SpeechMiner: A Framework for Investigating and Measuring Speculative Execution Vulnerabilities

Yuan Xiao, *Yinqian Zhang*, *Radu Teodorescu*

The Ohio State University

SPEculative Execution side Channel Hardware (SPEECH) Vulnerabilities

- Leverage transient execution on modern x86 processors to leak secret data whose access is forbidden

Speculatively executes instructions

Execution leaves trace in cache

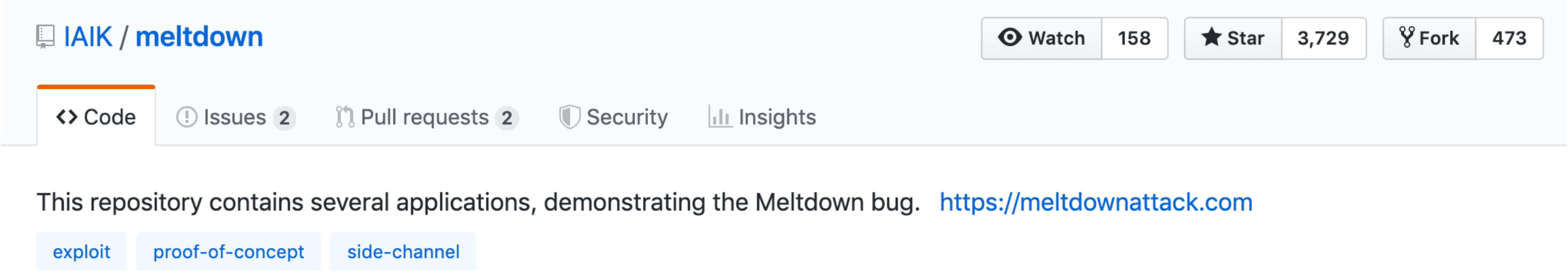
Execution results discarded

Secrets inferred by cache side channel

Race Condition

> *"…, there is a race condition between raising this exception and our attack step 2 (Transmitting the secret) …"*
>
> -- Lipp *et al.*, *Meltdown: Reading Kernel Memory from User Space*

- Is this true? What exactly are racing?
- Can we create better race conditions to increase exploitation success rate?



According to Original Authors' Github…

- **It just does not work on my computer, what can I do?**

  There can be a lot of different reasons for that. We collected a few things you can try:

  - Ensure that your CPU frequency is at the maximum, and frequency scaling is disabled.

