Hardware-Assisted Safety for seL4

Prashanth Mundkur, PhD
Gabriela Ciocarlie, PhD
Peter G. Neumann, PhD
Improving Computer System Assurance

- Principled design improvements to system components
  - Hardware
    - Tagged memory and capabilities
    - IOMMUs / TrustZone
  - Software
    - Memory-safe type-safe languages
    - Formal verification
System-Safe Design Space

- seL4
- RISC-V/Rust/Compartmentalization
- Linux C/Rust/Go
- CHERI
CHERI Design Choices

Goal: Mitigate issues in C/C++ languages via hardware/software co-design

Implementation:

- Achieve spatial memory safety via treating pointers as capabilities to memory
  - Hardware-supported bound and permission enforcement on memory access
- Temporal memory safety (work in progress)
- Segmenting memory address space and program execution into compartments with differing trust relationships
CHERI Design Choices

Cost/benefit tradeoff

+ Minimal perturbation to C/C++ language (benefit)
+ Leverage vast existing C/C++ application ecosystem (benefit)
- More complex hardware (cost)
  -> tagged memory and larger registers
  -> impact on cache footprint and usable memory bandwidth
- More complex program design
  -> use of compiler language annotations, application compartmentalization
- Kernel and C/C++ software toolchain support (cost, benefit)
A Different Design Choice on seL4
Current Software Landscape for seL4

- Has more choices for memory-safe system languages (Go, Rust, Ocaml,...)

- Fewer constraints from backward compatibility with existing C/C++ ecosystem

- Unlikely to have fully-verified user-space anytime soon (e.g., Firefox web browser)
A Different Design Choice on seL4

Goal: Mitigate application security issues in memory-safe languages via hardware/software co-design

- Leverage formal verification of C-language kernel
- Leverage strong inter-process compartmentalization
- **BUT** support large unverified user-space applications in memory-safe languages
A Different Design Choice on seL4

**Goal:** Mitigate application security issues in memory-safe languages via hardware/software co-design

- Leverage formal verification of C-language kernel
- Leverage strong inter-process compartmentalization
- **BUT** support large unverified user-space applications in memory-safe languages

**Key Need:** intra-process CHERI-like compartmentalization
A Different Design Choice on seL4

Cost/benefit tradeoff

+ Minimal perturbation to Rust/Go/.. language (benefit)
+ Simpler processor design than CHERI (benefit)
- Smaller Rust/Go/... ecosystem – especially on seL4 (cost,benefit)
- OS and language toolchain support (cost)
Motivating Intra-Application Compartmentalization

- User applications (web browsers, email clients, document editors, etc.) are increasing in complexity
  
- Require handling images, fonts, HTML, PDF, Javascript, etc.
  => Code of different quality and attack surfaces co-exist in the same address space
  
- Use of memory-safe languages does not mitigate attacks exploiting complex application logic
Example: Complexity of Firefox (DARPA TC: TRACE)
Intra-Application Compartmentalization: Processor Support

- Constrain code-execution to within-compartment control-flow transfers
- Constrain memory accesses to within-compartment data
- Support well-defined cross-compartment calls and returns
Intra-Application Compartmentalization: Language Support

- Leverage stronger module systems in memory-safe languages
  - Natural boundaries for language-level compartments

Potential Complications:
- Extend garbage collection for within-compartment collections
- Language-level exception handling
Conclusion

- Hardware-assisted support for seL4
  - Hardware support for within-address-space compartmentalization
  - Memory safety
  - Avoiding the need for formal verifications for large complex applications
SRI is hiring! Seeking researchers in computer science and related fields at all levels.