Hardware Implementation of Information Flow Signatures Derived via Program Analysis

Paul Dabrowski, William Healey, ***Karthik Pattabiraman***, Shelley Chen, Zbigniew Kalbarczyk, Ravishankar Iyer

Center for Reliable and High-Performance Computing
University of Illinois, Urbana-Champaign

Motivation

- Shrinking process technology → complexity
  - Insufficient validation → hardware design bugs
  - Intentional hardware bugs by malicious designer
  - Multi-core introduces many more entry points

- Comprehensive technique to protect from a broad class of memory/code vulnerabilities
  - Both known and unknown attacks
  - Protection even if attacker is inside system
  - Low area and performance overheads

This Paper: IFS Technique

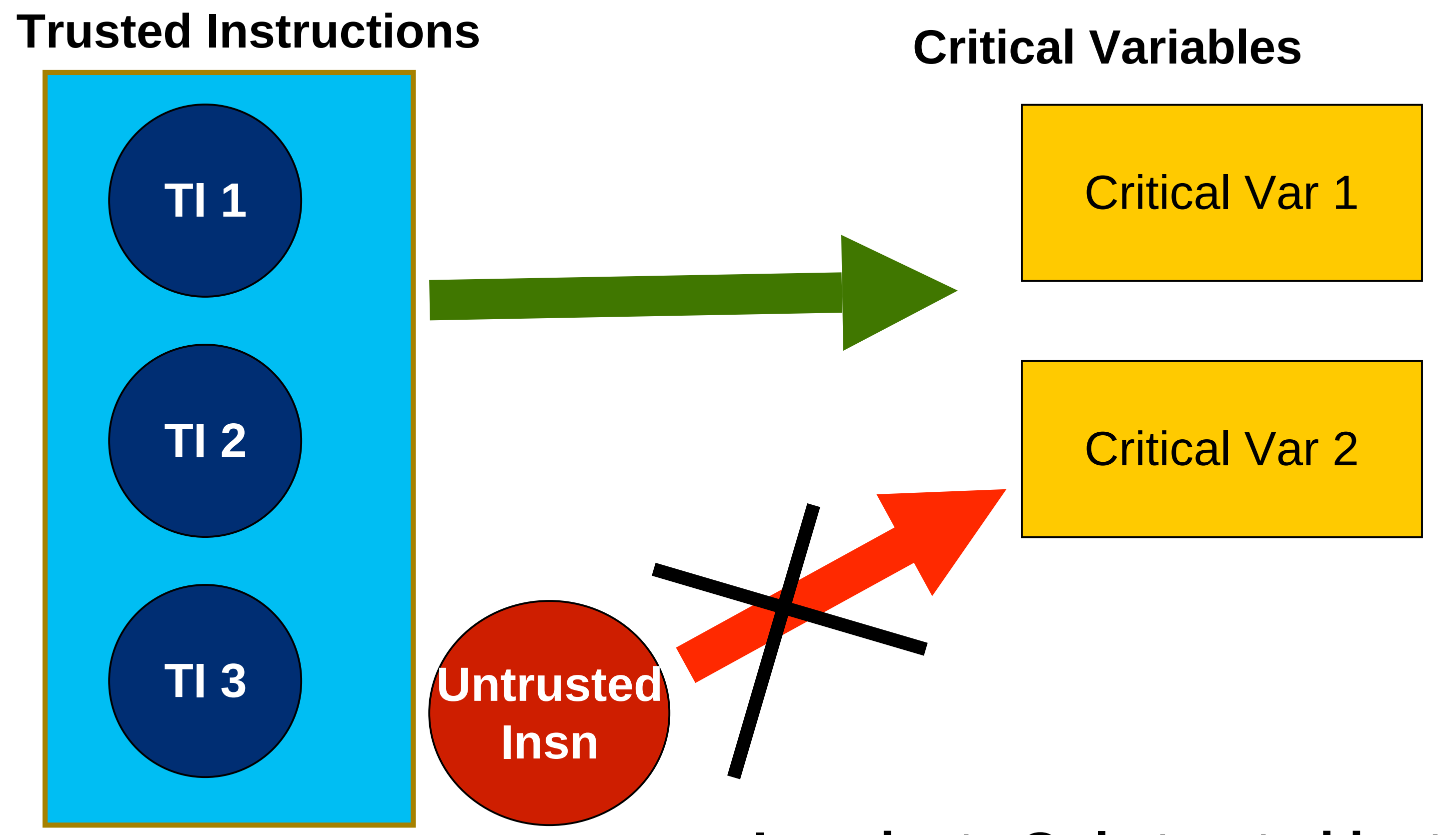
- Focuses on protecting the target of attack or the critical data based on insn. dependencies
- Protect from wide range of memory and code corruption attacks (existing and future)
  - No assumptions on possible entry points
  - No assumptions on source of attack
  - No reliance on trustedness of operating system

