SimSpect defense specification reference

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

Simple alloy model

- **Protection set, Leakage contract, and Execution contract**
  - Relaxations remove instructions, memory/registers, connecting edges from the model
  - Have generic speculation primitives, transmitters
  - Base of GLIFT-style information flow tracking to conservatively measure protection set propagation through the model

Stanford | ENGINEERING
Electrical Engineering

STT

Speculative Taint Tracking — 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. (like DIFT)*
- **Leakage contract:** explicit transmitter set user-defined, implicit transmitters also handled by aforementioned prediction and resolution taint blocks
- **Execution contract:** Architect-defined visibility point (at head of ROB, or oldest unresolved branch)

Stanford ENGINEERING
Electrical Engineering

STT

Mechanisms to exercise

- **Leakage contract**: must complicate transmitter tracking — implicit transmitters are blocked at both prediction and resolution, have two elements for those components and enable probing of relevant state for both
- **Protection set tracking**: first example of taint tracking, propagate direct dependencies GLIFT-style through the program
  - Basis for the model data-dependency propagation
  - Relax the data-dependency edges by removing them
    - This is sufficient because we do not check for overprotection, so no point in increasing the number of edges

Stanford | ENGINEERING
Electrical Engineering

STT
Known bugs

- **Store transmitters**
  - STT does not consider stores to be transmitters. This is in line with their paper and security definitions but not in line with real life or gem5

Stanford | ENGINEERING
Electrical Engineering

SpecShield

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

SpecShield

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

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

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

SpecShield

Mechanisms to exercise

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

Stanford | ENGINEERING
Electrical Engineering

NDA

Non-speculative Data Access — 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

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

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

NDA

Mechanisms to exercise

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

Stanford ENGINEERING
Electrical Engineering

DoM

Delay-on-Miss — ISCA ’19

- Blocks speculative loads from transmitting by allowing them to go to L1 cache but no further, so they don’t exhibit data-dependent timing or prime the cache
- Doesn’t discriminate between non-speculatively and speculatively acquired secrets
- Used value prediction to speed up secure speculation
- Has both eager and normal speculation modes

Stanford | ENGINEERING
Electrical Engineering

DoM
SimSpect specification

- **Protection set:** All registers and memory
- **Leakage contract:** Protects against load-address transmission only
- **Execution contract:** Protects instructions that fall under *shadows:*
  - E-Shadows – instructions that can generate exceptions
  - C-Shadows – control instructions with an unknown address or condition
  - D-Shadows – load dependencies due to stores with unknown addresses because of STL forwarding
  - M-Shadows – under strict memory models, load reordering could lead to squashes

Stanford | ENGINEERING
Electrical Engineering

DoM

Mechanisms to exercise

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

Stanford | ENGINEERING
Electrical Engineering

SDO

Speculative Data-Oblivious Execution — 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:** Architect-defined visibility point (at head of ROB, or oldest unresolved branch)

Stanford | ENGINEERING
Electrical Engineering

SDO

Mechanisms to exercise

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

SPT
Speculative Privacy Tracking — 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 tainting, with additional untainting when an operand is transmitted by a non-speculative transmit or branch, followed by propagation:*
  - *Forward output untainting: if all operands untainted, output untainted*
  - *Backward input untainting: if mov or invertible arith, a singular tainted input is untainted if everything else is*
  - *STL load address transmission untainting*
- **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

SPT

Mechanisms to exercise

- If we don’t just want a conservative non-taint logic, need to differentiate between instructions at a finer granularity such that we can properly cover the untainting

SPT
Known bugs

- **Subregister writing**
  - X86 artifact that you can write subregisters, unprotects half-written secrets but then the unaligned load can leak information
  - Gem5 issues 2 separate calls and SPT unprotects both loads becase it has byte granularity protection but assumes the memory access hits the whole thing
- **Rename stage**
  - Default to mark registers as unprotected, then protect when renamed

Stanford | ENGINEERING
Electrical Engineering

ReCon

RevealConceal — Micro ’23

- Tracking non-speculative leakage is a good security definition, but carries a lot of overhead in tracking complexity (SPT)
- Insight: the majority of these non-speculative leakage events are pointer dereferences (load address transmissions)
- STT has loss of MLP in prohibition of load-load dependencies
- Only track pointer dereference s.t. we can limit overhead

