-neutral protection model may be found in Chapter 11. Historical context and related work for CHERI may be found in Chapter 12. The glossary at the end of the report contains stand-alone definitions of many key ideas and terms, and may be useful reference material when reading the report.

CHERI Design Goals

CHERI has three central design goals aimed at dramatically improving the security of contemporary C-language TCBs, through processor support for fine-grained memory protection and scalable software compartmentalization, whose (at times) conflicting requirements have required careful negotiation in our design:

Fine-grained memory protection improves software resilience to escalation paths that allow low-level software bugs involving individual data structures and data-structure manipulations to be coerced into more powerful software vulnerabilities; e.g., through remote code injection via buffer overflows, control-flow and data-pointer corruption, and other memory-based techniques. Unlike MMU-based memory protection, CHERI

memory protection is intended to be driven by the compiler in protecting programmer-described data structures and references, rather than via coarse page-granularity protections. CHERI capabilities limit how pointers can be used by scoping the ranges of memory (via bounds) and operations that can be performed (via permissions). They also protect the integrity, provenance, and monotonicity of pointers in order to prevent inadvertent or inappropriate manipulation that might otherwise lead to privilege escalation.

Memory capabilities may be used to implement data pointers (protecting against a variety of data-oriented vulnerabilities such as overflowing buffers) and also to implement code pointers (supporting the implementation of control-flow integrity by preventing corrupted code pointers and return addresses from being used). Fine-grained protection also provides the foundation for expressing compartmentalization within application instances. We draw on, and extend, ideas from recent work in C-language *software bounds checking* by combining *fat pointers* with capabilities, allowing capabilities to be substituted for C pointers with only limited changes to program semantics.

CHERI permits efficient implementation of dialects of C and C++ in which various invalid accesses, deemed to be undefined behavior in those languages, and potentially giving arbitrary behavior in their implementations, are instead guaranteed to throw an exception.

Software compartmentalization involves the decomposition of software (at present, primarily application software) into isolated components to mitigate the effects of security vulnerabilities by applying sound principles of security, such as abstraction, encapsulation, type safety, and especially least privilege and the minimization of what must be trustworthy (and therefore sensibly trusted!). Previously, it seems that the adoption of compartmentalization has been limited by a conflation of hardware primitives for virtual addressing and separation, leading to inherent performance and programmability problems when implementing fine-grained separation. Specifically, we seek to decouple the virtualization from separation to avoid scalability problems imposed by MMUs based on translation look-aside buffers (TLBs), which impose a very high performance penalty as the number of protection domains increases, as well as complicating the writing of compartmentalized software.

A viable transition path must be applicable to current software and hardware designs. CHERI hardware must be able to run most current software without significant modification, and allow incremental deployment of security improvements starting with the most critical software components: the TCB foundations on which the remainder of the system rests, and software with the greatest exposure to risk. CHERI's features must significantly improve security, to create demand for upstream processor manufacturers from their downstream mobile and embedded device vendors. These CHERI features must at the same time conform to vendor expectations for performance, power use, and compatibility to compete with less secure alternatives.

We draw on *formal methodologies* wherever feasible, to improve our confidence in the design and implementation of CHERI. This use is necessarily subject to real-world constraints of timeline, budget, design process, and prototyping, but it has helped increase our confidence that CHERI meets our functional and security requirements. Formal methods can also help to avoid many of the characteristic design flaws that are common in both hardware and software. This desire requires us not only to perform research into CPU and software design, but also to develop new formal methodologies, and adaptations and extensions of existing ones.

We are concerned with satisfying the need for trustworthy systems and networks, where *trustworthiness* is a multidimensional measure of how well a system or other entity satisfies its various requirements – such as those for security, system integrity, and reliability, as well as human safety, and total-system survivability, robustness, and resilience, notably in the presence of a wide range of adversities such as hardware failures, software flaws, malware, accidental and intentional misuse, and so on. Our approach to trustworthiness encompasses hardware and software architecture, dynamic and static evaluation, formal and non-formal analyses, good software-engineering practices, and much more.

Architecture Neutrality and Architectural Instantiations

CHERI consists of an architectural-neutral protection model, and a set of instantiations of that model across multiple ISAs. Our initial mapping into the 64-bit MIPS ISA has allowed us to develop the CHERI approach; we have now expanded to include a more elaborated mapping into the 64-bit RISC-V ISA, and a sketch mapping into the x86-64 ISA. In doing so, we have attempted to maximize the degree to which specification is architecture neutral, and minimize the degree to which it is architecture specific. Even within a single ISA, there are multiple potential instantiations of the CHERI protection model, which offer different design tradeoffs – for example, decisions about whether to have separate integer and capability register files or to merge them into a single register file.

The successful mapping into multiple ISAs has led us to believe that the CHERI protection model is a portable protection model, that support portable software stacks in much the same way that portable virtual-memory-based operating systems can be implemented across a variety of architectural MMUs. Unlike MMUs, whose software interactions are primarily with the operating system, CHERI interacts directly with compiler-generated code, key system libraries, compartmentalization libraries, and applications; across all of these, we have found that an architecture-neutral approach can be highly effective, offering portability to the vast majority of CHERI-aware C/C++ code. We first consider the architecture-neutral model, and then applications of our approach in specific ISAs.

The Architecture-Neutral CHERI Protection Model

The aim of the CHERI protection model, as embodied in both the software stack (see Chapter 1) and architecture (see Chapter 3), is to support two vulnerability mitigation objectives: first, fine-grained pointer and memory protection within address spaces, and second,

primitives to support both scalable and programmer-friendly compartmentalization within address spaces. The CHERI model is designed to support low-level TCBs, typically implemented in C or a C-like language, in workstations, servers, mobile devices, and embedded devices. In contrast to MMU-based protection, this is done by protecting *references to code and data* (pointers), rather than the *location of code and data* (virtual addresses). This is accomplished via an *in-address-space capability-system model*: the architecture provides a new primitive, the *capability*, that software components (such as the OS, compiler, run-time linker, compartmentalization runtime, heap allocator, etc.) can use to implement strongly protected pointers within virtual address spaces.

