1. The Capability Hardware Enhanced RISC Instructions (CHERI) processor and software implements the key concepts of modern memory protection while simultaneously maintaining high performance and high security. This is a significant advancement over previous approaches, which often sacrificed performance for security. The CHERI architecture is motivated by the need to mitigate security vulnerabilities or map distributed system software to the CHERI processor and software.

2. CHERI addresses these problems through a least privilege architecture. The CHERI architecture enforces per-address space memory management and compartmentalization by targeting low-level software tobits. It provides the capability for developers to control which components of the system can access data in system memory. This is achieved through a combination of hardware and software mechanisms. The CHERI architecture is a set of hardware and software features that work together to ensure that different parts of the system can access only the data that they need to perform their tasks.

3. Within an address space, a threat’s security context is entirely isolated from other contexts by security context switches. CHERI is a multi-threaded processor supporting low-latency message passing of general purpose and capability registers. This translates the efficient protected subsystem invocation, orders of magnitude faster than can be supported in MMU-based hardware designs.

4. CHERI targets low-level software TBoPs. OS kernels, language run-time, and web browsers, as high-level data processing such as video decoding, CHERI’s hybrid capability architecture enables the capability for software to operate in a secure, controlled environment. CHERI combines untrusted RISC code by instructing loads and stores via general purpose registers through a reserved capability register. CHERI’s isolation of capabilities and the MMU places capability environment “above” the virtual address space.

5. CHERI’s 64-bit MIPS-derived CPU primitive is written in the BlueSpec hardware description language (HDL) infusing hardware prototyping and design space exploration with commodity CPUs today. The BlueSpec HDL is a hardware description language that allows designers to describe and simulate hardware systems. CHERI’s capability registers implement a large number of simultaneous and frequent switching among isolation domains to the highest efficiency, offering scalable data and software isolation. The size of CHERI’s memory management unit (MMU) is significantly reduced compared to traditional MMUs, allowing capabilities, code, and data to co-exist in system memory.

6. The CHERI architecture allows developers to flexibly configure and program their systems without sacrificing performance. This is achieved through a combination of hardware and software mechanisms. CHERI is a set of hardware and software features that work together to ensure that different parts of the system can access only the data that they need to perform their tasks.

7. Temporarily Enhanced Security Logic Assertions (TESLA) employs low-level, data-flow checking, pre-execution assertions into software at compile-time. The TESLA assertions are inserted into the source code at compile-time, and the program is then compiled with these assertions included. This allows the program to be checked for security vulnerabilities and performance costs before it is run.

8. TESLA’s simple assertion language adds temporal quantifiers previously unavailable to the C assertion syntax, allowing assertions to refer to past and future events specified in a precise and programmer-selected order (pre-order or global). Assertions can be specified both at run-time and compile-time, and can be expressed exactly as automata using the TESLA Assertion Language (TELAV). TELAV validates FreeBSD’s TCP implementation by checking that applications conform to the the TCP protocol specification. The technique can also be used to automate cryptographic protocol checking (e.g., PISC or SIE) and can be applied to other components. TESLA sandbox unmodified RISC code to other components. CHERI sandbox unmodified RISC code.

9. We plan to add new assertion types checking sampled data distributions over time and real-time properties. The TESLA assertions validate cryptographic and network protocol properties such as sequence numbers, timestamps, and message integrity.

10. TESLA is in development using clang/LLEM-based C instrumentation framework and the libtrees run-time library. Assertions are converted into C and the TESLA clang plug-in instruments function prologues, epilogues, and other language-visible events. Libtrees provides a mechanism for generating TESLA assertions that can be inserted into the source code at compile-time.