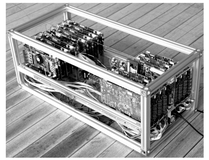
CTSRD

Dr Peter G. Neumann (SRI) and **Dr Robert N. M. Watson (Cambridge)**
Jonathan Anderson, Ross Anderson, David Chisnall, Nirav Dave, Brooks Davis, Rance DeLong, Khilan Gudka, Steven Hand, Asif Khan, Myron King, Ben Laurie,
Patrick Lincoln, Anil Madhavapeddy, Ilias Marinos, Andrew W. Moore, Simon W. Moore, Alan Mujumdar, Steven J. Murdoch, Robert Norton, Philip Paeps, Michael Roe,
John Rushby, Hassen Saidi, Muhammad Shahbaz, Stacey Son, Richard Uhler, Jonathan Woodruff, Bjoern A. Zeeb

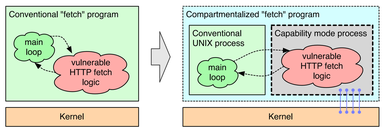


CHERI tablet and rack-mount BlueHive array, based on Terasic's DE4 FPGA board. CHERI CPUs support fine-grained compartmentalization, mitigating broad classes of known and unknown vulnerabilities.

CTSRD is developing a principled, formally-supported, robust, programmer-friendly, high-performance and incrementally adoptable hardware/software platform designed for efficient implementation of the principle of least privilege. Software security structures and design principles are reinforced by:

- **Capability Hardware Enhanced RISC Instructions (CHERI)**
- **Security Oriented Analysis of Application Programs (SOAAP)**
- **Temporally Enforced Security Logic Assertions (TESLA)**

CTSRD adopts a hybrid approach, able to run existing C-language operating systems and application software while supporting gradual adoption of its novel protection features for critical Trusted Computing Bases (TCBs) and high-risk components. CTSRD allows programmers to wipe the slate clean incrementally.



Capsicum and the application compartmentalization motivation

Programmers are turning to **application compartmentalization** to mitigate inevitable vulnerabilities: software is decomposed into sandboxed components, each with only the rights it requires to function. This approach employs the **principle of least privilege**: as granularity increases, rights delegated to individual sandboxes decrease. As vulnerabilities are exploited, only the rights of the affected component are leaked, forcing attackers to exploit many more vulnerabilities to accomplish the same goals.

The **Capsicum hybrid capability model** developed by Cambridge and Google blends contemporary OS design with capability system security, addressing semantic mismatches between OS features and application compartmentalization. Rights are delegated using **capabilities**, unforgeable tokens of authority. **Capability-mode** processes have access only to explicitly delegated rights as the use of OS global namespaces is denied.

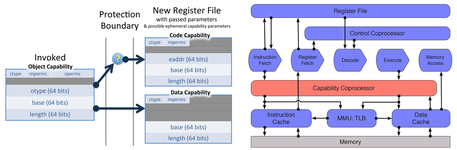
However, **compartmentalization scalability** – utilization of increasing numbers of sandboxes – is constrained by performance and programmability limitations of current hardware and software. Today's CPU instruction set architectures (ISAs) reflect a 1990s design consensus **conflating virtualization and protection**, limiting protection scalability. Compartmentalizing applications using IPC-linked processes also introduces **distributed systems programming problems** for local applications.

Current systems are exposed to greater threats, demanding dramatically increased use of compartmentalization, placing strain on protection and programming models designed for less risky workloads. CTSRD is developing clean-slate technologies to support large-scale deployment of compartmentalization for the first time.

Capability Hardware Enhanced RISC Instructions (CHERI)

CHERI provides a fine-grained protection model within address spaces, complementing existing virtual-memory based process models by providing efficient and more programmable support for application compartmentalization:

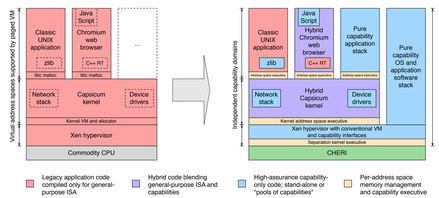
- Uses a reduced instruction set computer (RISC) approach, providing tools for compiler and operating system writers while minimizing hardware complexity.
- Targets low-level software TCBs: OS kernels, language runtimes and web browsers, as well as high-risk data processing such as video decoding.
- Allows simultaneous implementation of different security models, reflecting diverse OSs, programming languages, and application requirements.