In the security literature, capabilities are tokens of authority that are unforgeable and delegatable. *CHERI capabilities* are integer virtual addresses that have been extended with metadata to protect their integrity, limit how they are manipulated, and control their use. This metadata includes a *tag* implementing strong integrity protection (differentiating valid and invalid capabilities), *bounds* limiting the range of addresses that may be dereferenced, *permissions* controlling the specific operations that may be performed, and also *sealing*, used to support higher-level software encapsulation. Protection properties for capabilities include the ISA ensuring that capabilities are always derived via valid manipulations of other capabilities (*provenance*), that corrupted in-memory capabilities cannot be dereferenced (*integrity*), and that rights associated with capabilities are non-increasing (*monotonicity*).

CHERI capabilities may be held in registers or in memories, and are loaded, stored, and dereferenced using CHERI-aware instructions that expect capability operands rather than integer virtual addresses. On hardware reset, initial capabilities are made available to software via special and general-purpose capability registers. All other capabilities will be derived from these initial valid capabilities through valid capability transformations.

In order to continue to support non-CHERI-aware code, dereference of integer virtual addresses via legacy instruction is transparently indirected via a *default data capability* (DDC) for loads and stores, or a *program-counter capability* (PCC) for instruction fetch.

A variety of programming-language and code-generation models can be used with a CHERI-extended ISA. As integer virtual addresses continue to be supported, C or C++ compilers might choose to always implement pointers via integers, selectively implement certain pointers as capabilities based on annotations or type information (i.e., a *hybrid C* interpretation), or alternatively always implement pointers as capabilities except where explicitly annotated (i.e., a *pure-capability* interpretation). Programming languages may also employ capabilities internal to their implementation: for example, to protect return addresses, vtable pointers, and other virtual addresses for which capability protection can provide enhanced vulnerability mitigation.

When capabilities are being used to implement pointers (e.g., to code or data) or internal addresses (e.g., for return addresses), they must be constructed with suitably restricted rights, to accomplish effective protection. This is a run-time operation performed using explicit instructions (e.g., to set bounds, mask permissions, or seal capabilities) by the operating system, run-time linker, language runtime and libraries, and application code itself:

The operating-system kernel may narrow bounds and permissions on pointers provided

as part of the start-up environment when executing a program binary (e.g., to arguments or environmental variables), or when returning pointers from system calls (e.g., to new memory mappings).

The run-time linker may narrow bounds and permissions when setting up code pointers or pointers to global variables.

The system library may narrow bounds and permissions when returning a pointer to newly allocated heap memory.

The compartmentalization runtime may narrow bounds and permissions, as well as seal capabilities, enforcing compartment isolation (e.g., to act as sandboxes).

The compiler may insert instructions to narrow bounds and permissions when generating code to take a pointer to a stack allocation, or when taking a pointer to a field of a larger structure allocated as a global, on the stack, or on the heap.

The language runtime may narrow bounds and permissions when returning pointers to newly allocated objects, or when setting up internal linkage, as well as seal capabilities to non-dereferenceable types.

The application may request changes to permissions, bounds, and other properties on pointers, in order to further subset memory allocations and control their use.

The CHERI model can also be used to implement other higher-level protection properties. For example, tags on capabilities in memory can be used to support accurate C/C++-language temporal safety via revocation or garbage collection, and sealed capabilities can be used to enforce language-level encapsulation and type-checking features. The CHERI protection model and its implications for software security are described in detail in Chapter 2).

CHERI is an *architecture-neutral protection model* in that, like virtual memory, it can be deployed within multiple ISAs. In developing CHERI, we initially considered it as a concrete extension to the 64-bit MIPS ISA; using it, we could explore the implications downwards into the microarchitecture, and upwards into the software stack. Having developed a mature hardware-software protection model, we used this as the baseline in deriving an architecture-neutral CHERI protection model. This architecture-neutral model is discussed in detail in Chapter 3. We have demonstrated the possibility of adding CHERI protection to more than one base ISA by providing a detailed concrete instantiation for the 64-bit MIPS ISA (Chapter 7), a draft instantiation in the RISC-V ISA (Chapter 5), and a lightweight architectural sketch for the x86-64 ISA (Chapter 6).

An Architecture-Specific Mapping into 64-bit MIPS

The CHERI-MIPS ISA (see Chapter 4) is an instantiation of the CHERI protection model as an extension to the 64-bit MIPS ISA [9]. CHERI adds the following features to the

MIPS ISA[1] to support granular memory protection and compartmentalization within address spaces:

Capability registers describe the rights (*protection domain*) of the executing thread to access memory, and to invoke object references to transition between protection domains. We model these registers as a separate *capability register file*, supplementing the general-purpose integer register file.

Capability registers contain a tag, object type, permission mask, base, length, and offset (allowing the description of not just a bounded region, but also a pointer into that region, improving C-language compatibility). Capability registers are suitable for describing both data and code, and can hence protect both data integrity/confidentiality and control flow. Certain registers are reserved for use in exception handling; all others are available to be managed by the compiler using the same techniques used with conventional registers. Over time, we imagine that software will increasingly use capabilities rather than integers to describe data and object references.

Another potential integration into the ISA (which would maintain the same CHERI protection semantics) would be to extend the existing general-purpose integer registers so that they could also hold capabilities. This might reduce the hardware resources required to implement CHERI support. However, we selected our current approach to maintain consistency with the MIPS ISA extension model (in which coprocessors have independent register files), and to minimize *Application Binary Interface (ABI)* disruption on boundaries between legacy and CHERI-aware code for the purposes of rapid architectural and software iteration. We explore the potential space of mappings from the CHERI model into the ISA in greater detail in Section 3.10, as well as in Chapters 5 and 6 where we consider alternative mappings into non-MIPS ISAs.

