Speculative Taint Tracking (STT): A Comprehensive Protection for Transiently Accessed Secrets

Jiyong Yu, Mengjia Yan, Artem Khyzha,* Adam Morrison,* Josep Torrellas, Christopher W. Fletcher

University of Illinois at Urbana-Champaign *Tel Aviv University
Scope: Protecting Transient Secrets over any Covert Channel

Start speculative execution

Transient Instruction accesses secret “Access instruction”

Covert channel transmits secret

e.g., Spectre Variant 1:

```plaintext
if (addr < N) {
    secret = A[addr];  // access secret
    d = transmit(secret);  // covert channel
}  // start speculation
```
Adversary

- Powerful attacker
- Observes **HW resource contention** at FF/cycle granularity

- I.e., monitors **every** uarch covert channel `transmit(secret)`
  - From within same thread
  - Adjacent SMT context
  - Another core
  - Etc.
Main Insight of STT

It is safe to execute and forward the results of transient access instructions up until those results reach a covert channel.

Java Code:

```java
if (addr < N) {
    secret = A[addr];
    secret' = secret - 1;
    d = transmit(secret');
} 
```

- If `addr < N`:
  - `secret = A[addr]` is safe to execute.
  - `secret' = secret - 1` is safe to execute.
  - `d = transmit(secret')` is delayed/unreachable.

- If squash:
  - Unreachable

- If no squash:
  - Execute after delay

Issue more work early \(\rightarrow\)
Lower performance overhead
Security definition

Non-interference with respect to transiently read data

= 

Arbitrary speculative execution can only leak retired register file state
(not arbitrary program memory)

Blocks The Universal Read Gadget == many Spectre variants (1, 3, 4, ..), MDS attacks, Meltdown, etc.
STT, Informally

**Performance:** Walk through (not around) rock == Forward data aggressively

**Security:** Do not fall off rock == Leak zero bits over uarch side/covert channels
Rest of talk

1.) What can form a covert channel?

2.) How to protect secrets from covert channels

```c
if (addr < N) {
    secret = A[addr];
    d = transmit(secret);
}
```
1.) What can form a covert channel?

→ New abstraction for covert channels on spec. machines

Find new (classes of) covert channels in the process
Covert Channels are *Explicit* or *Implicit*

**Explicit channel:**
*Secret* passed to instruction w/ input-dependent resource contention

Examples: loads, floating point arithmetic

**Implicit channel:**
*Secret* influences the sequence of instructions executed/indirectly influences how those instructions execute

Examples: branches, memory dependence speculation
Implicit Channels can leak at **Prediction** and **Resolution** time

**Prediction time:**
When predictor state becomes a function of `secret`

**Resolution time:**
When predictor is not a function of `secret`

**Example:**
```
if (secret) { y++ }
z = *(non-secret)
```

- `secret` != prediction
- → squash
- → load executes twice!
Implicit Channels may feature *Explicit* or *Implicit* branches

**Explicit Branch:** control flow instruction (branch, jump, etc.).

**Implicit Branch:** HW branch, “which direction” is observable

**Example:** store-to-load forwarding

```c
*secret = foo;
bar = *(non-secret);
```

Whether load goes to cache depends on `secret`!

**Re-write load as:**

```c
if (secret==non-secret) {
    forward from LSQ
}
else cache_read(non-secret)
```

The implicit branch
Putting it all together

Covert channel

Explicit channel

Implicit channel

Explicit branch

Implicit branch

Prediction-based

Resolution-based

New!
2.) How to protect secrets from covert channels

→ Novel HW dynamic taint tracking scheme w/ “Fast Untaint”
Microarchitecture specifies...

1.) Access instructions ... access secrets
2.) Transmit instructions ... form explicit channels

E.g., access instructions = transmit instructions = loads

3.) Implicit channel branch predicates

E.g., store \(\rightarrow\) load forwarding “if load addr == addr of older in-flight store”

4.) Visibility Point (VP): point, after which, speculation is ‘unsafe’

E.g., oldest unresolved branch
Taint propagation in hardware

- Hardware **taints** instruction output registers of
  1.) Access instructions before Visibility Point (VP)
  2.) Instructions with tainted inputs
- Taint not propagated to memory on stores
- Tainted values are protected

```
if (addr < N) {
    secret = A[addr]; // access instruction
    secret’ = secret - 1;
    d = transmit(secret’);
}
```

“Visibility Point” (VP)

Taint shown in red
New “UnTaint” propagation in hardware

- Hardware\textit{untaints} instruction output registers of
  1.) Access instructions reached VP
  2.) Instructions with all inputs untainted
- Protection disabled when\textit{inst}\text{r}\text{o}\text{p}\text{e}nds become untainted

```c
if (addr < N) {
  secret = A[addr];
  secret -= 1;
  d = transmit(secret);
}
```

Non-trivial!