These seem too ad-hoc…

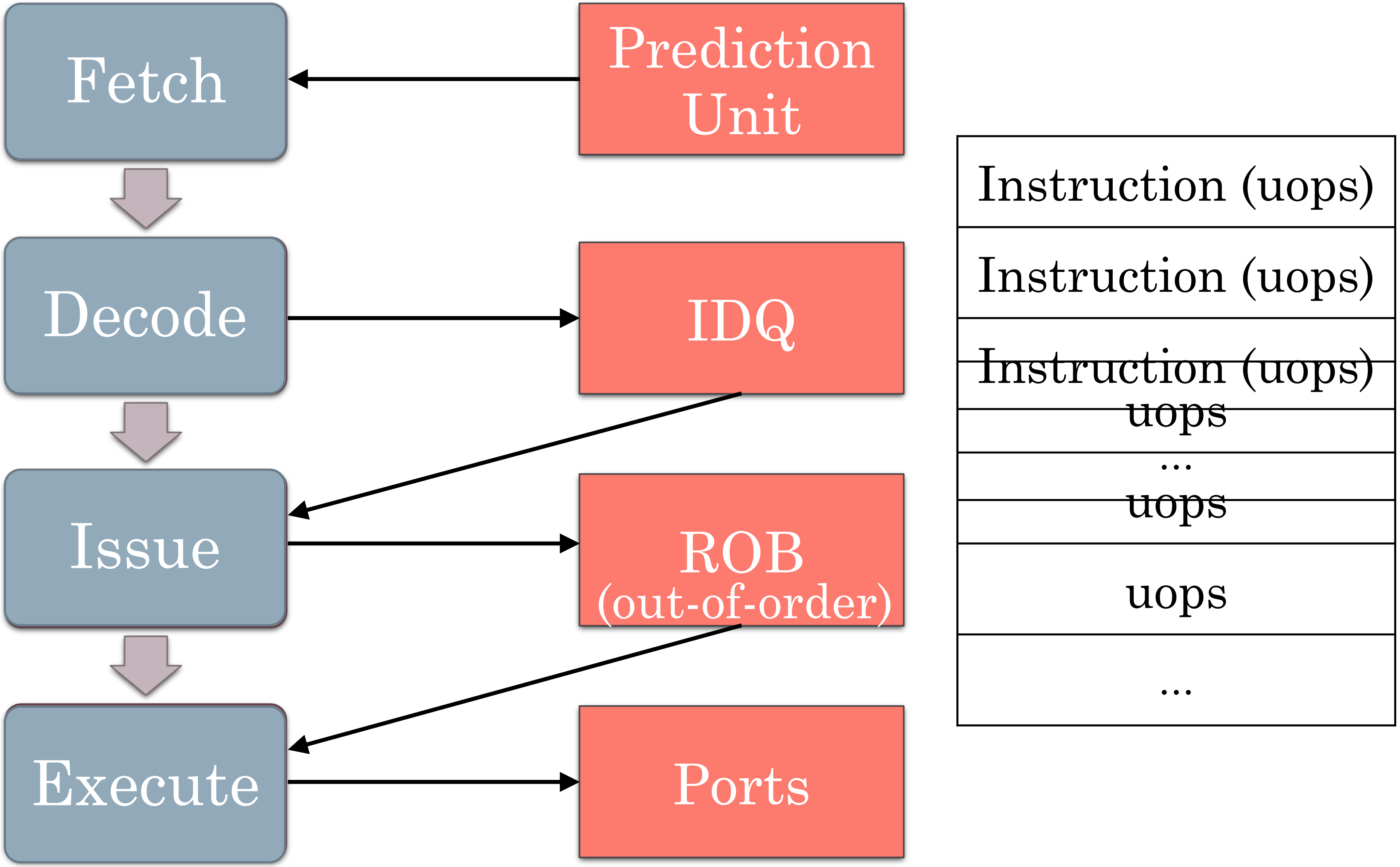
What if we directly peek into the processor hardware?

  - Use a different variant of Meltdown. This can be changed in `libkdump/libkdump.c` in the line `#define MELTDOWN meltdown_nonull`. Try for example `meltdown` instead of `meltdown_nonull`, which works a lot better on some machines (but not at all on others).
  - Try to create many interrupts, e.g. by running the tool `stress` with `stress -i 2` (or other values for the `i` parameter, depending on the number of cores).
  - Try to restart the demos and also your computer. Especially after a standby, the timing are broken on some computers.
  - Play around with the parameters of libkdump, e.g. increase the number of retries and/or measurements.

Overview
(Please Please Stay with Me and Don’t Get Lost)

1. SpeechMiner Framework
2. 2-phase Fault Handling Model
3. Understanding Speech Vulnerabilities

Basic x86 Execution Engine



SpeechMiner

Input: Instruction Sequence → Settings: Execution Environment → Execution → Results: Covert Channel

Infer processor micro-architectural states from covert channel data

- Systematically test the vulnerabilities on specific hardware
- Understand the Speech vulnerabilities better

Instruction Sequence

- Windowing Gadget.
  - Enlarge the speculation window
  - Eliminate side-effects of instruction issuing
- Speculation Primitive.
  - One or two instructions that will raise an exception when executed
  - Generated from Intel manual's list of causes of exceptions
- Disclosure Gadget.
  - Speculatively executed, utilizing covert-channel techniques to measure the speculation windows or the latency of data fetching, *etc*.

An example

```
// %RBX: address of uncached covert channel buffer
// %RDX: address of another uncached memory buffer
// *(%RDX) = %RBX
// %RCX: illegal address whose data is 0x42000
// ------------------------------------------------
// Windowing Gadget
    movq (%rdx), %rdx
// ------------------------------------------------
// Speculation Primitive
    movq (%rcx), %rcx
// ------------------------------------------------
// Disclosure Gadget
    movq (%rbx, %rcx, 1), %rbx
```

* All assembly code follows AT&T syntax.