Capability instructions allow executing code to create, constrain (e.g., by reducing bounds or permissions), manage, and inspect capability register values. Both unsealed (memory) and sealed (object) capabilities can be loaded and stored via memory capability registers (i.e., dereferencing). Object capabilities can be invoked, via special instructions, allowing a transition between protection domains, but are *immutable* and *non-dereferenceable*, providing encapsulation of the code or data that they refer to. Capability instructions implement *guarded manipulation*: invalid capability manipulations (e.g., to increase rights or length) and invalid capability dereferences (e.g., to access outside of a bounds-checked region) result in an exception that can be handled by the supervisor or language runtime. A key aspect of the instruction-set design is *intentional use of capabilities*: explicit capability registers, rather than ambient authority, are used to indicate exactly which rights should be exercised, to limit the damage that can be caused by exploiting bugs. Tradeoffs exist around intentional use, and in some

[1]Formally, CHERI instructions are added to MIPS as a *MIPS coprocessor* – a reservation of opcode space intended for third-party use. Despite the suggestive term "coprocessor", CHERI support will typically be integrated tightly into the processor pipeline, memory subsystem, and so on. We therefore eschew use of the term.

cases compatibility or opcode utilization may dictate implicit capability selection; for example, legacy MIPS load and store instructions implicitly dereference a Default Data Capability as they are unable to explicitly name a capability register. Most capability instructions are part of the user-mode ISA, rather than the privileged ISA, and will be generated by the compiler to describe application data structures and protection properties.

Tagged memory associates a 1-bit tag with each capability-aligned and capability-sized word in physical memory, which allows capabilities to be safely loaded and stored in memory without loss of integrity. Writes to capability values in memory that do not originate from a valid capability in the capability register file will clear the tag bit associated with that memory, preventing accidental (or malicious) dereferencing of invalid capabilities.

This functionality expands a thread's effective protection domain to include the transitive closure of capability values that can be loaded via capabilities via those present in its register file. For example, a capability register representing a C pointer to a data structure can be used to load further capabilities from that structure, referring to further data structures, which could not be accessed without suitable capabilities.

Non-bypassable tagging of unforgeable capabilities enables not only reliable and secure enforcement of capability properties, but also reliable and secure identification of capabilities in memory for the purposes of implementing other higher-level protection properties such as temporal safety.

In keeping with the RISC philosophy, CHERI instructions are intended for use primarily by the operating system and compiler rather than directly by the programmer, and consist of relatively simple instructions that avoid (for example) combining memory access and register value manipulation in a single instruction. In our current software prototypes, there are direct mappings from programmer-visible C-language pointers to capabilities in much the same way that conventional code generation translates pointers into general-purpose integer register values; this allows CHERI to continuously enforce bounds checking, pointer integrity, and so on. There is likewise a strong synergy between the capability-system model, which espouses a separation of policy and mechanism, and RISC: CHERI's features make possible the implementation of a wide variety of OS, compiler, and application-originated policies on a common protection substrate that optimizes fast paths through hardware support.

Our prototype of this approach, instantiating our ideas about CHERI capability access to a specific instruction set (the 64-bit MIPS ISA) has necessarily led to a set of congruent implementation decisions about register-file size, selection of specific instructions, exception handling, memory alignment requirements, and so on, that reflect that starting-point ISA. These decisions might be made differently with another starting-point ISA as they are simply surface features of the underlying approach; we anticipate that adaptations to ISAs such as ARM, RISC-V, and x86-64 would adopt instruction-encoding conventions, and so on, more in keeping with their specific flavor and design (see Chapters 5 and 6).

Other design decisions reflect the goal of creating a platform for prototyping and exploring the design space itself; among other choices, this includes the initial selection of 256-bit capabilities, giving us greater flexibility to experiment with various bounds-checking and capability behaviors. However, a 256-bit capability introduces potentially substantial cache overhead for pointer-intensive applications – so we have also developed a "compressed" 128-bit in-memory representation. This approach exploits redundancy between the virtual address represented by a capability and its lower and upper bounds – but necessarily limits granularity, leading to stronger alignment requirements.

In our CHERI-MIPS prototype implementation of the CHERI model, capability support is tightly coupled with the existing processor pipeline: instructions propagate values between general-purpose integer registers and capability registers; capabilities transform interpretation of virtual addresses generated by capability-unaware instructions including by transforming the program counter; capability instructions perform direct memory stores and loads both to and from general-purpose integer registers and capability registers; and capability-related behaviors deliver exceptions to the main pipeline. By virtue of having selected the MIPS-centric design choice of exposing capabilities as a separate set of registers, we maintain a separate capability register file as an independent hardware unit – in a manner comparable to vector or floating-point units in current processor designs. The impacts of this integration include additional control logic due to maintaining a separate register file, and a potentially greater occupation of opcode space, whereas combining register files might permit existing instructions to be reused (with care) across integer and capability operations.

Wherever possible, CHERI systems make use of existing hardware designs: processor pipelines and register files, cache memory, system buses, commodity DRAM, and commodity peripheral devices such as NICs and display cards. We are currently focusing on enforcement of CHERI security properties on applications running on a general-purpose processor; in future work, we hope to consider the effects of implementing CHERI in peripheral processors, such as those found in Network Interface Cards (NICs) or Graphical Processing Units (GPUs).

Architectural Neutrality: CHERI-RISC-V and CHERI-x86-64

We believe that the higher-level memory protection and security models we describe encompass not only a number of different potential expressions within a single ISA (e.g., whether to have separate capability registers or to extend general-purpose integer registers to also optionally hold capabilities), but also be applied to other RISC (and CISC) ISAs. This should allow reasonable source-level software portability (leaving aside language runtime and OS assembly code, and compiler code generation) across the CHERI model implemented in different architectures – in much the same way that conventional OS and application C code, as well as APIs for virtual memory, are moderately portable across underlying ISAs.

