System-Level Security Verification of RISC-V Based SoCs

Dr. Nicole Fern
Senior Hardware Security Engineer
RISC-V Summit 2019, San Jose, CA
Outline

• Why is system-level security important for RISC-V?
• Examples of system-level security vulnerabilities
• System-level security verification tools and methodologies
• Key Takeaways
SoCs are being built around RISC-V processors

https://www.sifive.com/chip-designer


System-level Security Requirements

1. No vulnerabilities in **hardware implementation** of RISC-V processor or other peripherals/IP in SoC

2. No vulnerabilities introduced during **SoC integration**
   - Improper management of debug and test interfaces
   - Accidentally ground privilege bit in a peripheral bus interface

3. Correct **software** configuration and usage of hardware features
   - Mismanagement of assets during a secure boot
   - Memory protection unit (MPU) misconfiguration
   - Programming error in on-chip bus access control table
Example Chip-level Vulnerabilities

Improper isolation of components connected via in-vehicle bus allows hackers to remotely disable breaks

https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

FPGA-based root of trust bitstream can be modified by attacker

https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20190513-secureboot

BLEEDINGBIT
Malicious Bluetooth Low Energy packets insecurely processed by Texas Instruments wireless SoCs

https://go.armis.com/bleedingbit

Android Alert: Users Urged To Patch Critical Flaw In Qualcomm Snapdragon Chips, Millions At Risk

Outline

• Why is system-level security important for RISC-V?
• Examples of system-level security vulnerabilities
• System-level security verification tools and methodologies
• Key Takeaways
• **Competition Goal:** Find security bugs inserted in RISC-V based SoC
  - Alpha Phase: 2 months
  - Beta Phase: 33 hours @ DAC Conference
  - Bugs “donated” by Intel and Qualcomm

• **Team:** Tortuga Logic and Texas A&M University won 1st place in both phases

Bugs found and reported include block-level hardware vulnerabilities, integration errors, and software security bugs
Example Block-Level Hardware Bugs

- JTAG is password protected
  - Password only protects against reading (can still write)
  - Lock status bit not set to known value on reset
- AES (counter mode)
  - IV and counter value fixed
- Platform Level Interrupt Controller (PLIC)
  - Multiple drivers issue routing SPI and UART interrupts
Example System Integration Security Issues

• AES Register Interface
  • Key is hard-coded and can be read through register interface
  • Internal state accessible through register interface

• JTAG Password Storage
  • Password stored in location unprivileged SW can write and read

• Boot code stored in dedicated ROM
  • Can be overwritten by unprivileged SW

• Register Locks
  • Unlocked on reset, which can be issued by software debug command

• Direct Memory Access (DMA)
  • Assets (ex. system timers) could be accessed by unprivileged software via DMA
• AXI-4 crossbar firewall settings programmed during boot allow unprivileged access to almost all peripherals
• Trap handler code can be overwritten by unprivileged software which can then trigger the trap and execute in machine-mode
• Beta phase software APIs for SHA and AES (privileged code) don’t validate input pointers allowing writes to arbitrary memory locations
Outline

• Why is system-level security important for RISC-V?
• Examples of system-level security vulnerabilities
• System-level security verification tools and methodologies
• Key Takeaways
System-level Security Verification Essential

Preventing integration, configuration and usage errors requires novel security verification methodologies and tools.

Security Verification has Gaps!

- Threat Modeling
- Security Specification
- Block and Subsystem Design
- SoC Integration
- Low-level Software Testing
- Post-Si Security Testing

GAP

- Architecture Design Review
- Formal Verification
- Directed Tests
- Penetration Testing

Copyright Tortuga Logic Inc., 2019
Radix™ Pre-silicon Security Verification Solution

Standard Verification Flow

**Simulation**
- Cadence Xcelium™
- Synopsys VCS™
- Mentor QuestaSim™

**Emulation**
- Cadence Palladium™
- Synopsys ZeBu™
- Mentor Veloce™
Example: Detect AES Key Leakage

- AES peripheral accessible through register interface
  - Plaintext, ciphertext, control and status bits, etc.
- Software at any privilege level can access AES
  - Security Bug: all key bits can be directly read by software through memory-mapped address
• AES key should never flow out of AES block except as ciphertext
• Security Rule: key_in =/> all_outputs ignoring { ct }

Rule fails when key is read through bus interface

Red shading indicates information from source (ex. AES Key) is flowing to signal

For any signal, explore drivers and loads

Browse design hierarchy
SW Issue: AES Fuse Keys Leaking to Processor, DRAM
Key Takeaways

• RISC-V processors are being integrated in commercial SoCs!

• RISC-V processors must be verified in the context of the larger design (hardware and software) to provide system-level security

• Tortuga Logic’s Radix ensures security maintained throughout the process of integrating a RISC-V core into a SoC/FPGA system
Security begins with hardware.

nicole@tortugallogic.com