Systematic Evaluation of Variants

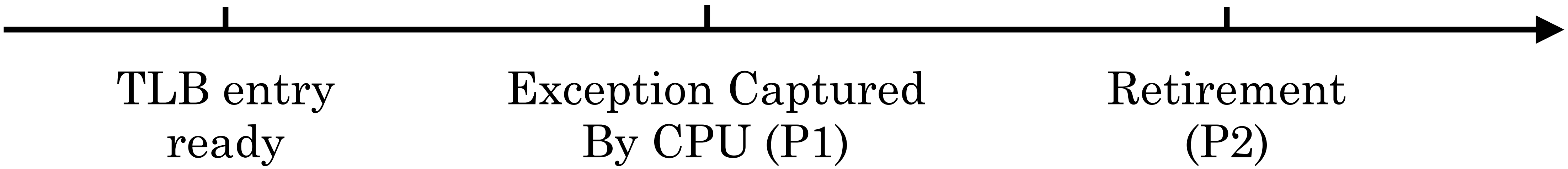
Variant	Laptop 1 KabyLake	Laptop 2 KabyLake	Desktop 1 Haswell-EP	Desktop 2 SandyBridge	Desktop 3 Westmere-EP	Desktop 4 CoffeeLake	Desktop 5 KabyLake	Desktop 6 AMD EPYC	Cloud 1 Skylake-SP
PTE (Present)	Y	N/A	N/A	N/A	N/A	Y	Y	N/A	Y
PTE (Reserved)	Y	N/A	N/A	N/A	N/A	Y	Y	N/A	Y
PTE (US)	Y	Y	Y	Y	Y	Y	Y	R	Y
Load CR4	R	R	R	R	R	R	R	R	R
Load MSR (0x1a2)	R	R	R	R	R	R	R	N/A	N/A
Protection Key (User)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Y
Protection Key (Kernel)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Y
SMAP violation	Y	Y	N/A	N/A	N/A	Y	Y	Y*	Y**
PTE (write w/ RW=0)	Y	Y*	Y	Y	Y	Y	Y	R	Y
Load xmm0 (CR0.TS)	Y	Y	Y	Y	Y	Y	Y	N/A	N/A
BOUND (32-bit)	Y	Y	Y	Y	Y	Y	Y	Y	Y
DS Over-Limit (32-bit)	N	N	N	N	N	N	N	Y	N
SS Over-Limit (32-bit)	N	N	N	N	N	N	N	Y	N/A
DS Not-Present (32-bit)	R	R	R	R	R	R	R	R	R
SS Not-Present (32-bit)	R	R	R	R	R	R	R	R	R
DS Execute-Only (32-bit)	R	R	R	R	R	R	R	R	R
CS Execute-Only (32-bit)	R	R	R	R	R	R	R	R	R
DS Read-Only (write, 32-bit)	Y	Y	Y	Y	Y	Y	Y	R	Y
SS Read-Only (32-bit)	R	R	R	R	R	R	R	R	R
DS Null (32-bit)	N	N	N	N	N	N	N	R	N
SS Null (32-bit)	R	R	R	R	R	R	R	R	R
SS $DPL \neq CPL$ (32-bit)	R	R	R	R	R	R	R	R	R

Exploitability of certain variants are implementation-specific. All tests are done with secret in L1D and TLB entry present.

Overview
(Here Comes the Big Part… Are You Still Here?)

1. SpeechMiner Framework
2. 2-phase Fault Handling Model
3. Understanding Speech Vulnerabilities

2-phase Fault Handling Model



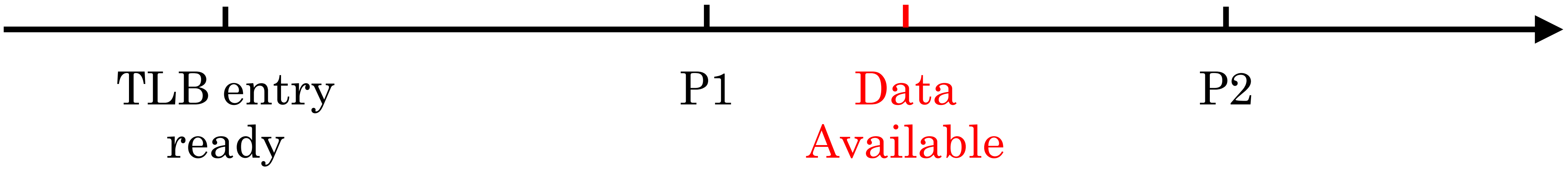
- P1: Processor's exception handling scheme on executing uop
- P2: To commit execution result of the instruction