We have therefore developed two further mappings of the CHERI protection model into specific ISAs: CHERI-RISC-V (Chapter 5) and CHERI-x86-64 (Chapter 6). CHERI-RISC-V is a draft architecture that we are in the process of defining and implementing: RISC-V derives many of it foundational design choices from MIPS, with some more contemporary

architectural choices such as hardware page-table walking, and the adaptation to CHERI is very similar. In some areas, we have chosen to leave open specific aspects of the design, learning from our work on CHERI-MIPS, to allow evaluation of performance tradeoffs – e.g., as relates to using a split or merged capability register file. CHERI-x86-64 is an architectural sketch that we have developed to better understand how the CHERI model might apply to more CISC instruction sets. Despite substantive underlying differences between x86-64 and MIPS, we find that many aspects of our approach carry through. We do not yet have implementation aims for CHERI-x86-64, although we hope to explore this further in the future.

Deterministic Protection

CHERI has been designed to provide strong, non-probabilistic protection rather than depending on short random numbers or truncated cryptographic hashes that can be leaked and re-injected, or that could be brute forced. Essential to this approach is using out-of-band memory tags that prevent confusion between data and capabilities. Software stacks can use these features to construct higher-level protection properties, such as preventing the transmission of pointers via Inter-Process Communication (IPC) or network communications. They are also an essential foundation to strong compartmentalization, which assumes a local adversary.

Formal Modeling and Provable Protection

The design process for CHERI has used formal semantic models as an important tool in various ways. Our goal here has been to understand how we can support the CHERI design and engineering process with judicious use of mathematically rigorous methods, both in lightweight ways (providing engineering and assurance benefits without the costs of full formal verification), and using machine-checked proof to establish high confidence that the architecture design provides specific security properties.

The basis for all this has been use of formal specifications of the ISA instruction behavior as a fundamental design tool, initially for CHERI-MIPS in L3 [6], and now for CHERI-MIPS and CHERI-RISC-V in Sail [2]. L3 and Sail are domain-specific languages specifically designed for expressing instruction behavior, encoding data, etc. Simply moving from the informal pseudocode commonly used to describe instruction behavior to parsed and type-checked artifacts already helps maintain clear specifications. The CHERI-MIPS instruction descriptions in Chapter 7 are automatically included from the Sail model, keeping documentation and model in sync.

Both L3 and Sail support automatic generation of executable models (variously in SML, OCaml, or C) from these specifications. These executable models have been invaluable, both as golden models for testing our hardware prototypes, and as emulators for testing CHERI software above. The fact that they are automatically generated from the specifications again helps keep things in sync, enabling regression testing on any change to the specification, and makes for easy experimentation with design alternatives. The generated emulators run fast

enough to boot FreeBSD in a few minutes (booting cheribsd currently takes around 250s, roughly 320kips).

We have also used the models to automatically generate ISA test cases, both via simple random instruction generation, and using theorem-prover and SMT approaches [3].

Finally, the models support formal verification, with mechanized proof, of key architectural security properties. L3 and Sail support automatic generation of versions of the models in the definition languages of (variously) the HOL4, Isabelle, and Coq theorem provers, which we have used as a basis for proofs. Key architectural verification goals including proving not just low-level properties, such as the monotonicity of each individual instruction and properties of the CHERI Concentrate compression scheme, but also higher-level goals such as compartment monotonicity, in which arbitrary code sequences isolated within a compartment are unable to construct additional rights beyond those reachable either directly via the register file or indirectly via loadable capabilities. We have proved a number of such properties about the CHERI-MIPS ISA, to be documented in future papers and reports.

The CHERI design process has also been enhanced by interactions with our work on rigorous semantics for C [14, 13].

Subsequent drafts (e.g., Version 8-Alpha-2, 30 September 2019) will be available to DARPA on request, prior to the next version that requires approval for public release (because of commingled authors also funded by ECATS).

B.3 CHERI: Notes on the Meltdown and Spectre Attacks

Full title: Capability Hardware Enhanced RISC Instructions (CHERI): Notes on the Meltdown and Spectre Attacks

https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-916.pdf

Robert N. M. Watson, Jonathan Woodruff, Michael Roe, Simon Moore, and Peter G. Neumann

In this report [22], we consider the potential impact of recently announced Meltdown and Spectre microarchitectural side-channel attacks arising out of superscalar (out-of-order) execution on Capability Hardware Enhanced RISC Instructions (CHERI) computer architecture. We observe that CHERI's in-hardware permissions and bounds checking may be an effective form of mitigation for one variant of these attacks, in which speculated instructions can bypass software bounds checking. As with MMU-based techniques, CHERI remains vulnerable to side-channel leakage arising from speculative execution across compartment boundaries, leading us to propose a software-managed compartment ID to mitigate these vulnerabilities for other variants as well. (This paper had support from CTSRD, but not ECATS.)

The chapters of this report are as follows:

```
CHERI and Microarchitectural Side-Channel Attacks:
1 Background
2 Applicability to the CHERI Architecture
```

B.4 CHERI Programmer's Guide

Version 1.15, September 2019 (This document is compatible with the CHERI ISA architect document, Version 7.)

Robert N. M. Watson, John Baldwin, David Chisnall, Brooks Davis, Khilan Gudka, Wojciech Koszek, Simon W. Moore, Steven J. Murdoch, Edward Napierala, Peter G. Neumann, Alex Richardson, Stacey Son, Andrew Turner, and Jonathan Woodruff

Abstract: The *CHERI Programmer's Guide* documents the programming models, software, and development environment for the Capability Hardware Enhanced RISC Instructions (CHERI) architecture developed by SRI International and the University of Cambridge. The Guide is targeted at hardware and software developers developing and working with capability-enhanced software. We explore CHERI's implications for both "hybrid" and "pure-capability" C/C++ code. We provide detailed implementation information about our CHERI Clang/LLVM compiler suite and related toolchain such as linker and debugger, intended to assist with understanding our changes as well as applying similar changes to other compiler and toolchain suites. We similarly describe our CheriBSD operating system and its use of CHERI features, as well as its support for the CheriABI pure-capability process environment. Finally, we describe the Qemu-CHERI ISA-level emulator, which is able to boot and run CheriBSD.

