Meltdown, Spectre, ZombieLoad

Daniel Gruss, Moritz Lipp, Michael Schwarz
October 1, 2019

Graz University of Technology
FANTASTIC TIMERS
AND WHERE TO FIND THEM
HIGH-RESOLUTION MICROARCHITECTURAL ATTACKS IN JAVASCRIPT

ANOTHER FLIP IN THE ROW

REAL JAVASCRIPT AND ZERO SIDE-CHANNEL ATTACKS
side channel
= obtaining meta-data and deriving secrets from it

CHANGE MY MIND
Intel Analysis of Speculative Execution Side Channels

White Paper
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc

Let's avoid the term Speculative Side-Channel Attacks.
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc
Speculative Side-Channel Attacks?

- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc
- Let’s avoid the term Speculative Side-Channel Attacks
Revolutionary concept!

Store your food at home, never go to the grocery store during cooking.

Can store ALL kinds of food.

ONLY TODAY INSTEAD OF $1,300

ORDER VIA PHONE: +555 12345
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
CPU Cache

```c
printf("%d", i);
printf("%d", i);
```

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
CPU Cache

printf("%d", i);
printf("%d", i);

Cache miss
Request
Response

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
printf("%d", i);

Cache miss

printf("%d", i);

Cache hit

Cache miss Request
Response
i
printf("%d", i);

Cache hit

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
CPU Cache

- **Cache miss**
- **Cache hit**
- **DRAM access, slow**

```c
printf("%d", i);
printf("%d", i);
```

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
CPU Cache

printf("%d", i);

Cache miss
Request
Response
i

printf("%d", i);

Cache hit
No DRAM access,
much faster
DRAM access,
slow

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
Flush+Reload

Shared Memory

ATTACKER

flush

access

VICTIM

access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush

access

access
Flush+Reload

ATTACKER

Shared Memory

VICTIM

flush
access

access
Flush+Reload

ATTACKER

 Shared Memory

 VICTIM

flush
access

Shared Memory

access
• use pseudo-serializing instruction rdtscp (recent CPUs)
• use pseudo-serializing instruction `rdtscp` (recent CPUs)
• and/or use serializing instructions like `cpuid`
• use pseudo-serializing instruction `rdtscp` (recent CPUs)
• and/or use serializing instructions like `cpuid`
• and/or use fences like `mfence`
Accurate Microarchitecture Timing

- use pseudo-serializing instruction rdtscp (recent CPUs)
- and/or use serializing instructions like cpuid
- and/or use fences like mfence

Intel Publishes Microcode Security Patches, No Benchmarking Or Comparison Allowed!

UPDATE: Intel has resolved their microcode licensing issue which I complained about in this blog post. The new license text is here.
HELLO FROM THE OTHER SIDE (DEMO):
VIDEO STREAMING OVER CACHE COVERT CHANNEL
7. Serve with cooked and peeled potatoes
Wait for an hour
Wait for an hour

LATENCY
1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened
1. Wash and cut vegetables
2. Pick the basil leaves and set aside
3. Heat 2 tablespoons of oil in a pan
4. Fry vegetables until golden and softened
int width = 10, height = 5;

float diagonal = sqrt(width * width  
                   + height * height);
int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
int width = 10, height = 5;

float diagonal = sqrt(width * width + height * height);

int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
*(volatile char*) 0;
array[84 * 4096] = 0;
• Flush+Reload over all pages of the array

Access time [cycles]

Page

0 50 100 150 200 250

500

400

300

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
- Flush+Reload over all pages of the array

- “Unreachable” code line was **actually executed**
Building Meltdown

- Flush+Reload over all pages of the array

- “Unreachable” code line was actually executed
- Exception was only thrown afterwards
• Out-of-order instructions *leave microarchitectural traces*
Out-of-order instructions leave microarchitectural traces.

- We can see them for example through the cache.
Out-of-order instructions leave microarchitectural traces.

- We can see them, for example, through the cache.

