Capabilities Revisited: A Holistic Approach to Bottom-to-Top Assurance of Trustworthy Systems

Peter G. Neumann and Robert N. M. Watson
SRI International Computer Science Laboratory and
University of Cambridge Computer Laboratory

Layered Assurance Workshop
Austin TX, 6-7 December 2010

Abstract: Long active in computer security, our two laboratories 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.

Introduction

SRI International's Computer Science Laboratory and the University of Cambridge Computer Laboratory have recently embarked on a project for DARPA's new CRASH program (Clean-slate design of Resilient, Adaptive, Survivable Hosts). Our CTSRD project (CRASH-worthy Trustworthy Systems R&D[1]) is summarized here, from the perspective of layered assurance. The essence of this project is to revisit the best of what we and our project collaborators have learned in the past half-century from our own and our institutions' participation in various system projects (e.g., hardware-software architectures such as Multics [15, 3], SRI's PSOS [13, 14], Cambridge's CAP [25], Karger and Herbert [7, 6], Gong [5], FreeBSD, TrustedBSD, Newcastle's Distributed Secure System [21], and, more recently, Capsicum [24], advances in separation kernels [18, 19, 20] and virtual-machine monitors, and extensive work in formal methods for software and hardware). Also relevant is the KeyKos/EROS history (e.g., [16, 22, 2]), seL4 [8], and other capability system efforts.

We are using this background to pursue a hardware-software co-design approach in which assurance is synergistic with the system architecture and can be provided from the hardware up through the lower-layer software, in such a way that desired system trustworthiness properties can be enforced architecturally to support a variety of applications. We augment a current Reduced Instruction Set Computing (RISC) Field Programmable Gate Array (FPGA) soft core with new capability system features. In particular, our Capability

[1]This effort is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237.

Hardware Enhanced RISC Instructions (CHERI) concept includes capability registers, capability instructions, and tagged memory. CHERI is designed to allow incremental adoption of higher-assurance approaches, with a focus on security-critical components making up contemporary Trusted Computing Bases (TCBs): the separation kernel / hypervisor, operating system kernel, and language runtimes. At each layer in the system, hardware capability semantics may be selected to support either high assurance design (for separation kernel and type-safe language runtimes) or hybrid models combining traditional virtual addressing and capability operation (for commodity kernels and applications).

We expect that the use of our notion of tagged, typed, nonforgeable, and ubiquitous capabilities can greatly enhance trustworthiness in hardware and software that we can develop in the relatively near future, aided by recent advances in hardware development. Perhaps surprisingly, we also expect to be able to provide considerable backward compatibility with legacy software, although we also anticipate that over time this capability mode of operation could eventually dominate in newly designed critical systems.

Systemic Considerations

The need for principled secure systems is longstanding, and decades of research have illustrated many approaches that might usefully contribute to building such systems. In light of these considerations, a reasonable question is "why now?". What has changed that would allow such an effort to succeed where so many previous efforts have not succeeded? Several factors have motivated our decision to begin this project:

- Dramatic changes in threat models, resulting from ubiquitous connectivity and pervasive uses of computer technology in many diverse and widely used applications such as wireless mobile devices, automobiles, and critical infrastructures.

- New opportunities for research into (and possible revisions of) hardware-software interfaces, brought about by programmable hardware (especially FPGA soft cores) and complete open-source software stacks.

- An increasing ability to expose the inherent multiprocessing of hardware through virtual machines and explicit software multiprogramming, and an increasing awareness of information flow for reasons of power and performance that can nicely align with the requirements of security.

- Emerging advances in programming languages, such as the ability to map language structures into protection parameters in order to more easily express and implement various policies.

- Reaching the tail end of a "compatibility at all costs" trend in CPU design, because of the possibilities inherent in heterogeneous distributed systems and possibilities for incremental adoption of new system concepts. While Wintel remains entrenched on