The chapters of this report are enumerated as follows:

B.5 An Introduction to Pure-Capability CHERI C/C++ Programming

Robert N. M. Watson, Brooks Davis, and Alexander Richardson, Working draft, June 2019

This document is a brief introduction to the pure-capability CHERI C/C++ programming languages. It describes the most commonly encountered differences between those languages on CHERI versus on conventional architectures, and where existing software may require minor changes. It also explains how modest language extensions allow selected software, such as memory allocators, to further refine permissions and bounds on pointers. This guidance is based on the experience of adapting the FreeBSD operating-system userspace, and applications such as PostgreSQL and WebKit, to run in a pure-capability programming environment.

The preparation of this document was funded at SRI by CTSRD, and at Cambridge by both CTSRD and ECATS. It was intended exclusively for potential CHERI adopters on a relatively short fuse, and has now been merged into the CHERI Programmer's Guide.

B.6 CheriABI Extended Report

https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-932.pdf

This is a considerably extended version of the CheriABI paper for ASPLOS 2019 [4].

Brooks Davis, Robert N. M. Watson, Alexander Richardson, Peter G. Neumann, Simon W. Moore, John Baldwin, David Chisnall, James Clarke, Nathaniel Wesley Filardo, Khilan Gudka, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, J. Edward Maste, Alfredo Mazzinghi, Edward Tomasz Napierala, Robert M. Norton, Michael Roe, Peter Sewell, Stacey Son, and Jonathan Woodruff

Abstract: The CHERI architecture allows pointers to be implemented as capabilities (rather than integer virtual addresses) in a manner that is compatible with, and strengthens, the semantics of the C language. In addition to the spatial protections offered by conventional fat pointers, CHERI capabilities offer strong integrity, enforced provenance validity, and access monotonicity. The stronger guarantees of these architectural capabilities must be reconciled with the real-world behavior of operating systems, run-time environments, and applications. When the process model, user-kernel interactions, dynamic linking, and memory management are all considered, we observe that simple derivation of architectural capabilities is insufficient to describe appropriate access to memory. We bridge this conceptual gap with a notional *abstract capability* that describes the accesses that should be allowed at a given point in execution, whether in the kernel or userspace. To investigate this notion at scale, we describe the first adaptation of a full C-language operating system (FreeBSD) with

an enterprise database (PostgreSQL) for complete spatial and referential memory safety. We show that awareness of abstract capabilities, coupled with CHERI architectural capabilities, can provide more complete protection, strong compatibility, and acceptable performance overhead compared with the pre-CHERI baseline and software-only approaches. Our observations also have potentially significant implications for other mitigation techniques.

B.7 Trustworthy Total-System Integration: Exploration of Hidden Security Problems, and Potential Approaches for Resolving Them

This unreleased draft interim report represented a summary of our initial work on direct memory access, internal microcontrollers, and input-output.

Modern computer systems typically contain many microprocessors, input-output peripherals, and other active devices. Some of these components have undocumented direct memory access to the main processor(s) or other ways in which they could compromise the trustworthiness of the overall system. Of particular interest here are systems using laptops and mobile devices, as well as many kinds of servers and network support systems such as switches and controllers. All of these systems tend to share the problems of unknown and sometimes mysterious behavior resulting from the multitude of internal hardware components.

This report seeks to provide a sound basis for the design, development, and operation of trustworthy total-system architectures that encompass a variety of processors, microprocessors, peripherals, and other computational devices. In such systems, we seek compositions of these components that can eliminate or otherwise neutralize the threats resulting from many undesirable vulnerabilities – and also provide some predictable assurance of trustworthiness. This report enumerates many of the potential security flaws and attacks, and considers a wide variety of approaches for increasing total-system trustworthiness. It also assesses which of these approaches – architectural and otherwise – might realistically lead to much greater assurance than is presently possible. It considers somewhat cleaner-slate holistic approaches for long-term future systems, and also suggests some nearer-term improvements that might be potentially increase the trustworthiness of today's architectures. There is of course a large spectrum between the clean-slate long-term approaches and the remedial short-term ones. We consider some of the primary tradeoffs within that spectrum. This interim version of the final report represents work still in progress. We expect to add considerable detail regarding the comparative evaluation of various architectural alternatives in the coming project year – based on our experience with the ongoing CHERI hardware technology transfer efforts.

This unreleased draft report provided the foundation for our paper, Thunderclap: Exploring Vulnerabilities in Operating-System IOMMU Protection via DMA from Untrustworthy Peripherals, Network and Distributed Systems Security (NDSS 2019), San Diego CA, 24-27 February 2019 [12], which received support from CTSRD and (later) ECATS.

B.8 Bluespec Extensible RISC Implementation (BERI): Hardware Reference

The *BERI Hardware Reference* [18] describes the Bluespec Extensible RISC Implementation (BERI) protoype developed by SRI International and the University of Cambridge. The reference is targeted at hardware and software developers working with the BERI1 and BERI2 processor prototypes in simulation and synthesized to FPGA targets. We describe how to use the BERI1 and BERI2 processors in simulation, the BERI1 debug unit, the BERI unit-test suite, how to use BERI with Altera FPGAs and Terasic DE4 boards, the 64-bit MIPS and CHERI ISAs implemented by the prototypes, the BERI1 and BERI2 processor implementations themselves, and the BERI Programmable Interrupt Controller (PIC).

B.9 Bluespec Extensible RISC Implementation (BERI): Software Reference

The *BERI Software Reference* [8] documents how to build and use the FreeBSD operating system on the Bluespec Extensible RISC Implementation (BERI) developed by SRI International and the University of Cambridge. The reference is targeted at hardware and software programmers who will work with BERI or BERI-derived systems.

B.10 CHERI Formal Methods

Version 3.0, 18 January 2016

This document describes strategies suitable for employing formal methods in the design of SRI International and University of Cambridge's Capability Hardware Enhanced RISC Instructions (CHERI) Instruction-Set Architecture (ISA), its total-system architecture, and its hardware-software implementation. The document has evolved from capturing our thoughts on what seemed desirable early in the research and development cycle into what has actually been accomplished during the course of the main five-year project, and what might be done in the future.

