SimSpect defense specification examples

Guide to defense implementations in SimSpect formal language and model notes

Stanford | ENGINEERING
Electrical Engineering

Contents

For each defense we outline:
- Defense overview and paper notes
- SimSpect specification of each defense variant
- Implementation notes and mechanisms to exercise in test generation
- Known bugs (if applicable)

Defenses:

Stanford | ENGINEERING
Electrical Engineering

STT — MICRO '19

- Defines explicit vs. implicit covert channels
- Stalls **explicit** transmitters until youngest load whose return value influences data becomes nonspeculative **(YROT)**
- Blocks **implicit branch prediction** from operating as a function of tainted data
- Blocks **implicit branch resolution** from transmitting until predicate is nonspeculative
- Untaints the output of an access instruction when all older control flow instructions resolve (ctrl speculation) or when access instruction cannot be squashed (does this in 1 cycle)

Stanford | ENGINEERING
Electrical Engineering

STT SimSpect specification

- **Protection set:** all memory. *Propagation: strict dependency + YROT untaint calculus — a. Any speculative access instruction is tainted. b. Other output is tainted if input is tainted. Untainted once YROT nonspeculative. YROT is youngest YROT of args if not AI, or else AI addr.*
- **Leakage contract:** explicit transmitter set user-defined, implicit transmitters also handled by aforementioned prediction and resolution taint blocks
- **Execution contract:** Visibility point,

Stanford | ENGINEERING
Electrical Engineering

STT mechanisms to exercise

- **Protection set:** remove instructions, memory/registers, edges
- **Leakage contract**
- **Execution contract**

Stanford | ENGINEERING
Electrical Engineering

SpecShield — PACT ’19

- **STL** – delays AIs for the duration of speculation, being defined as reaching the head of the ROB
- **ERP** – delays AIs for the duration of speculation, by waiting until all older branches are resolved and nonspeculative and all older loads and stores have no faults (nothing is speculative)
- **ERP+** – tracks taint and allows low leakage instructions to see AIs of speculative things as long as they don’t pass to high leakage things

- Handles speculation primitives: meltdown, branch misprediction, memory alias misprediction, control flow hijacking
- ONLY defends memory secrets, not register secrets, through AI provision

Stanford | ENGINEERING
Electrical Engineering

SpecShield-STL SimSpect specification

- **Protection set:** All memory. *Propagation: strict dependency — a. Any speculative access instruction is protected. b. Other output is protected if input is protected (conservative because this should not be exercised because cut off at AI)*
- **Leakage contract:** Transmitter type irrelevant, should protect for all
- **Execution contract:** Instructions must reach the head of the ROB to become non-speculative

Stanford | ENGINEERING
Electrical Engineering

SpecShield-ERP SimSpect specification

- **Protection set:** All memory. *Propagation: strict dependency — a. Any speculative access instruction is protected. b. Other output is protected if input is protected (conservative because this should not be exercised because cut off at AI)*
- **Leakage contract:** Transmitter type irrelevant, should protect for all
- **Execution contract:** Instructions are non-speculative at head of ROB, but also before head of ROB if ∄ older unresolved branches, and ∄ unresolved or faulted older loads and stores

Stanford | ENGINEERING
Electrical Engineering

SpecShield-ERP+ SimSpect specification

- **Protection set:** All memory. *Propagation: strict dependency — a. Any speculative access instruction is protected. b. Other output is protected if input is protected*
- **Leakage contract:** High-likelihood transmitters
- **Execution contract:** Instructions are non-speculative at head of ROB, but also before head of ROB if ∄ older unresolved branches, and ∄ unresolved or faulted older loads and stores

Stanford | ENGINEERING
Electrical Engineering

NDA — MICRO ’19

- Two types of attacks
  - Control steering, victim control flow leaks its own information
  - Chosen code, attacker uses their own code to violate permissions speculatively
- An instruction is **safe** if not after a branch with unresolved target and direction or a store with an unresolved address
- When a microop resolves, mark things safe until the next outstanding speculation primitive
  - Delay broadcast of outputs until safe
- Strict vs permissive propagation version, which only treats loads as AIs (only protects memory)

NDA-S SimSpect specification

- **Protection set:** all memory and registers. *Propagation: strict dependency — a. Any speculative access instruction is protected. b. Other output is protected if input is protected*
- **Leakage contract:** Transmitter type irrelevant, should protect for all
- **Execution contract:** Instructions are non-speculative at head of ROB, but also before head of ROB if ∄ older branches with unresolved target or address, and ∄ stores with unresolved address

Stanford | ENGINEERING
Electrical Engineering

NDA-P SimSpect specification

- **Protection set:** all memory. *Propagation: strict dependency — a. Any speculative access instruction is protected. b. Other output is protected if input is protected*
- **Leakage contract:** Transmitter type irrelevant, should protect for all
- **Execution contract:** Instructions are non-speculative at head of ROB, but also before head of ROB if ∄ older branches with unresolved target or address, and ∄ stores with unresolved address

Stanford ENGINEERING
Electrical Engineering

SDO — ISCA ’20

- Insight: Because of STT implicit branch stopping, the predictor is safe. There should be no tainted data affecting a predictor
  - Squashes are revealable because they happen only when predicate is no longer tainted
  - Only be concerned about explicit transmitters
- Translate into a set of transmitter functions that cover path variability, and then predict path based on nonspeculative data (squashes are fine)
- Predict based on PC
- Loads are the complicated aspect, load function hides timing but not address leakage, must still block these types of leaks

Stanford | ENGINEERING
Electrical Engineering

SDO SimSpect specification

- **Protection set:** all memory. *Propagation: STT - strict dependency + YROT untaint calculus — a. Any speculative access instruction is tainted. b. Other output is tainted if input is tainted. Untainted once YROT nonspeculative. YROT is youngest YROT of args if not AI, or else AI addr.*
- **Leakage contract:** implicit transmitters handled by aforementioned prediction and resolution taint blocks as in STT, explicit transmitters refactored such that µpaths don't vary based on data
- **Execution contract:** Visibility point

Stanford | ENGINEERING
Electrical Engineering

SPT — MICRO ’21

- STT modified with declassification pass back based on implicit instruction functions for leakage propagation
- SNI – does not protect against things that haven’t already leaked
- DOES NOT discriminate between partial vs full leakage of an instruction
- Delayed execution protection policy

Stanford | ENGINEERING
Electrical Engineering

SPT SimSpect specification

- **Protection set:** all memory. *Propagation: STT. strict dependency + YROT untaint calculus — a. Any speculative access instruction is tainted. b. Other output is tainted if input is tainted. Untainted once YROT nonspeculative. YROT is youngest YROT of args if not AI, or else AI addr.*
- **Leakage contract:** explicit transmitter set user-defined, implicit transmitters also handled by aforementioned prediction and resolution taint blocks
- **Execution contract:** Visibility point

Stanford | ENGINEERING
Electrical Engineering

Recon — Micro ’23

Stanford | ENGINEERING
Electrical Engineering

Doppelganger Loads — ISCA ’23

Delay on Miss — smth ’year

Stanford | ENGINEERING
Electrical Engineering

Non-applicable defenses

Stanford | ENGINEERING
Electrical Engineering