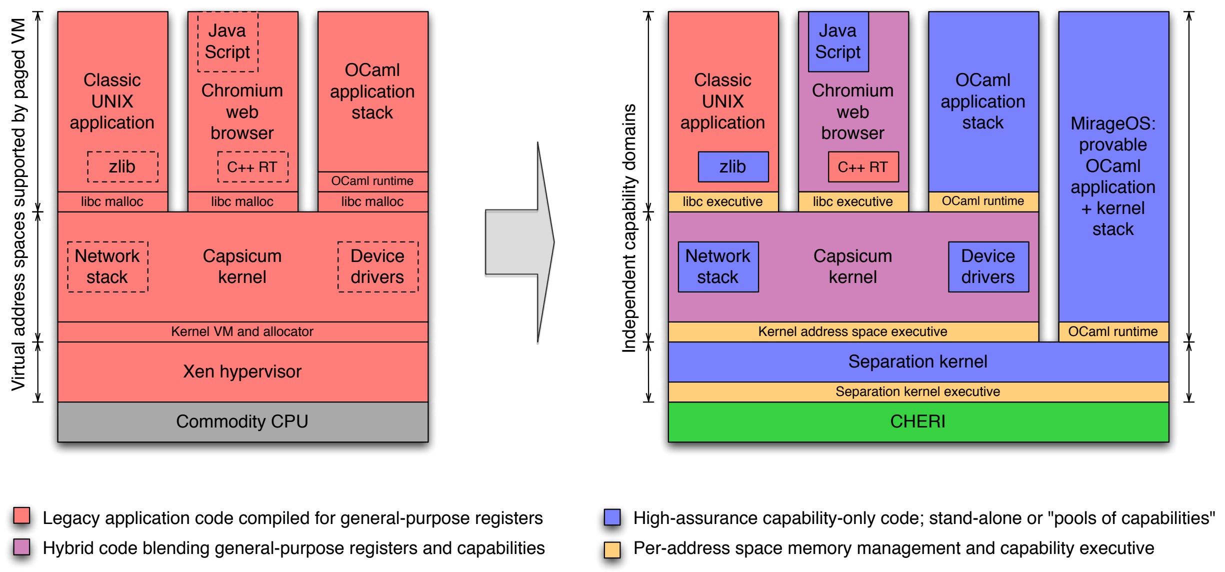
Virtual address spaces supported by paged VM
Classic UNIX application
zlib
libc malloc
Java Script
Chromium web browser
C++ RT
libc malloc
OCaml application stack
OCaml runtime
libc malloc
Network stack
Capsicum kernel
Device drivers
Kernel VM and allocator
Xen hypervisor
Commodity CPU
Independent capability domains
Classic UNIX application
zlib
libc executive
Java Script
Chromium web browser
C++ RT
libc executive
OCaml application stack
OCaml runtime
MirageOS: provable OCaml application + kernel stack
Network stack
Capsicum kernel
Device drivers
Kernel address space executive
OCaml runtime
Separation kernel
Separation kernel executive
CHERI
🟥 Legacy application code compiled for general-purpose registers
🟪 Hybrid code blending general-purpose registers and capabilities
🟦 High-assurance capability-only code; stand-alone or "pools of capabilities"
🟨 Per-address space memory management and capability executive

Figure 1: CTSRD proposes a *hybrid* view of the world, in which commodity page-oriented virtual memory system elements are combined with a capability system world view. Commodity applications such as Chromium will be able to exploit "pools of capabilities" — subcomponents that rely on stronger hardware protections — while allowing other portions of the applications to use lower-assurance models.

the desktop, mobile systems — such as phones and tablet PCs, as well as appliances and embedded devices — are significantly more diverse, running on a wide variety of hardware architectures (especially ARM and MIPS). Likewise, new diversity in operating systems has been seen, in which commercial products such as Apple's iOS and Google's Android extend open source systems such as FreeBSD and Linux. These new platforms abandon many traditional constraints, requiring significant changes to applications due to new security models, programming languages, hardware architectures, and user input modalities.

- Significant changes in the combination of hardware, software, and formal methods to enhance assurance (such as those noted above) now make possible the development of trustworthy system architectures that previously were too far ahead of their times.

The System Architecture

CTSRD draws on two distinct, and previously uncombined, designs for processor architecture.

- Page-oriented virtual memory systems, in which an executive (often the operating system kernel) configures the memory management unit (MMU) to create a *process* abstraction. In this model, the kernel is responsible for maintaining separation using this relatively coarse tool, and then providing system calls that allow spanning process isolation subject to access control. Systems such as this make only weak distinctions between code and data, and in the mapping from programming language to machine code discard most typing and security information.

- Capability system designs, often based on a single global address space, in which type information and protection constraints map from the programming language into instruction selection. Code at any given moment in execution exists in a protection domain consisting of a dynamic set of rights whose delegation is controlled by the flow of code. Such a design is generally recognized to offer greater assurance, as the *principle of least privilege* is applied at a fine granularity.

In CTSRD, our goal is to provide access to both models in a *hybrid design*, with the aim of allowing the gradual adoption of capability properties in hardware. As with current commodity systems, a series of address spaces is constructed by an executive, allowing coarse-grained memory control and the construction of isolated processes. However, within each address space, a hardware-supported capability security model will be constructed, managed by an in-process executive with full rights to the address space, which may then construct capabilities and limit the rights available to code that it invokes. The basic system architecture, named Capability Hardware Enhanced RISC Instructions (CHERI) is emerging as a new tagged, typed, capability-based system description that will be formally specified and formally analyzed as appropriate.