We discuss the overall design and its desired assurance properties, known weaknesses, and some mitigations provided by our approach. We present an initial formal consideration of the CHERI ISA, and automated test-case generation capability to check the formal model against its Bluespec SystemVerilog (BSV) implementation. We also describe the application of formal methods to link design goals and a BSV-based prototype of the CHERI processor, requiring the development of new tools to support the formal validation of BSV-based designs. We conclude with a summary of subsequent analyses that remain to be done. Although the desired formal analyses of the CHERI ISA, hardware, and low-level software could not be completed within the scope, time-scale, and funding of this project, we hope to be able to subsequently pursue the detailed analyses in the future under other funding alternatives.

This report is mentioned here primarily for historical reasons. It has been completely

supplanted by the subsequent work for the CIFV project, which will be reported independently.

B.11 Deimos– CHERI Demo Operating System from Mars

22 January 2012

Robert N. M. Watson, Jonathan Woodruff, Jonathan Anderson, Simon W. Moore, Steven J. Murdoch, Michael Roe, Philip Paeps, Peter G. Neumann.

Deimos is a demonstration microkernel operating system that uses the CHERI ISA's capability features to sandbox untrustworthy applications. For the purposes of this demonstration, the CHERI prototype CPU was implemented in the Terasic tPad platform using an Altera FPGA; the tPad includes a VGA touchscreen, which is used for the Deimos user interface. This report is largely historical in nature, reflecting our early experimentation with the CHERI ISA; Deimos as demonstrated in November 2011 is not able to run on current CHERI hardware due to ISA changes. It is also unable to take advantage of a number of contemporary CHERI features, such as access to a CHERI-aware compiler. This text previously appeared in a chapter of the CHERI User's Guide. (This interim document was never released for public consumption, and mentioned here only for historical reasons.)

B.12 Hardware Specifications and Formal Proofs

The Version 7 document above contains the full specification of our open-source CHERI MIPS ISA. That specification is the basis for Arm's CHERI-ARM-64 processor, which is proprietary. The ECATS CHERI-RISC-V specifications and the CHERI-ARM-M conceptual architecture are open-sourced and available on GITHUB. The CIFV ISA models and their emerging formal proofs are also available on GITHUB.

B.13 Monthly and Quarterly CTSRD Reports

In addition to the publications and released reports noted above, we have delivered 107 (monthly or quarterly) progress reports to I2O, each of which documented our progress during the month in considerable detail. Those reports are not public, in part because of the temporal sensitivity of the tech transfer with Arm, and in part because the reports since late 2017 also include references to our progress in the ECATS project, which is subject to Controlled Unclassified Information (CUI) constraints. Nevertheless, the entire set of CSTRD monthly and quarterly reports could be very useful as a carefully documented history of our nine-year CTSRD efforts, showing our detailed fine-grained month-by-month evolution and many of the problems encountered that had to be rectified along the way.

C List of CSTRD-related PhDs

PhD theses written under or otherwise related to the CTSRD project (denoted by an asterisk if in progress).

Hesham Almatary (CTSRD, ECATS) *
James Clarke (CTSRD, ECATS) *
Brett Gutstein, Cambridge (CTSRD, ECATS, Gates Cambridge Trust) *
Nirav Dave, MIT (CTSRD)
Lawrence Esswood, Cambridge (CTSRD and ARM) *
Alexandre Joannou, Cambridge (CTSRD) *
Robert Kovacsics, Cambridge (CTSRD and ARM) *
Alfredo Mazzinghi, Cambridge (CTSRD and Google) *
Robert Norton-Wright, Cambridge (CTSRD and EPSERC-REMS)
Alex Richardson, Cambridge (CTSRD and HP Enterprise) *
Colin Rothwell, Cambridge (CTSRD, funded by ARM) *
Richard Uhler, MIT (CTSRD)
Jonathan Woodruff, Cambridge (CTSRD)

D Acknowledgments

The authors of this report thank the following members of the CTSRD, MRC2, ECATS, CIFV, and REMS teams, our past and current research collaborators at SRI and Cambridge, as well as colleagues at other institutions who have in various ways made invaluable contributions during our work on CTSRD:

Sam Ainsworth, Hesham Almatary, Jonathan Anderson, Ross J. Anderson, John Baldwin, Graeme Barnes, Hadrien Barral, Thomas Bauereiss, Stuart Biles, Brooks Davis, Matthias Boettcher, David Brazdil, Ruslan Bukin, Brian Campbell, Gregory Chadwick, David Chisnall, James Clarke, Serban Constantinescu, Chris Dalton, Nirav Dave, Brooks Davis, Dominique Devriese, Lawrence Esswood, Nathaniel Wesley Filardo, Wedson Filho, Anthony Fox, Paul J. Fox, John Goodacre, Paul Gotch, Richard Grisenthwaite, Tom Grocutt, Khilan Gudka, Brett Gutstein, Jong Hun Han, Andy Hopper, Alex Horsman, Alexandre Joannou, Timothy Jones, Asif Khan, Myron King, Chris Kitching, Wojciech Koszek, Robert Kovacsics, Ben Laurie, Patrick Lincoln, Anil Madhavapeddy, Ilias Marinos, A. Theodore Markettos, Tim Marsland, Ed Maste, Alfredo Mazzinghi, Kayvan Memarian, Dejan Milojicic, Andrew W. Moore, Will Morland, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, Robert Norton-Wright, Kyndylan Nienhuis, Philip Paeps, Lucian Paul-Trifu, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, John Rushby, Hassen Saidi, Hans Petter Selasky, Andrew Scull, Muhammad Shahbaz, Bradley Smith, Lee Smith, Stacey Son, Ian Stark, Andrew Turner, Richard Uhler, Munraj Vadera, Jacques Vidrine, Hugo Vincent, Philip Withnall, Hongyan Xia, and Bjoern A. Zeeb.