Retirement (P2)

- Squashes following instructions in ROB
  - Already executed: results discarded; never retires
  - Not executed: never executes
- IDQ stops issuing instructions to ROB and is flushed
- Exception information is saved for exception handler usage
- Frontend is redirected to exception handler

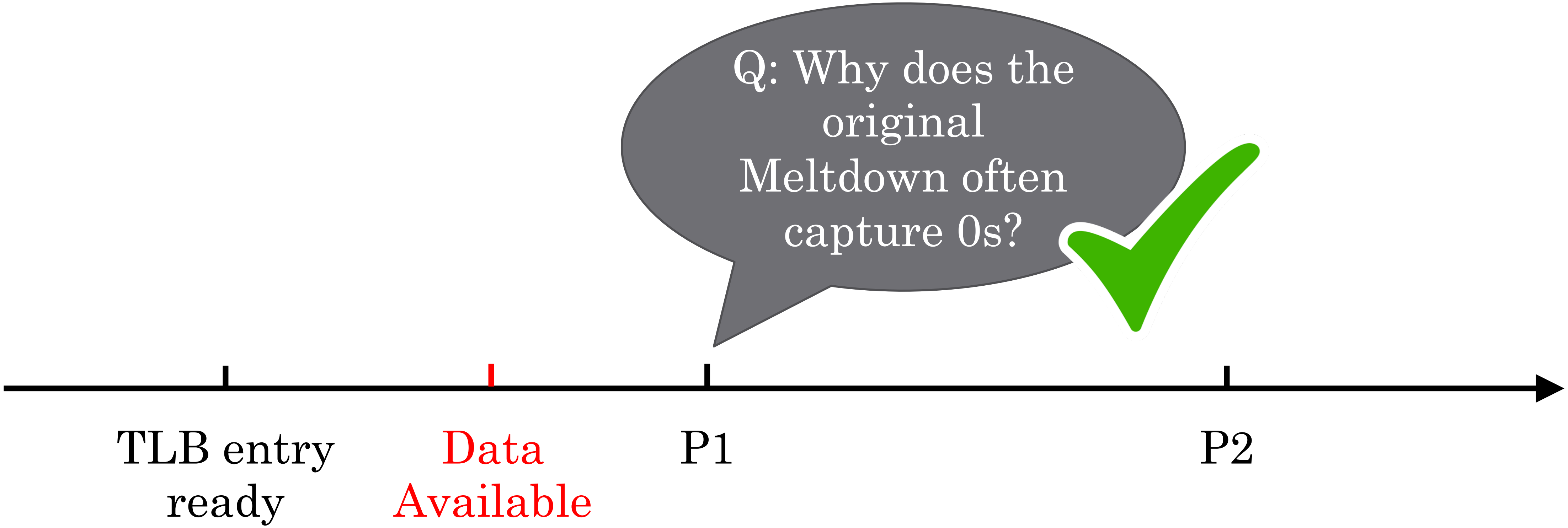
Exception Captured By CPU (P1)

- Assumption: processor's security check takes constant time after TLB is ready (given the same execution environment).
- Change data fetching latency and prove:
  - P1 stops current computation (LD for Meltdown-type)
  - P1 only affects current execution unit



- If data not fetched yet (from memory):
  - Stops fetching
  - Returns dummy value (0) as data

Exception Captured By CPU (P1)



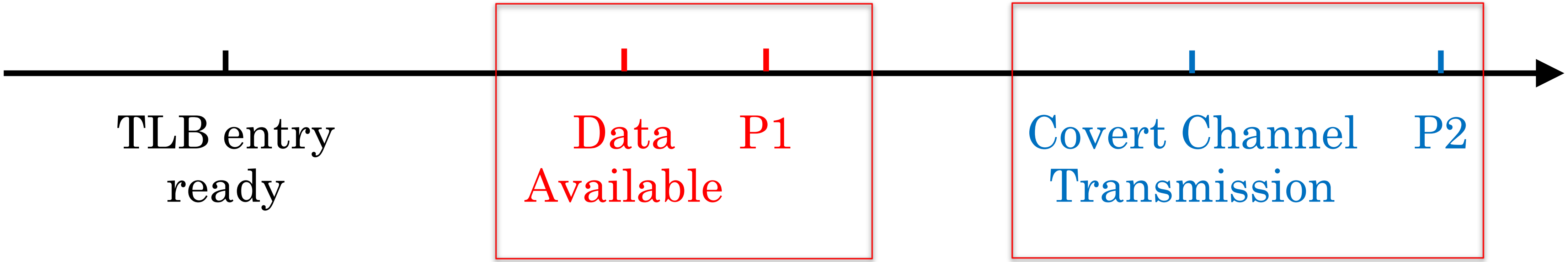
- If data already fetched (from L1D):
  - Data immediately used by following instructions when it is available
  - Nothing to stop at P1