Stanford ENGINEERING
Electrical Engineering

ReCon

ReCon-STT SimSpect specification

- **Protection set:** all memory. *Propagation: STT propagation – data leaked non-speculatively by a load-load pair address transmission*
- **Leakage contract:** explicit transmitter set user-defined, implicit transmitters also handled by aforementioned STT prediction and resolution taint blocks
- **Execution contract:** Shadows, as in DoM

Stanford | ENGINEERING
Electrical Engineering

ReCon

ReCon-NDA SimSpect specification

- **Protection set:** all memory. *Propagation: STT propagation – data leaked non-speculatively by a load-load pair address transmission*
- **Leakage contract:** all transmitters
- **Execution contract:** Shadows, as in DoM

Stanford | ENGINEERING
Electrical Engineering

ReCon

Mechanisms to exercise

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

Stanford | ENGINEERING
Electrical Engineering

ReCon
Known bugs

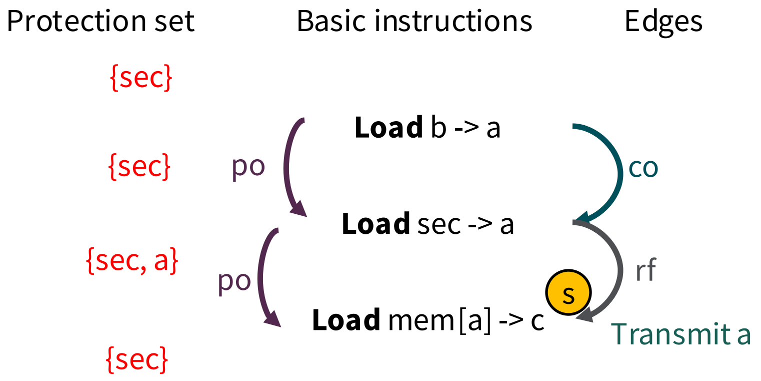
- **STT load order bug**
  - Stalls transmitters until oldest load that the sensitive op depends on has become nonspeculative (should be youngest)
- **Secondary STT bug**
  - Untaints register when the youngest load it depends on reads unprotected memory, but if there are multiple you must check all speculative loads it depends on
- **Memory unprotection protection set tracking bug**
  - Recon unprotects secret memory when an interleaved store makes the memory secret but its prior value leaks later

Stanford | ENGINEERING
Electrical Engineering

ReCon

STT load order bug

Recon stalls transmitters until oldest load that the sensitive operand depends on, should be the youngest.



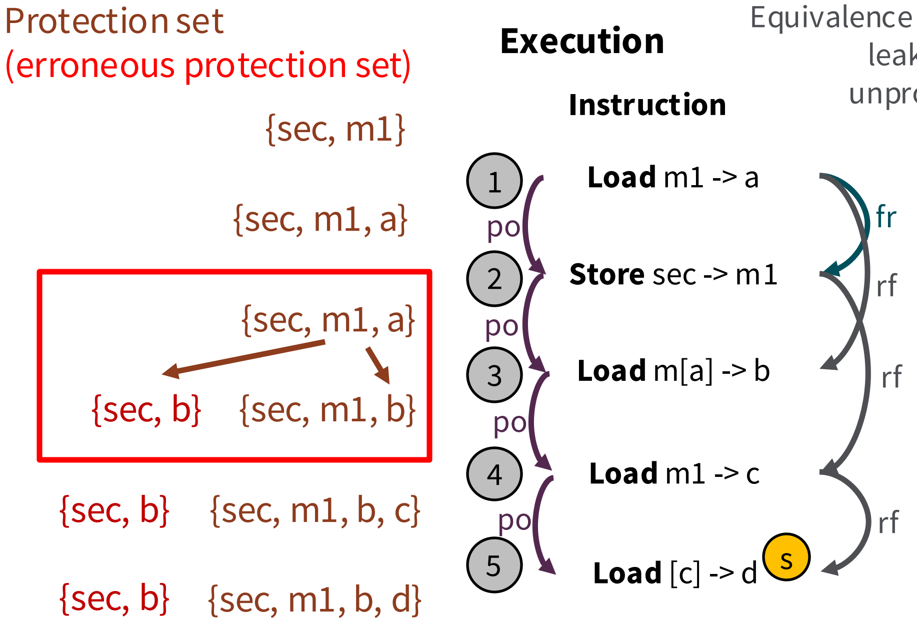
A protected item (a) leaks speculatively