We wish to thank past and current members of its external oversight group for significant support and feedback, particularly in the first five years of the project:

Lee Badger, Simon Cooper, Rance DeLong, Jeremy Epstein, Virgil Gligor, Li Gong, Mike Gordon, Steven Hand, Andrew Herbert, Warren A. Hunt Jr., Doug Maughan, Greg Morrisett, Brian Randell, Kenneth F. Shotting, Joe Stoy, Tom Van Vleck, and Samuel M. Weber. We also acknowledge the late David Wheeler and Paul Karger, both of whose conversations with the authors about the CAP computer and capability systems contributed to our thinking.

Finally, we are enormously grateful to Howie Shrobe, MIT professor and past DARPA CRASH and MRC program manager, who has offered both technical insight and support throughout this work, and whose wisdom began the journey summarized in this report. We are also grateful to Robert Laddaga, Stu Wagner, and Jonathan Smith, who in turn succeeded Howie in overseeing the CRASH program, John Launchbury (DARPA I2O office director), Dale Waters (DARPA AEO office director), Linton Salmon (DARPA SSITH program manager), Daniel Adams and Laurisa Goergen (DARPA I2O SETAs supporting the CRASH and MRC programs), and Marnie Dunsmore and John Marsh (DARPA MTO SETAs supporting the SSITH program). All of these individuals provided valuable assistance.

E List of Symbols, Abbreviations, and Acronyms

ASL Arm's preferred specification language for processor architectures.

ABI Application Binary Interface. CheriABI is the ABI for using the CHERI ISA directly.

BERI Bluespec Extensible RISC Implementation (BERI)

BSD Berkeley version of Unix. CHERI-BSD is the BSD-enhanced operating system that understands and properly uses the CHERI ISA.

BSL An experimental extension of BSV, allowing compilation of both hardware and software specifications

BSV Bluespec SystemVerilog, developed by Bluespec Inc. to enable the use of the BSV compiler to transform hardware specifications written in the BSV specification language into a form that can be executed on FPGAs or simulated.

CFI Control-Flow Integrity

CHERI Capability Hardware Enhanced RISC Instructions. This acronym is used with respect to the CHERI hardware Instruction-Set Architecture (ISA) and the CHERI system architecture, among other entities.

CIFV CHERI Instruction-Set Architecture Formal Verification, DARPA Contract FA8650-18-C-7809, 31 January 2018 to 20 February 2019.

CISC Architecture Complex Instruction-Set Computer (as opposed to RISC)

CRASH Clean-slate Resilient Adaptable and Secure Hosts. This is the DARPA program under which the CTSRD project was created and executed.

CTSRD CRASH-worthy Trustworthy Systems Research and Development, DARPA I2O project under contract FA8750-10-C-0237, September 2010 to September 2019.

ECATS Extending the CHERI Architecture for Trustworthiness in SSITH SSITH, DARPA project HR0011-18-C-0016, 21 Nov 2017 to 20 Feb 2021

FPGA Field-Programmable Gate Arrays enable meta-hardware implementations with the ability to execute an instruction-set architecture as if it were real hardware.

GDB An open-sourced debugger

GOT Global Offset Table

IOMMU Input-Output Memory Management Unit

ISA Instruction-Set Architecture (ISA). As opposed to a total-system architecture or an operating-system architecture, an ISA is a detailed definition (possibly formally defined, as in the CHERI ISA) of each instruction.

LLDB A debugger associated with C and C++ programs generated by LLVM. CHERI-LLDB is the CHERI-enhanced version that understands and properly uses the CHERI ISA.

LLVM Backend compiler for C and C++ programs, CHERI-LLVM is the CHERI-enhanced version that understands and properly uses the CHERI ISA.

MIPS (1) Million Instructions Per Second is a unit of measure for the speed of a processor. (2). MIPS is the name of a class of RISC instruction-set architecture processors. MIPS processors have been widely used, partly because the specifications are open-sourced.

MMU Memory Management Unit

MRC Mission-oriented Resilient Clouds, a companion DARPA program to CRASH, which ran from 2011 to 2015.

(MRC)^2 (or, lazily, (MRC)2, pronounced *MRC-squared*) Modular Research-based Composable trustworthy Mission-oriented Resilient Clouds. This SRI-Cambridge MRC project encompassed some clean-slate approaches to secure software-defined networking (SDN) and trustworthy cloud servers, among many other innovations and developed prototypes. Its 2015 final report (which now seems somewhat dated) is online: http://www.csl.sri.com/neumann/private/mrc2-report.pdf .

PLT Program Linkage Table

QEMU QEMU is an open-source generic emulation tool. In CTSRD It is used to model and execute CHERI ISAs.

RISC Architecture Reduced Instruction-Set Computer (as opposed to CISC)

SOAAP Security-Oriented Analysis of Application Programs

TCB Trusted Computing Base, Something upon which security depends that must be assumed to be trustworthy, whether it is or not.

TCP Transmission Control Protocol

TESLA Temporally Enhanced System Logic Assertions

TLB Translation Lookaside Buffer

References

[1] Jonathan Anderson, Robert N. M. Watson, David Chisnall, Khilan Gudka, Brooks Davis, and Ilias Marinos. TESLA: Temporally Enhanced System Logic Assertions. In *Proceedings of The 2014 European Conference on Computer Systems (EuroSys)*, Amsterdam, The Netherlands, April 2014.

[2] Alasdair Armstrong, Thomas Bauereiss, Brian Campbell, Alastair Reid, Kathryn E. Gray, Robert M. Norton, Prashanth Mundkur, Mark Wassell, Jon French, Christopher Pulte, Shaked Flur, Ian Stark, Neel Krishnaswami, and Peter Sewell. ISA semantics for ARMv8-A, RISC-V, and CHERI-MIPS. In ***POPL 2019**: Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages*, January 2019. Proc. ACM Program. Lang. 3, POPL, Article 71.

[3] Brian Campbell and Ian Stark. Randomised testing of a microprocessor model using SMT-solver state generation. *Sci. Comput. Program.*, 118:60–76, 2016.

[4] Brooks Davis, Robert N. M. Watson, Alexander Richardson, Peter Neumann, Simon Moore, John Baldwin, David Chisnall, James Clarke, Khilan Gudka, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Ed Maste, Edward Tomasz Napierala, Robert Norton, Michael Roe, Peter Sewell, Stacey Son, and Jonathan Woodruff. CheriABI: Enforcing valid pointer provenance and minimizing pointer privilege in the POSIX C run-time environment. In *Proceedings of the 24nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2019)*, April 2019.