Overview
(It's Almost Over… Hang in There A Little Bit!)

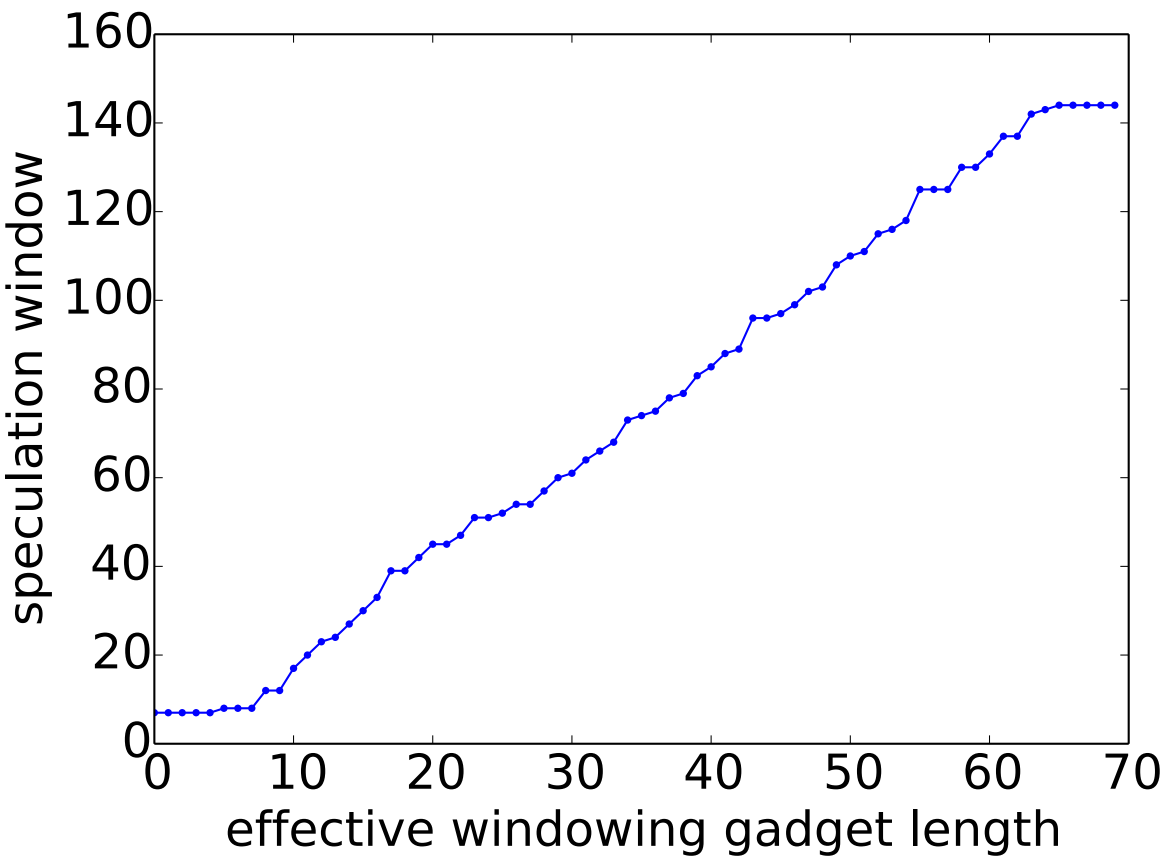
1. SpeechMiner Framework
2. 2-phase Fault Handling Model
3. Understanding Speech Vulnerabilities

The Two Races

- Race I: data fetching *vs.* processor fault handling
- Race II: covert channel transmission *vs.* speculative instruction squashing