Taint is dynamic over instruction lifetime
Who is aware of taint

Only processor core (e.g., rename table) tracks taint

Processor memory system does not track taint

Taint is transparent to SW/Users
Using taint to block covert channels

**Block Explicit Channels:**
Delay execution of transmit instructions until operands *untainted*

**Block Prediction-time Implicit Channels:**
Delay predictor updates for explicit/implicit branches until *untainted*

**Block Resolution-time Implicit Channels:**
Delay explicit/implicit branch resolution until predicate (operands) *untainted*
What speculative work can we safely do?

- Safe to execute all instructions with **untainted** operands
- Safe to execute non-transmit instructions with **tainted** operands
- Safe to predict on implicit/explicit branches with **tainted** predicates
  
  Note: predictors have high accuracy.

```c
for (i = 0; i < secret; i++) {
    ... }
```

Safe to predict multiple iterations before first one resolves!
Architecture for “Fast Untaint”

Instructions stored in program order in Reorder Buffer (ROB)

**Observation:**
Each instruction with *tainted* arguments, has a “Youngest Root of Taint” (YRoT) in ROB

**Fast Untaint (high level):**
1.) Track YRoT per instruction
2.) Broadcast changes to VP
3.) $\text{Untaint} = \text{VP} > \text{YRoT}$
   (ROB head = position 0)
if (addr < N)
secret = A[addr]
secret -= 1
transmit(secret)

Branch resolves

if (addr < N)
secret = A[addr]
secret -= 1
transmit(secret)
Evaluation

Average Execution Time for SPEC and PARSEC (Normalized)

- Conventional insecure baseline: 1.45x
- DelayAccess: 1.11x
- STT: 2.92x
- DelayAccess: 1.16x

VP = oldest unresolved branch
VP = ROB head
Summary

STT blocks leakage of transient secrets over any uarch covert channel
Speculation can only leak retired register values

- No software changes
- No partitioning
- No flushing
- User does not indicate what is sensitive
- No memory system changes
Backup slides
Strawman Defense

*Delay executing access instructions until they reach the Visibility Point*

- Start speculative execution
- Instruction accesses secret “Access instruction”
- Covert channel transmits secret
- Delay until branch resolves
- Set access instruction = all loads

```c
if (addr < N) {
    secret = A[addr];
    d = transmit(secret);
    Delay
    Delayed/Unreachable
}
```

If squash:
unreachable
If no squash:
Execute after delay
Why Focus on Protecting Transient Access Instructions?

Covers a broad class of attacks related to memory isolation/safety.

Examples:
- Many Spectre variants (1, 3, 4, ..)
- MDS attacks
- Meltdown
- The Universal Read Gadget

Secrets accessed through transient loads

Others as well: e.g., privileged register read, FP state

As we will see, also facilitates making the scheme lightweight & SW-transparent.
Taint and Untaint

- Hardware **taints** instruction output registers of
  - Access instructions before VP
  - Instructions with tainted inputs
- Hardware **untaints** instruction output registers of
  - Access instructions reached VP
  - Instructions with all inputs untainted
- Taint not propagated to memory on stores

```c
if (addr < N) {
    secret = A[addr];
    secret -= 1;
    d = transmit(secret);
}
```

--- Branch resolves---

```c
if (addr < N)
    secret = A[addr]
    secret -= 1
    transmit(secret)
```
When to untaint

```c
if (addr < N) {
    secret = A[addr];
    secret -= 1;
    d = transmit(secret);
}
```

Earlier issue for known-safe instructions $\rightarrow$ Better performance

Can safely issue `transmit(secret')` as soon as branch resolves

*(Do not need to wait for `transmit(secret')` to reach head of ROB)*

ILLOISI