Information Flow Signatures (IFS)

- **Programmer** identifies critical data in application based on knowledge of application semantics
- **Static Analysis:** Extract inter-procedural backward slice for critical variables
  - Identify instructions in backward slice (trusted)
  - Identify data objects for trusted instructions
- **Runtime Enforcement** (Using both H/W + S/W)
  - Ensure that runtime behavior conforms to slice

Level 1 Checking

Checked for all instructions in program (using hardware)

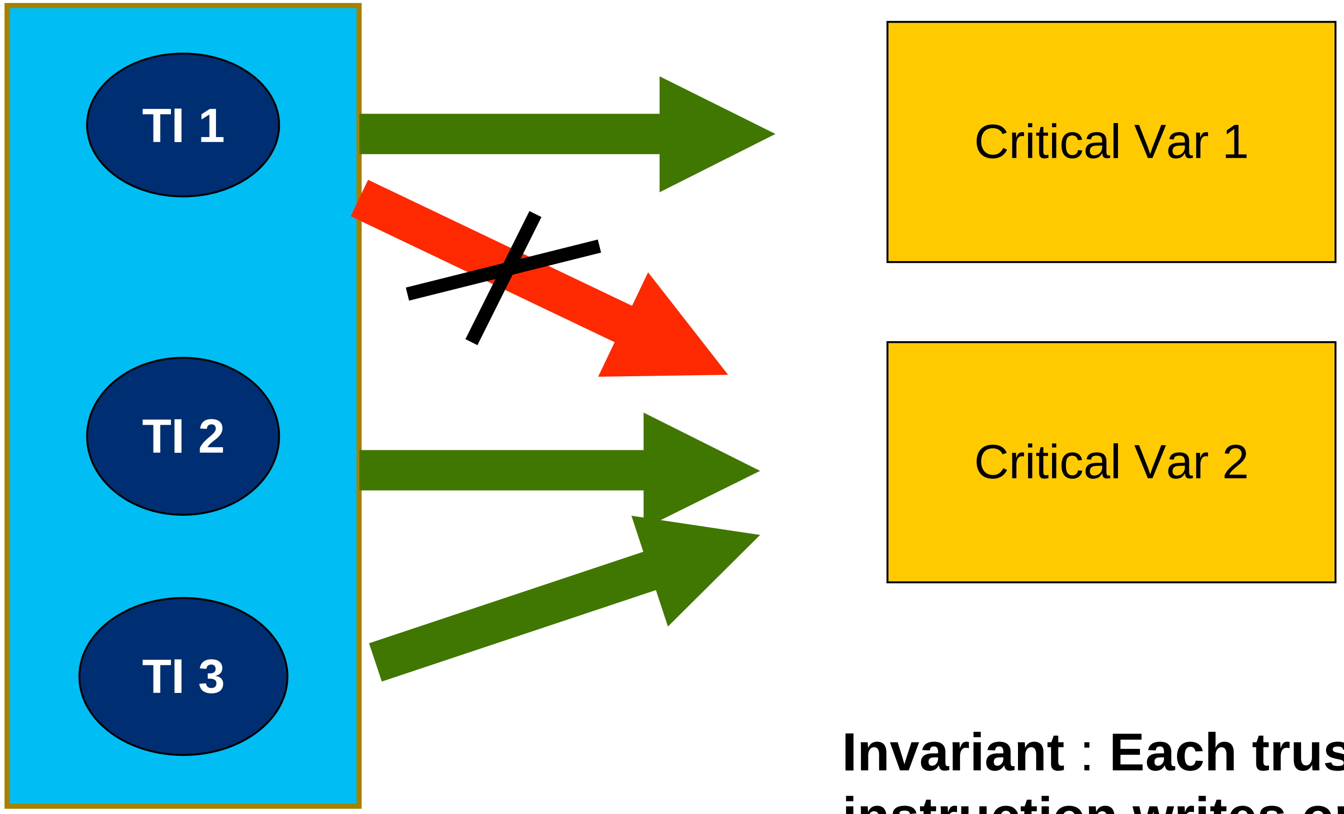


Trusted Instructions: In backward slice of critical variables

Invariant : **Only trusted instructions can influence values of critical variables**

Level 2 Checking

Checked only for trusted instructions in the program (using software)



Trusted Instructions: In backward slice of critical variables

Invariant : **Each trusted instruction writes only to its allowed target objects (according to static analysis)**

IFS Level 1 Check Implementation (Hardware Enforcement)