Stanford | ENGINEERING
Electrical Engineering

ReCon

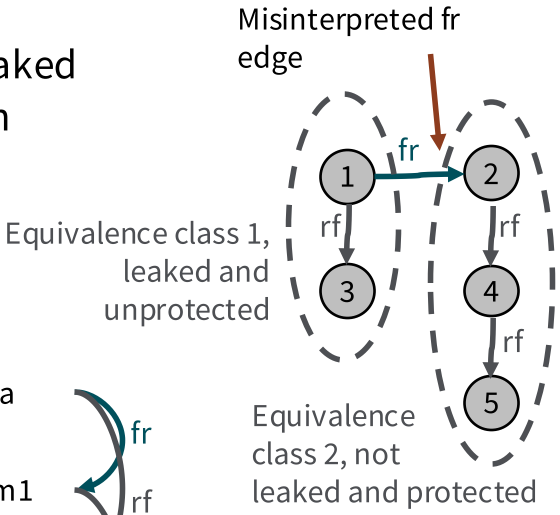
Known bug in memory unprotection

Recon unprotects memory when it is leaked nonspeculatively despite there being an interleaved secret store.



Transmit a
Recon thinks m1 leaked, unprotects m1 and a, splits protection sets

Transmit c
Sec leaked


Misinterpreted fr edge
Equivalence class 1, leaked and unprotected
1
fr
2
rf
rf
3
4
rf
5
Equivalence class 2, not leaked and protected

Hypothesis: Consider the equivalence classes of arch. state tracked by a defense. Poor protection set tracking occurs when non-rf edges are misinterpreted to collapse equivalence classes

Equivalence set to the output of instruction i at its execution time = what state could be unprotected by transmission at i = **i.(^(~rf+rf) & ^(~po))**

Stanford ENGINEERING
Electrical Engineering

ReCon

Secondary STT bug

IN PROGRESS

bad execution:

- prot m1
- spec (true)
- load ra from m1
- spec (false)
- load ra from m2
- load rb from m\[ra] *leak ra*

Stanford | ENGINEERING
Electrical Engineering

Doppelganger Loads

Doppelganger Loads — ISCA ’23

- SDO helps with the MLP problem but doesn’t neatly conceal load address, just load timing behavior
- Insight: predict load addresses with *doppelganger loads* that predict load address and load information from memory without predicating on speculative data
- Threat-model transparent!

Stanford | ENGINEERING
Electrical Engineering

Doppelganger Loads

DL-NDA 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

Doppelganger Loads

DL-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. (like DIFT)*
- **Leakage contract:** explicit transmitter set user-defined, implicit transmitters also handled by aforementioned prediction and resolution taint blocks
- **Execution contract:** Architect-defined visibility point (at head of ROB, or oldest unresolved branch)

Stanford | ENGINEERING
Electrical Engineering

Doppelganger Loads

DL-DoM SimSpect specification

- **Protection set:** All registers and memory
- **Leakage contract:** Protects against load-address transmission only
- **Execution contract:** Protects instructions that fall under *shadows:*
  - E-Shadows – instructions that can generate exceptions
  - C-Shadows – control instructions with an unknown address or condition
  - D-Shadows – load dependencies due to stores with unknown addresses because of STL forwarding
  - M-Shadows – under strict memory models, load reordering could lead to squashes

Stanford | ENGINEERING
Electrical Engineering

Doppelganger Loads

Mechanisms to exercise

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

Protean

Mechanisms to exercise

- **Under construction**

Stanford | ENGINEERING
Electrical Engineering

Non-applicable defenses

Todo

Stanford | ENGINEERING
Electrical Engineering