Race II Can Always Be Won



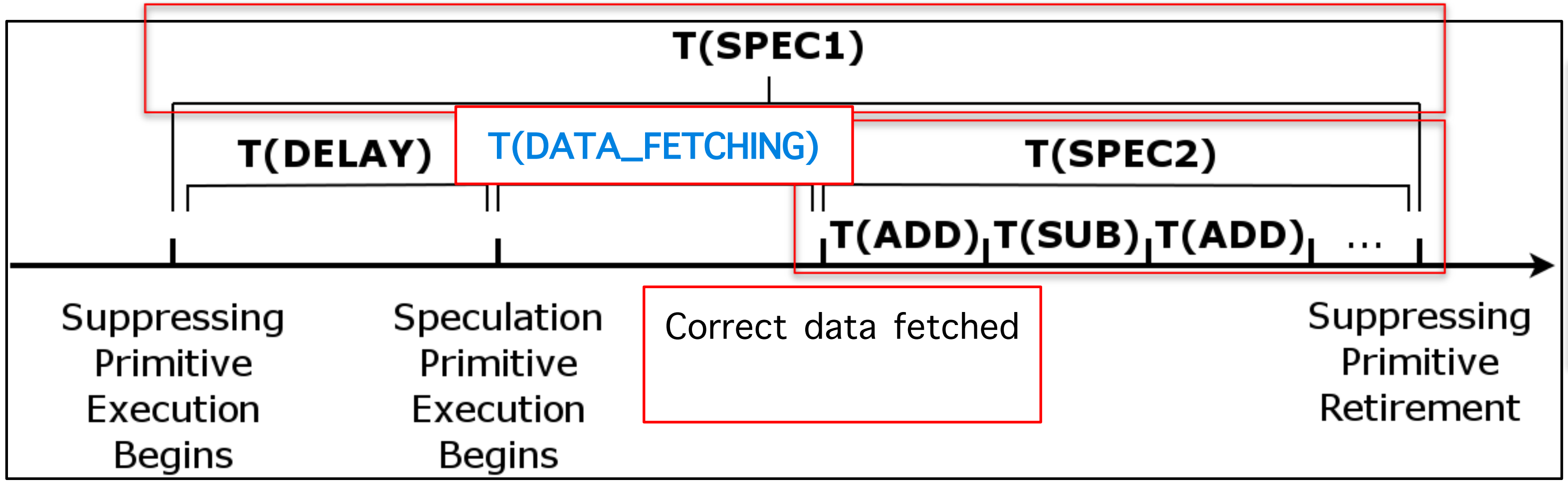

```
// %RBX: address of uncached covert channel buffer
// %RCX: illegal address whose data is 0x42000
// ——————————————————————————————
// Windowing Gadget
  movapd \%xmm0, \%xmm1
  addpd \%xmm1, \%xmm0
  [cpuid]
  mulpd \%xmm1, \%xmm0
  ...
  movapd \%xmm0, \%xmm1
  addpd \%xmm1, \%xmm0
  mulpd \%xmm1, \%xmm0
// ——————————————————————————————
// Speculation Primitive
  movq (%rcx), %rcx
// ——————————————————————————————
// Disclosure Gadget
  [add $1, %rcx]
  [sub $1, %rcx]
  ...
  movq (%rbx, %rcx, 1), %rbx
```

Listing 7: Tuning P2 latency.

Race II: covert channel transmission *vs.* speculative instruction squash

Race I Can Be Quantitatively Measured

(Race I: data fetching *vs.* processor fault handling)




```
// Suppressing Primitive
[MOV (%RAX), %RAX] // legal
[MOV (%RAX), %RAX] // legal
…
MOVQ (%RAX), %RAX // illegal

// Speculation Primitive
MOVQ (%RCX), %RCX // measured

// Disclosure Gadget
[ADD $1, %RCX]
[SUB $1, %RCX]
…
MOVQ (%RBX, %RCX, 1), %RCX
```

- T(SPEC1) = Suppressing Primitive window
- T(SPEC2) = Speculation Primitive window
- T(P1) = T(SPEC1) – T(SPEC2) – T(DELAY)
- Similarly, T(DATA_FETCHING)
  = T(SPEC1) – T'(SPEC2) – T(DELAY)
- Thus, **T(RACE1)**
  = T(DATA_FETCHING) – T(P1)
  = T(SPEC2) – T'(SPEC2)

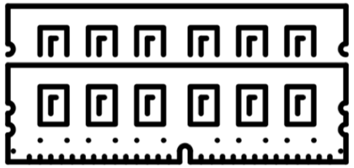
One more thing…

Q: Why can Meltdown-US steal secrets not in L1D while Foreshadow (L1TF) requires that the secrets are in L1D?