A RISC approach to capability system security

CHERI extends MIPS with additional **capability registers**, which represent the current thread's security context. Application code utilizes **data capabilities** to provide low-level memory safety linked to program constructs, and **object capabilities** to implement efficient security domain switching. Thread context switches are security domain switches, offering a number of potential hardware implementations of object invocation, including hardware message passing facilities.

With CHERI, domain switching and shared memory between security domains are potentially orders of magnitude faster than conventional MMU-based designs – allowing orders of magnitude greater software compartmentalization to be deployed within our most sensitive and highest risk software components. Subdivision within kernel and application address spaces using capabilities allows software to play by single address space rules, avoiding distributed system programming problems.



An incrementally adoptable hybrid capability system model

CHERI evaluates capabilities prior to MMU virtual address translation, giving each UNIX process, and the kernel, its own capability system. Unmodified sandboxed programs fetch instructions, and load and store data implicitly via reserved capability registers. Capability-aware and legacy MIPS code can be combined, allowing unmodified programs to use compartmentalised libraries.

Thanks to its hybrid capability architecture, CHERI can be adopted one software component at a time. CHERI provides immediate security benefits for critical software, while offering a long-term capability system vision motivated by the principle of least privilege – with access to a complete software stack from day one.

Bluespec Extensible RISC Implementation (BERI)

BERI is a platform for research into the hardware-software interface:

- 64-bit MIPS ISA FPGA soft CPU core
- Off-the-shelf open-source software stack including FreeBSD, LLVM, and applications

Bluespec is a high-level hardware description language that facilitates hardware design-space exploration. BERI runs in a cycle-accurate simulation, and in Altera FPGA-based Terasic DE4 boards at 100 MHz and the Xilinx-based NetFPGA 10G boards.

CHERI1 and CHERI2 hardware prototypes

We have developed two prototypes of the 64-bit CHERI processor based on BERI, both able to boot FreeBSD and run a full suite of open-source applications as well as act as a testbed for fine-grained, hardware-supported compartmentalization:

- **CHERI1** employs a conventional approach to pipelined processor implementation in Bluespec, focused on a performant contemporary implementation. This has been the primary software prototyping and demonstration platform to date.
- **CHERI2** uses a stylised form of Bluespec intended to support formal verification techniques through a mapping from Bluespec into PVS. CHERI2 also implements initial support for multi-threading and multi-core.

CHERI ISA-level testing and verification

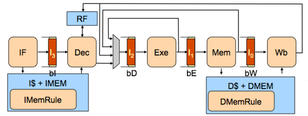
CHERI and CHERI2 implement both the 64-bit MIPS and CHERI ISAs. This requires significant testing and analyses:

- We have created a PVS model of the CHERI ISA to reason about security properties and check the expected results of tests
- We have developed an extensive MIPS and CHERI ISA test suite (thousands of tests).
- We have implemented a fuzzing suite to detect pipeline and exception-handling bugs.

```
cSetLength(s : CMachine, cd : Register,
  cb : Register, rt : DWord) :
    CMachine =
    IF registerAccessible(s, cd) AND
       registerAccessible(s, cb) AND
       s`registers(cb)`tag AND
       s`registers(cb)`unsealed AND
       0 <= rt AND
       rt <= s`registers(cb)`length THEN
       s WITH [
          `registers := s`registers WITH [
             (cd) := s`registers(cb) WITH [
                `length := rt
             ]
          ]
       ] ELSE
          exception(s)
    ENDIF
```

Formal verification of the CHERI microprocessor

In collaboration with Bluespec, Inc. we have provided a translation from BSV intermediate representations to Seri, a high-level Haskell-based constraint language. This provides a path to reason about BSV designs in Sal, PVS, or Yices. Combined with our CHERI2 prototype, we will soon begin formal verification of the CHERI implementation, including ISA-level properties and pipeline correctness.