[5] Brooks Davis, Robert N. M. Watson, Alexander Richardson, Peter Neumann, Simon Moore, John Baldwin, David Chisnall, James Clarke, Khilan Gudka, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Ed Maste, Edward Tomasz Napierala, Robert Norton, Michael Roe, Peter Sewell, Stacey Son, and Jonathan Woodruff. CheriABI: Enforcing valid pointer provenance and minimizing pointer privilege in the POSIX C runtime environment, extended report. Technical Report UCAM-CL-TR-932, University of Cambridge, Computer Laboratory, April 2019.

[6] Anthony Fox. Improved tool support for machine-code decompilation in hol4. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving: 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, pages 187–202, Cham, 2015. Springer International Publishing.

[7] Khilan Gudka, Robert N. M. Watson, Jonathan Anderson, David Chisnall, Brooks Davis, Ben Laurie, Ilias Marinos, Peter G. Neumann, and Alex Richardson. Clean Application Compartmentalization with SOAAP. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS 2015)*, October 2015.

[8] Khilan Gudka, Robert N.M. Watson, Jonathan Anderson, David Chisnall, Brooks Davis, Ben Laurie, Ilias Marinos, Steven J. Murdoch, Peter G. Neumann, and Alex Richardson. Clean application compartmentalization with SOAAP (extended version). Technical Report UCAM-CL-TR-873, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, December 2015.

[9] Joseph Heinrich. *MIPS R4000 User's Manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[10] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988.

[11] Aaron Lippeveldts. Linear capabilities for CHERI: an exploration of the design space. In *ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity – SPLASH Companion 2019*, pages 47–48, October 2019.

[12] A. Theodore Markettos, Colin Rothwell, Brett F. Guttstein, Allison Pearce, Peter G. Neumann, Simon W. Moore, and Robert N. M. Watson. Thunderclap: Exploiting operating-system IOMMU bypass vulnerabilities with DMA from malicious peripherals. In *Proceedings of the Network and Distributed Systems Symposium (NDSS)*, February 2019.

[13] Kayvan Memarian, Victor B. F. Gomes, Brooks Davis, Stephen Kell, Alexander Richardson, Robert N. M. Watson, and Peter Sewell. Exploring C semantics and pointer provenance. In ***POPL 2019**: Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages*, January 2019. Proc. ACM Program. Lang. 3, POPL, Article 67.

[14] Kayvan Memarian, Justus Matthiesen, James Lingard, Kyndylan Nienhuis, David Chisnall, Robert N.M. Watson, and Peter Sewell. Into the depths of C: elaborating the de facto standards. In *Proceedings of PLDI 2016*. ACM, June 2016.

[15] Peter G. Neumann. Fundamental Trustworthiness Principles in CHERI. In H. Shrobe, D. L. Shrier, and A. Pentland, editors, *New Solutions for Cybersecurity*, chapter 6. MIT Press/Connection Science, 2018.

[16] Kyndylan Nienhuis, Alexandre Joannou, Anthony Fox, Michael Roe, Thomas Bauereiss, Brian Campbell, Matthew Naylor, Robert M. Norton, Simon W. Moore, Peter G. Neumann, Ian Stark, Robert N. M. Watson, and Peter Sewell. Rigorous engineering for hardware security: formal modelling and proof in the CHERI design and implementation process. Technical Report UCAM-CL-TR-940, University of Cambridge, Computer Laboratory, September 2019.

[17] Robert N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway. Capsicum: Practical capabilities for Unix. In *Proceedings of the 19th USENIX Security Symposium*. USENIX, August 2010.

[18] Robert N. M. Watson, David Chisnall, Brooks Davis, Wojciech Koszek, Simon W. Moore, Steven J. Murdoch, Peter G. Neumann, and Jonathan Woodruff. Bluespec Extensible RISC Implementation (BERI): Software Reference. Technical Report UCAM-CL-TR-853, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, June 2014.

[19] Robert N. M. Watson, Simon W. Moore, Peter Sewell, and Peter G. Neumann. An Introduction to CHERI. Technical Report UCAM-CL-TR-941, University of Cambridge, Computer Laboratory, September 2019.

[20] Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin, David Chisnall, Brooks Davis, Nathaniel Wesley Filardo, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Simon W. Moore, Steven J. Murdoch, Kyndylan Nienhuis, Robert Norton, Alex Richardson, Peter Sewell, Stacey Son, and Hongyan Xia. Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 7). Technical Report UCAM-CL-TR-927, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, phone +44 1223 763500, May 2019.

[21] Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Jonathan Anderson, John Baldwin, David Chisnall, Brooks Davis, Alexandre Joannou, Ben Laurie, Simon W. Moore, Steven J. Murdoch, Robert Norton, Stacey Son, and Hongyan Xia. Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 6). Technical Report UCAM-CL-TR-907, University of Cambridge, Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom, phone +44 1223 763500, April 2017.

[22] Robert N. M. Watson, Jonathan Woodruff, Michael Roe, Simon W. Moore, and Peter G. Neumann. Capability Hardware Enhanced RISC Instructions (CHERI): Notes on the Meltdown and Spectre Attacks. Technical Report UCAM-CL-TR-916, University of Cambridge, Computer Laboratory, February 2018.

[23] Jonathan Woodruff, Alexandre Joannou, Hongyan Xia, Anthony Fox, Robert Norton, Thomas Bauereiss, David Chisnall, Brooks Davis, Khilan Gudka, Nathaniel W. Filardo, A. Theodore Markettos, Michael Roe, Peter G. Neumann, Robert N. M. Watson, and Simon W. Moore. CHERI concentrate: Practical compressed capabilities. *IEEE Transactions on Computers*, 2019.

Note: A comprehensive bibliography related to a wider background of CHERI-related efforts can be found in the CHERI version 7 ISA document [20].