- Ou… S
  req…
- A c…
- It i…
  imp…
- Fac…

Uncachable memory

27TH USENIX SECURITY SYMPOSIUM

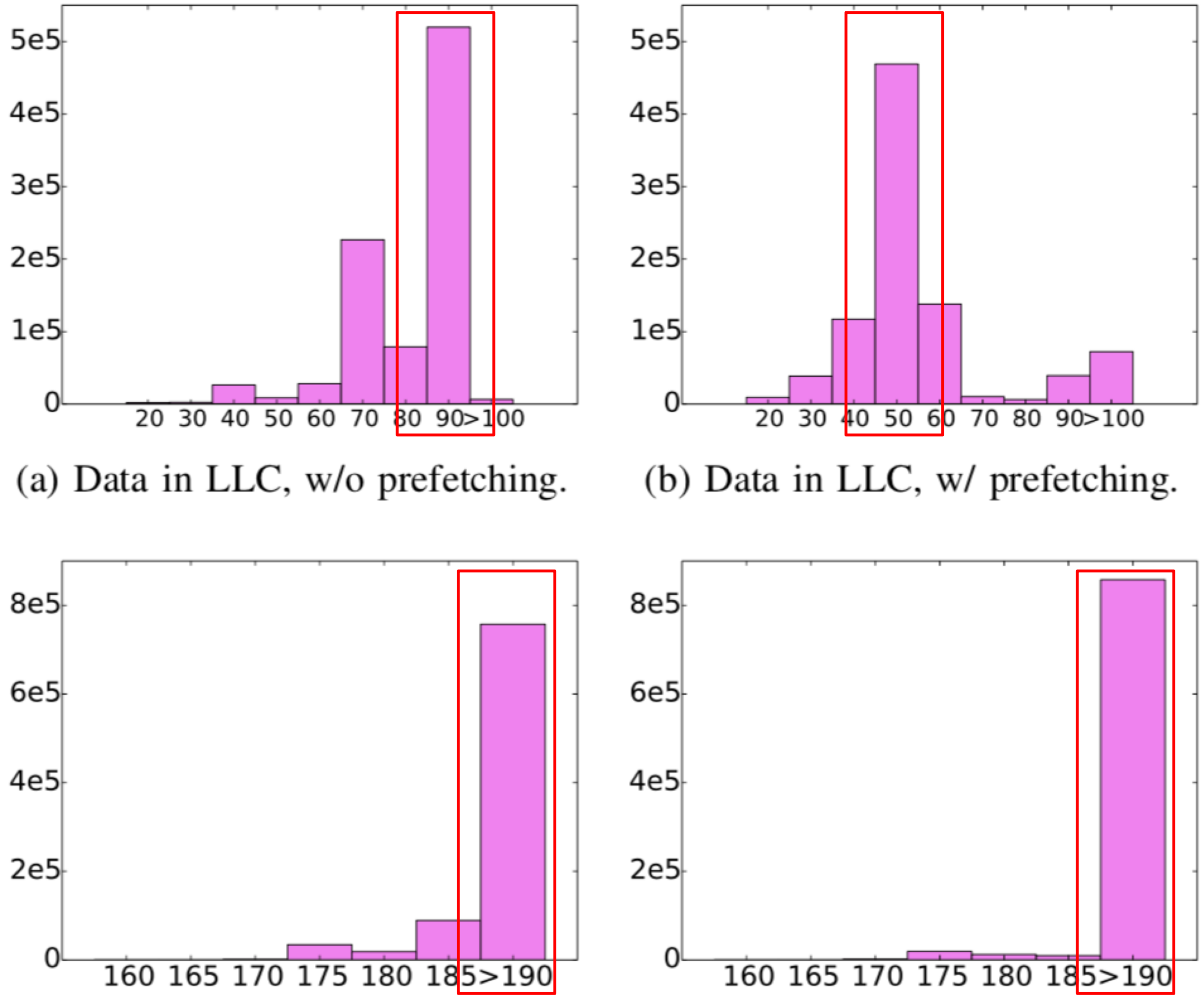


- Mark pages in page tables as UC (uncachable)
  - Every read or write operation will go to main memory
- If the attacker can trigger a legitimate load (system call, ...) on the same CPU core, the data still can be leaked
- Meltdown might read the data from one of the fill buffers
  - as they are shared between threads running on the same core