CHERI extensions to Clang/LLVM

We have added compiler back-end support for CHERI features to the LLVM compiler infrastructure and begun experimenting with C extensions via the Clang front end. This includes hardware enforcement of language rules and informal patterns and new extensions. For example, the hardware can enforce immutability and bounds checking on pointers and safely sharing memory between mutually-distrusting compartments. We next plan to extend support to higher-level languages, including enforcing object encapsulation in Objective-C, even with C code doing arbitrary pointer arithmetic.

```
int aFunction(const __capability int *x)
{
  // This will abort at run time:
  *(__capability int) x = 42;
  return *x;
}
```

CheriBSD: Adapting UNIX for hardware capabilities

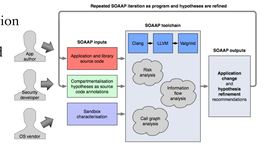
CheriBSD is an adaptation of the widely used FreeBSD operating system to support CHERI's protection features. CHERI sandboxes replace capability-mode UNIX processes in Capsicum, supporting orders of magnitude more compartmentalization granularity. We have made modest changes to support capability-aware software:

- The kernel maintains per-thread capability register state.
- The kernel enforces that system calls are made only by unsandboxed user code.
- A software capability invocation exception handler has been implemented.

CheriBSD supports tablet, desktop, server, and embedded demonstration scenarios.

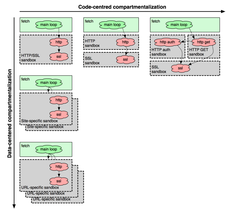
Security Oriented Analysis of Application Programs (SOAAP)

Experience with Capsicum shows that adapting programs for compartmentalization is difficult, leading to problems with correctness, performance, complexity, and critically, security. SOAAP is a set of semi-automated techniques to assist programmers with compartmentalization.



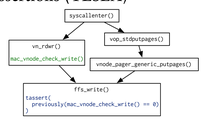
Compartmentalization hypotheses are explored through source-code annotations describing sandboxing strategy (e.g., sandbox creation, rights delegation, and RPC forwarding). Security goals and properties (such as information flow constraints and past vulnerabilities) can also be labeled in program source code.

SOAAP uses static and dynamic analysis to engage developers in interactive dialogue, identifying potential correctness bugs (e.g., data inconsistencies), and security breaches (e.g., information leaks). SOAAP builds on Clang, LLVM, Valgrind, and DTrace.

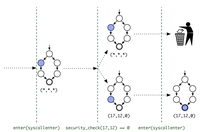


Temporally Enhanced Security Logic Assertions (TESLA)

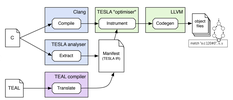
TESLA allows programmers to describe temporal properties of security-critical software. For instance, in the inset example, a programmer specifies that calls to the ffs_write() function must be proceeded by a successful mac_vnode_check_write() call, but this check is missing in one code path through the VM paging system.



Programmers can write TESLA assertions in C, allowing tight integration with the code they describe – including C type checking – or in the domain-specific Temporal Event Assertion Language (TEAL). Assertions are converted into finite-state automata that are driven by software events observed by instrumentation. This allows us to validate security properties at runtime.



TESLA is implemented as a Clang analysis tool, which interprets assertions, and a suite of LLVM-based tools that instrument code and implement automata.



Members of the CTSRD team and its external oversight group at our May 2011 review meeting in Cambridge, UK

Joe Stoy (Bluespec), Jonathan Woodruff (Cambridge), Ben Laurie (Google), Ross Anderson (Cambridge), Virgil Gligor (CMU), Philip Paeps (Cambridge), Li Gong (Mozilla), Peter Neumann (SRI)

Simon Cooper, Michael Roe (Cambridge), Robert Watson (Cambridge), Howie Shrobe (DARPA), Steven Murdoch (Cambridge), Sam Weber (NSF), Jonathan Anderson (Cambridge), Simon Moore (Cambridge)

Anil Madhavapeddy (Cambridge), Dan Adams (DARPA), Rance DeLong (LynuxWorks), Jeremy Epstein (SRI), Hassen Saidi (SRI)

Many Cambridge members of the CTSRD team meeting for an August 2012 project photo in Cambridge, UK

Jonathan Woodruff, Richard Clayton, Mchael Roe, Ross Anderson, David Chisnall, Robert Watson

Khilan Gudka, Robert Norton, Simon Moore

Approved for public release. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237. The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.