Verifying enclave systems with Serval

Luke Nelson
w/ James Bornholt, Ronghui Gu, Andrew Baumann, Emina Torlak, Xi Wang
University of Washington, Columbia University, Microsoft Research
Enclave monitors are hard to get right

- Correctness of enclave monitor code is critical for security
- Many different kinds of bugs are security vulnerabilities:
  - **Low-level bugs**: e.g., buffer overflow or division-by-zero
  - **Logic bugs**: implementation does something unintended
  - **Design bugs**: intended design of the system is not secure
- Each can be exploited to compromise the entire system
Eliminating bugs with formal verification

- **Goal**: prove absence of low-level, logic, and design bugs

- **Approach**: Use automated verification techniques
  - Low proof burden: symbolic evaluation / SMT solvers
  - Bounded loops in code, likely true of monitors
  - Limitations: no concurrency or side channels
Challenges

- Difficulty of building verifiers
  - Need detailed RISC-V machine model
  - Need to reason at ISA level
- Difficulty of scaling to practical systems
  - Symbolic evaluation
  - SMT solving
Serval: A framework for verifying low-level systems

- Built on top of Rosette
- Lift ISA interpreter into verifier
  - Easier to write and test
  - Supports LLVM IR and RISC-V
- Use symbolic profiling to identify verification bottlenecks
- Use symbolic optimizations to scale verification
Main Results

- Applied to CertiKOS (PLDI’16) and Komodo (SOSP’17), previously manually verified using Coq and Dafny
- Found and fixed 15 Linux BPF JIT bugs, all now upstreamed.
  - [https://git.kernel.org/linus/1e692f09e091](https://git.kernel.org/linus/1e692f09e091)
  - [https://git.kernel.org/linus/46dd3d7d287b](https://git.kernel.org/linus/46dd3d7d287b)
  - [https://git.kernel.org/linus/68a8357ec15b](https://git.kernel.org/linus/68a8357ec15b)
  - [https://git.kernel.org/linus/6fa632e719ee](https://git.kernel.org/linus/6fa632e719ee)
- Work in progress: identified implementation and design bugs in Keystone
Outline

● Verification stack
● Workflow
● Demo
Verification stack

Inputs

- System implementation
- System specification

Serval

- Automated verifiers
- Symbolic optimizations

Rosette

- Symbolic evaluation
- Symbolic profiling
- Symbolic reflection

Solver

- Satisfiability checking
- Counterexample generation
[1/3] Proving absence of low-level bugs

C program

Clang

LLVM IR + UBsan checks

Serval LLVM verifier

✓
[2/3] Proving functional correctness

- $t_1$ to $t_2$: Monitor call specification
- $S_1$ to $S_2$: Monitor call RISC-V instructions from GCC
- $t_1$ to $S_1$: Abstraction function
Proving noninterference

- Example: bogus monitor call that returns enclave secrets
- Integrity — OS should not be able to modify enclave-visible state
- Confidentiality — Behavior of OS is independent of enclave secrets
Demo: Komodo

- A verified software enclave monitor
- We have ported to RISC-V and verified using Serval

Demonstration:
- Low-level buffer overflow vulnerability
Demo
Conclusion

• Automated verification is effective at eliminating bugs in low-level systems
• If you are building enclave systems, talk to us!
• Paper to appear at SOSP’19
• Code will be released shortly

https://serval.unsat.systems/