28

M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg

Study of Prefetching Effects of Meltdown-US



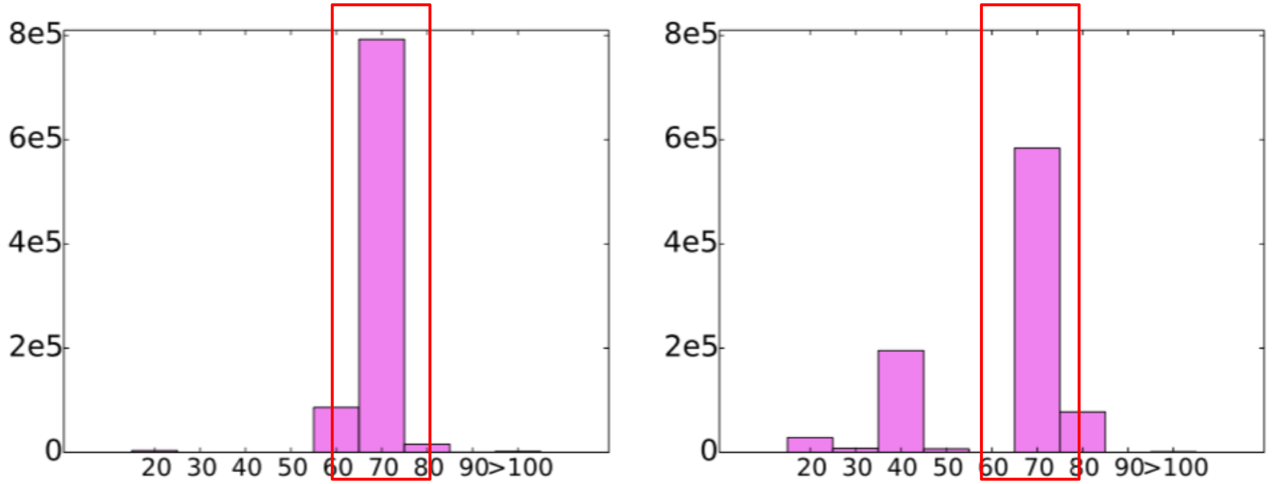
(a) Data in LLC, w/o prefetching. (b) Data in LLC, w/ prefetching.

(c) Data in memory, w/o prefetching. (d) Data in memory, w/ prefetching.

x-axis: access latency; y-axis: frequency of latency

- Experiment:
  1. Force data in certain cache or in memory.
  2. (a) Execute speculation primitive to access the illegal data.
     (b) Go to step 3.
  3. Reload data and measure its access latency.
  4. Repeat for 1,000,000 times and count distribution of reload latency.

Study of Prefetching Effects of Meltdown-P



(e) Terminal fault, data in LLC, w/o prefetching.

(f) Terminal fault, data in LLC, w/ prefetching.

x-axis: access latency; y-axis: frequency of latency

* Meltdown-P is the speculative primitive of L1TF.

Truth of Attacking Non-L1D Secret

- ONE ROUND of Meltdown-US can only fetch L1D data, but its Speculation Primitive is able to “PREFETCH” L2/L3 data into faster cache to facilitate future attacks.
- “PREFETCH” with Speculation Primitive also needs time during speculation. Memory-to-cache seems too slow to finish.
- The Speculation Primitive of Meltdown-P CANNOT “PREFETCH” L2/L3 data into faster cache, probably due to “terminal fault”.
- For claims that Meltdown-US also works for non-cached data, we believe they actually refer to the newly disclosed RIDL-like attacks which leverages LFB whose latency is lower than L1D.

Finally… Thank You!

xiao.465@osu.edu

SpeechMiner:
A Framework for Investigating and Measuring Speculative Execution Vulnerabilities

Yuan Xiao, *Yinqian Zhang*, *Radu Teodorescu*
The Ohio State University