The notion of hybrid design is key to the adoption argument: CHERI systems will be able to execute today's commodity operating systems with few modifications. Use of capability features can then be selectively introduced in order to raise our confidence in the robustness and security of individual system components, which may fluidly interact with other unenhanced components. This notion of hybrid design first arose in Cambridge's Capsicum [24], which blends the POSIX Application Programming Language interface (API), as implemented in the FreeBSD operating system, with a capability design by allowing processes to execute in hybrid mode or in *capability mode*. Traditional POSIX code can run side by side with capability-mode processes, allowing sandboxes to be constructed, and using a capability model, rights delegated to sandboxes by applications that embody complex security policies — one such example from our USENIX Security Capsicum paper [24] is the Chromium web browser, which must map the distributed World Wide Web security model into local OS containment primitives.

CTSRD's software stack will employ hybrid design principles from the bottom up: a new capability-enhanced Separation Kernel and Hypervisor (SKAH) will implement an internal, hardware-supported capability model used to ensure its robustness. SKAH will then provide an execution substrate on which both commodity systems built on traditional RISC instruction models, such as Capsicum, can run side-by-side with a pure capability-oriented software stack, such as capability-adapted language runtimes. Further, Capsicum and its applications will be able to employ CHERI features in their own implementation.

To this end, the CHERI CPU design allows processor contexts to operate in one of two modes: pure capability mode, in which only accesses to the virtual address space authorized by delegated capabilities will be permitted, and hybrid mode, in which the use of capability instructions implies capability constraints, but traditional load and store instructions allow direct access to the virtual address space. For example, in this model Capsicum employs capability-oriented instructions in the implementation of risky data manipulations (such as network packet manipulation), while still relying on traditionally written and compiled code for the remainder of the kernel. Similarly, within the Chromium web browser, the JavaScript interpreter might be implemented in terms of capability-oriented instructions to offer greater robustness, while the majority of Chromium uses traditional instructions.

One particularly interesting property of our hardware design is that it is possible for capabilities to take on different semantics within different address spaces, with each address space's executive integrating memory management and capability generation. In the Capsicum kernel, for example, virtual addressing and capability use can be blended, with the compiler and kernel memory allocator using capabilities for certain object types, but not for others. In various userspace processes, a hybrid UNIX / C runtime might implement limited pools of capabilities for specially compiled components, but another process might use just-in-time (JIT) compilation techniques to map Java bytecode into CHERI instructions, offering improved performance and a significantly smaller and stronger Java TCB.

We are further considering the adaptation of Cambridge's MirageOS stack [11], an OCaml-based operating system, to CHERI. By modifying the OCaml runtime to use capability instructions, we would both harden and dramatically reduce the size of the MirageOS TCB. MirageOS could execute either under the CHERI-adapted Capsicum operating system as a user process, or directly over the separation kernel in order to provide a "pure" capability-oriented and high-assurance software stack.

This hybrid view offers a vision for a gradual transition to stronger protections, in which individual libraries, applications, and even whole operating systems, can incrementally adopt stronger hardware memory protections without sacrificing the existing software stack.

Critical to this effort is the recent growth in power and expressiveness of FGPA technology; FPGAs are able to implement *soft cores,* or dynamically programmed CPU cores constructed from arrays of logic units. This newly found flexibility permits us not just to simulate but also to implement new CPU designs at relatively low cost (even moderately high-end FPGA reference boards cost in the low thousands of dollars). For our starting point, we have selected the University of Cambridge's TIGER MIPS research platform, which includes 32-bit and 64-bit soft core designs, which will be extended to support a new suite of capability registers and instructions.

Another key aspect of this work is the use of open source licenses and methodology: the CHERI specification, a reference soft core implementation, and most layers in the software stack will be built on and distributed as open source software.

Layered Assurance

The CHERI system (and more generally the CTSRD architecture) is intended to support the pragmatic application of assurance techniques throughout design and implementation. Layered assurance has several unified thrusts:

- The overall system architecture is consistently and compatibly capability aware, from the hardware through the separation kernel to the programming environment (e.g., OCaml extended and its modified compiler). Each abstraction will hide whatever can be desirably hidden without loss of flexibility.

