Background

Problem
• From 2006-2019, ~70% of patched security vulnerabilities in Microsoft software were C/C++-language memory-safety vulnerabilities. Similar rates exist in other stacks (e.g., Android).
• These vulnerabilities directly enable confidential data extraction, unauthorized data modification, and arbitrary code execution.
• There are no viable solutions today due to concerns over adoption cost, software compatibility, and performance.

Approach
CHERI (Capability Hardware Enhanced RISC Instructions) extends existing computer Instruction-Set Architectures (ISAs) with support for a new hardware data type architectural capabilities. Capabilities are used by the C/C++-language compiler/toolchain, operating system, and applications to enable:
• High-performance fine-grained memory protection. Exploited memory-safety vulnerabilities, such as buffer overflows, and exploit techniques, such as Return-Oriented Programming (ROP) throw exceptions rather than leaking or corrupting data, or enabling code execution, with modest performance overhead (typically <5%).
• Scalable software compartmentalization. Whereas today, a system may only support a dozen concurrent sandboxes to constrain untrusted code, CHERI systems scale 1-2 orders of magnitude better, enabling greater compartmentalization granularity. For example, rather than sandboxing each website, CHERI can sandbox at the granularity of every image on a page.

CHERI transition activities
First developed in the DARPA I2O CRASH program (2010), CHERI addressed ~30% of exploited vulnerabilities. DARPA MTO SITH has extended CHERI to mitigate 70% of patched vulnerabilities, as well as evaluated and optimized security, performance, and compatibility. There are three significant in-progress technology transition efforts:
• Since 2015, in collaboration with Arm, SRI and Cambridge have been working to transition CHERI to Arm’s 64-bit ARMv8-A ISA. Arm’s first experimental CHERI-based System-on-Chip (SoC), Morello, a 7nm multi-core design, will ship in late 2021 following funding from DARPA I2O, ATO, and MTO, and, announced in 2019, additional support of $236M (£187M) by Arm, Microsoft, Google, and Amazon, and the UK government.
• Since 2017, Arm has adapted the ARMv8-M to support the CHERI protection model. (See separate poster and demonstration.)
• Since 2017, SRI and Cambridge have implemented a full open-source RISC-V-based reference architecture, multiple example CPU cores validated on FPGA, and complete open-source software stack for demonstration, evaluation, and transition to custom ASICs.

Technical Approach

Research and development approach
CHERI provides architectural mitigation for C/C++ TCB vulnerabilities:
• Tagged memory, new hardware capability data type protect the integrity, provenance validity, and target data of each pointer
• The CHERI model hybridizes cleanly with contemporary RISC ISAs, CPUs, MMU-based OSes, and C/C++-language software
• Incremental software deployment of CHERI protection features: recompile existing code with few or no source-level changes

CHERI was developed through iterative hardware-software-semantics co-design, prototyping, evaluation, and refinement over 10 years:
• CHERI abstract protection model; concrete ISA instantiations in 64-bit MIPS, 32/64-bit RISC-V, 64-bit ARMv8-A, 32-bit ARMv8-M.

CHERI architectural capabilities

CHERI extends pointer values with:
• Tags protect capabilities in registers and memory
• Dereferencing an untagged capability throws an exception
• In-memory overwrite automatically clears capability tag
• Bounds limit range of address space accessible via pointer
• Larger allocations have stronger alignment requirements
• Out-of-bounds pointer support for C-language compatibility
• Permissions limit operations – e.g., load, store, fetch
• Sealing: immutable, non-dereferenceable capabilities – used for non-monotonic transitions

Example microarchitecture: CHERI-Piccolo Microcontroller

Results and Impact

CHERI software protection and compatibility
• Capabilities are refined by the kernel, run-time linker, compiler-generated code, heap and stack allocators, …
• Protection mechanisms:
  • Referential memory safety
  • Spatial memory safety + privilege minimization
  • Temporal memory safety
• Applied automatically at two levels:
  • Language-level pointers point explicitly at stack and heap allocations, global variables, …
  • Sub-language pointers implement control flow, linkage, etc.
• Sub-language protection mitigates bugs in the language runtime and generated code, as well as attacks that cannot be mitigated by higher-level memory safety
  • (e.g., union type confusion)

CHERI-RISC-V early evaluation across multiple FPGA-based cores
• Prototyped for three CPUs
• Median cycles overhead (MiBench):
  • 3-stage pipeline (P1): 20% (pre-optimization)
  • 5-stage pipeline (P2): 9% (pre-optimization)
• Superscalar core (P3): <1% (pre-optimization)
• CWE-based security evaluation (by vulnerability class):
  • 100% of buffer-related errors
  • 75% of resource management errors
• 35% of numeric errors
• Vendor security analysis (estimates based on vendor information)
>70% of Microsoft 2019 patched security vulnerabilities mitigated
Comparable portion of Android/Chrome vulnerabilities mitigated
100% of Apple iOS GPO 2019 zero-day vulnerabilities mitigated
• Early compartmentalization evaluation shows a 90%+ reduction in IPC overhead to sandbox applications using multiple UNIX processes

If adopted, CHERI will eliminate a majority of known and potential future security vulnerabilities while maintaining industrially viable performance and software ecosystem compatibility.

https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement A – Approved for Public Release, Distribution Unlimited. All Illustrations are authors’ own.