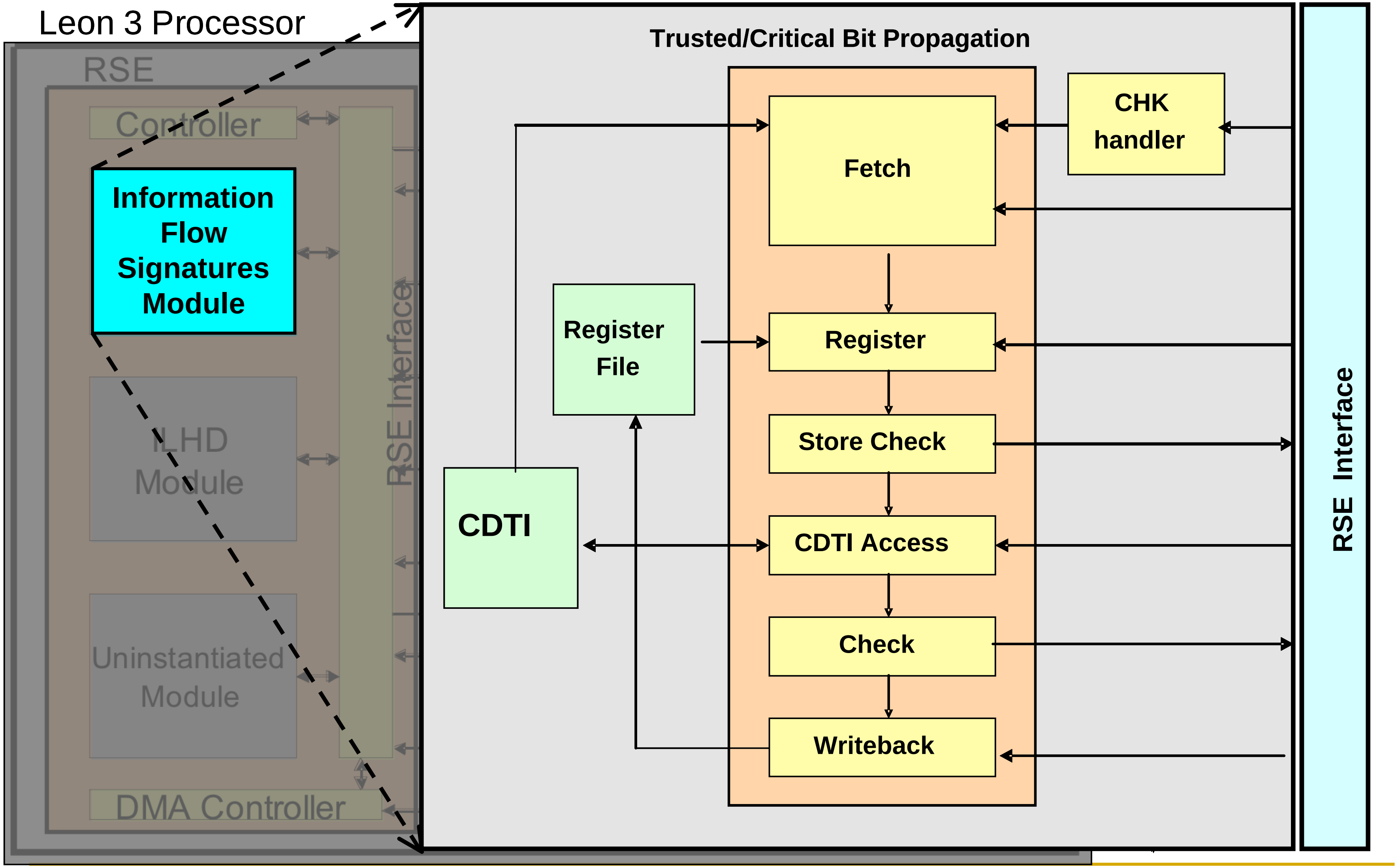
Every instruction and data item has a trusted bit associated with it

Trusted(I.dest) ← Trusted(I.pc) && OperandsTrusted

OperandsTrusted ← Trusted(I.op1) && …. && Trusted(I.opN)

I.dest I.pc, Operands	Critical Data	Non-critical Data
(Untrusted, Untrusted)	Raise Alarm	Allow
(Untrusted, Trusted)	Raise Alarm	Allow
(Trusted, Untrusted)	Raise Alarm	Raise Alarm
(Trusted, Trusted)	Pass to Level 2	Pass to Level 2

Hardware Implementation



Results

Benchmark	Power	TSP
# Instructions	10388	5144
# Trusted Instructions	726 (7.0%)	118 (2.3%)
# Trusted/Critical Memory Locations	30	1
Performance Overhead	1%	69%

Hardware Area overhead of 4.2%

Conclusion and Future Work

- IFS Technique to protect critical data
  - Combination of hardware and software support
  - Hardware overhead < 5%
  - Performance overhead highly dependent on app

- Future Work
  - Level 2 checks in hardware
  - Extend CDTI to work with virtual addressing
  - Extend to superscalar processors and multi-core

Related Work

- Focus on defending against specific attacks
  - Stack smashing/Heap buffer overflows
  - System call based attacks

- Cannot protect critical data once attacker gains access to system (**Insider Attacks**)

- Have prohibitive space and time overheads or impose restrictions on source language