- The formal methods and selective formal analyses will take advantage of the specification of abstractions and representations of essential properties at these layers, along with their explicitly bound interrelationships among those abstractions (e.g., respectful of the state mappings and abstract implementations of the Robinson-Levitt paper [17]).

- We are considering formal methods techniques that can be mechanically applied to our hardware design, such as model checking and formal verification, which will give us greater confidence in the hardware semantics that we have selected.

Conclusions

Although the project described here began only in the fall of 2010, we believe that our approach is sound, timely, and much needed. Throughout, our key design principles support the goal of high assurance design grounded in hardware **and** a hybrid approach offering an incremental adoption path. This brief paper is of course a preliminary attempt to characterize our intentions, and clearly subject to refinement. We look forward to discussions at LAW 2010.

Appendix: Some Relevant Historical Predecessors

Throughout the 1970s and 1980s, it was envisioned that high-assurance systems would employ a capability-oriented design mapping program structure and security policy into hardware enforcement; for example, Lampson's BCC design exploited this linkage to approximate least privilege [9, 10]. Morris first described the potential for linking programming language structure and enforcement for operating systems [12]. Systems such as the CAP Computer at Cambridge attempted to realize this vision through embedding notions of capabilities in the memory management unit of the CPU [25].

However, with a transition from complex instruction set computers (CISC) to reduced instruction set computers (RISC) and a shift away from microcode toward operating system implementation of complex CPU functionality, the attention of security researchers turned to microkernels. Carnegie Mellon's Hydra, Accent, and Mach systems embodied this approach, in which microkernel message passing between separate tasks stood in for hardware-assisted security domain crossings at capability invocation. Successors have taken this approach further, with Shapiro's EROS capturing fine-grained compartmentalization on a capability-oriented microkernel model [22, 2] on commodity hardware. General-purpose systems also adopt elements of the capability design philosophy, such as Apple's Mac OS X using Mach interprocess communication (IPC) objects as capabilities [1], and Cambridge's Capsicum research project, which attempts to blend capability-oriented design with UNIX.

Our CHERI capability hardware design responds to all these design trends — and their problems. Reliance on traditional paged virtual memory for hard address space separation, as used in Mach, EROS, and UNIX, comes at significant cost: attempts to compartmentalize system software and applications sacrifice the programmability benefits of a language-based capability design (a point made convincingly by Fabry [4]), as well as introducing significant performance overhead to cross security domain boundaries. However, running these existing software designs is critical in order to improve the odds of technology transfer, and to allow us to incrementally apply ideas in CHERI to large-scale contemporary applications such as office suites. CHERI's hybrid approach allows a gradual transition from virtual address separation to capability-based separation within a single address space, restoring programmability and performance so as to facilitate fine-grained compartmentalization throughout the system and its applications.

We consider some of our own past system designs in greater detail, especially as they relate to CTSRD.

Multics [15, 3] The Multics system incorporated many new concepts in hardware, software, and programming. The Multics hardware provided independent virtual memory segments, paging, interprocess and intra-process separation, and cleanly separated address spaces. The Multics software provided symbolically named files that were dynamically linked for efficient execution, rings of protection providing layers of security and system integrity, hierarchical directories, and access-control lists. Input-output was also symbolically named and dynamically linked, with separation of policy and mechanism and separation of device independence and device dependence. A subsequent redevelopment of the inner rings enabled Multics to support multilevel security in the commercial product. Multics was implemented in a stark subset of PL/I that considerably diminished the likelihood of many common programming errors. In addition, the stack discipline inherently avoided buffer overflows.

PSOS [13, 14] SRI's Provably Secure Operating System hardware-software design was formally specified, with encapsulated modular abstraction, interlayer state mappings, and abstract programs relating each layer to those on which it depended. The hardware design provided tagged, typed, nonforgeable capabilities that were required for every operation. In addition to a few primitive types, application-specific types could be defined and their properties enforced with the hardware assistance provided by the capability access controls. The design allowed application layers to execute instructions efficiently, with capability-based addressing directly to the hardware.

Capsicum [24, 23] Capsicum is a lightweight OS capability and sandbox framework planned for inclusion in FreeBSD 9. Capsicum extends, rather than replaces, UNIX APIs, providing new kernel primitives (sandboxed capability mode and capabilities) and a userspace sandbox API. These tools support compartmentalization of monolithic UNIX applications into logical applications, an increasingly common goal supported poorly by discretionary and mandatory access control. This approach was demonstrated by adapting core FreeBSD utilities and Google's Chromium web browser to use Capsicum primitives, showing significant complexity and robustness benefits to Capsicum over other confinement techniques.

References

[1] Apple Inc. Mac OS X Snow Leopard. `http://www.apple.com/macosx/`, 2010.

[2] H. Chen and J. Shapiro. Using build-integrated static checking to preserve correctness invariants. In *Proceedings of the Eleventh ACM Conference on Computer and Communications Security (CCS)*, Washington, D.C., November 2004.

[3] R.C. Daley and P.G. Neumann. A general-purpose file system for secondary storage. In *AFIPS Conference Proceedings, Fall Joint Computer Conference*, pages 213–229. Spartan Books, November 1965.

[4] R.S. Fabry. The case for capability based computers (extended abstract). In *SOSP '73: Proceedings of the Fourth ACM Symposium on Operating System Principles*, page 120, New York, NY, USA, 1973. ACM.

[5] L. Gong. A secure identity-based capability system. In *Proceedings of the 1989 Symposium on Research in Security and Privacy*, pages 56–63, Oakland, California, May 1989. IEEE Computer Society.

[6] P.A. Karger. *Improving Security and Performance for Capability Systems.* PhD thesis, Computer Laboratory, University of Cambridge, Cambridge, England, October 1988. Technical Report No. 149.

[7] P.A. Karger and A.J. Herbert. An augmented capability architecture to support lattice security and traceability of access. In *Proceedings of the 1984 Symposium on Security and Privacy*, pages 95–100, Oakland, California, April 1984. IEEE Computer Society.

[8] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification of an operating-system kernel. *Communications of the ACM*, 53:107–115, June 2009.

[9] B.W. Lampson. Dynamic protection structures. In *AFIPS '69 (Fall): Proceedings of the November 18-20, 1969, Fall Joint Computer Conference*, pages 27–38, New York, NY, USA, 1969. ACM.

[10] B.W. Lampson. Protection. *SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

[11] A. Madhavapeddy, R. Mortier, R. Sohan, T. Gazagnaire, S. Hand, T. Deegan, D. McAuley, and J. Crowcroft. Turning down the LAMP: software specialisation for the cloud. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, HotCloud'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[12] J.H. Morris, Jr. Protection in programming languages. *Communications of the ACM*, 16(1):15–21, 1973.

[13] P.G. Neumann, R.S. Boyer, R.J. Feiertag, K.N. Levitt, and L. Robinson. A Provably Secure Operating System: The system, its applications, and proofs. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, May 1980. 2nd edition, Report CSL-116.

[14] P.G. Neumann and R.J. Feiertag. PSOS revisited. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003), Classic Papers section*, pages 208–216, Las Vegas, Nevada, December 2003. IEEE Computer Society. http://www.acsac.org/ and http://www.csl.sri.com/neumann/psos03.pdf.

[15] E.I. Organick. *The Multics System: An Examination of Its Structure.* MIT Press, Cambridge, Massachusetts, 1972.

[16] S.A. Rajunas, N. Hardy, A.C. Bomberger, W.S. Frantz, and C.R. Landau. Security in KeyKOS. In *Proceedings of the 1986 IEEE Sympsium on Security and Privacy*, April 1986.

[17] L. Robinson and K.N. Levitt. Proof techniques for hierarchically structured programs. *Communications of the ACM*, 20(4):271–283, April 1977.

[18] J.M. Rushby. The design and verification of secure systems. In *Proceedings of the Eighth ACM Symposium on Operating System Principles*, pages 12–21, Asilomar, California, December 1981. [ACM Operating Systems Review, 15(5)].

[19] J.M. Rushby. Proof of Separability–a verification technique for a class of security kernels. In *Proceedings of the Fifth International Symposium on Programming*, pages 352–367, Turin, Italy, April 1982. M. Dezani-Cianaglini and U. Montanari, eds., Springer-Verlag, Berlin, Lecture Notes in Computer Science, Vol. 137.

[20] J.M. Rushby. A separation kernel formal security policy in PVS. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, March 2004.

[21] J.M. Rushby and B. Randell. A distributed secure system (extended abstract). In *Proceedings of the 1983 IEEE Symposium on Security and Privacy*, pages 127–135, Oakland, California, April 1983. IEEE Computer Society.

[22] J.S. Shapiro and N. Hardy. EROS: A principle-driven operating system from the ground up. *IEEE Software*, 19(1):26–33, January/February 2002.

[23] R.N.M. Watson. New Approaches to Operating System Security Extensibility. Technical report, Ph.D. Thesis, University of Cambridge, Cambridge, UK, October 2010.

[24] R.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.

[25] M.V. Wilkes and R.M. Needham. *The Cambridge CAP Computer and Its Operating System*. Elsevier North Holland, New York, 1979.