- Give such instructions a name: transient instructions.
• Out-of-order instructions leave microarchitectural traces
  • We can see them for example through the cache
• Give such instructions a name: transient instructions
• We can indirectly observe the execution of transient instructions
• Add another *layer of indirection* to test

```c
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```
• Add another layer of indirection to test

```c
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

• Then check whether any part of array is cached
Building Meltdown

- Flush+Reload over all pages of the array

- Index of cache hit reveals data
Building Meltdown

- Flush+Reload over all pages of the array

- Index of cache hit reveals data

- Permission check is in some cases not fast enough
I SHIT YOU NOT
THERE WAS KERNEL MEMORY ALL OVER THE TERMINAL
Kernel Address Isolation to have Side channels Efficiently Removed
Kernel Address Isolation to have Side channels Efficiently Removed
Without KAISER:

Shared address space

User memory

Kernel memory

context switch

With KAISER:

User address space

Not mapped

Kernel address space

SMAP + SMEP

Kernel memory

context switch

addr. space

Interrupt
dispenser
Without KAISER:

Shared address space

User memory  Kernel memory

0  −1

context switch

With KAISER:

User address space

User memory  Not mapped

0  −1

context switch

Kernel address space

SMAP + SMEP  Kernel memory

0  −1

Interrupt dispatcher
Kaiser (Stronger Kernel Isolation) Patches

- Adopted in Linux
- Adopted in Windows
- Adopted in OSX/iOS

Now in every computer
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux
- Adopted in Windows
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux

- Adopted in Windows

- Adopted in OSX/iOS

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
KAISER (Stronger Kernel Isolation) Patches

- Our patch
- Adopted in Linux

- Adopted in Windows
- Adopted in OSX/iOS

→ now in every computer

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
A table for 6 please
Speculative Cooking
index = 0;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 0;

char* data = "textKEY";

if (index < 4)
  then
    LUT[data[index] * 4096]
  else
    0

else
index = 0;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    Speculate

Prediction

0
index = 0;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 1;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 1;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 1;
char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 1;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 2;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 2;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 2;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 2;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 3;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 3;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
0
index = 3;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 3;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 4;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 4;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0

Prediction
index = 4;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 4;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]
index = 5;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
Prediction
0
index = 5;

char* data = "textKEY";

if (index < 4)
then
LUT[data[index] * 4096]
else
0
index = 5;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096]
else
    0
index = 5;

char* data = "textKEY";

if (index < 4)
    LUT[data[index] * 4096];
else
    0
index = 6;

char* data = "textKEY";

if (index < 4)
    
    then

    LUT[data[index] * 4096]

    Prediction

else
    0

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
index = 6;

char* data = "textKEY";

if (index < 4)
    then
        Prediction
    else
        0

LUT[data[index] * 4096]
```c
index = 6;
char* data = "textKEY";
if (index < 4) {
    LUT[data[index] * 4096]
} else {
    0
}
```
index = 6;

char* data = "textKEY";

if (index < 4)
  LUT[data[index] * 4096]
else
  Execute
  0
Transient cause?

**Spectre-type**
- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL [32]

**Meltdown-type**
- Meltdown-NM [86]
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-UD
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP [10, 41]

**prediction fault**
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-UD
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP [10, 41]

**fault type**
- Meltdown-AC
- Meltdown-DE
- Meltdown-PF
- Meltdown-UD
- Meltdown-SS
- Meltdown-BR
- Meltdown-GP [10, 41]

**microarchitectural buffer**
- Spectre-type
- Meltdown-type

**in-place (IP) vs., out-of-place (OP)**
- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL [32]

**Mistraining strategy**
- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL [32]

**Cross-address-space**
- PHT-CA-IP ★
- PHT-CA-OP ★
- PHT-SA-IP [54, 52]
- PHT-SA-OP ★
- BTB-CA-IP [54, 18]
- BTB-CA-OP [54]
- BTB-SA-IP ★
- BTB-SA-OP [18]
- RSB-CA-IP [64, 56]
- RSB-CA-OP [56]
- RSB-SA-IP [64, 56]

**Same-address-space**
- PHT-SA-IP [54, 52]
- PHT-SA-OP ★
- BTB-SA-IP ★
- BTB-SA-OP [18]
- RSB-SA-IP [64]
- RSB-SA-OP [64, 56]
Mitigations?
BLOCKCHAIN
Computer Architecture Today

Informing the broad computing community about current activities, advances and future directions in computer architecture.

Let’s Keep it to Ourselves: Don’t Disclose Vulnerabilities

by Gus Uht on Jan 31, 2019 | Tags: Opinion, Security
### Table 1: Spectre-type defenses and what they mitigate.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Intel</th>
<th>ARM</th>
<th>AMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre-PHT</td>
<td>⬤⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td>⬤⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td>⬤⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Spectre-STL</td>
<td>⬤⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
</tbody>
</table>

Symbols show if an attack is mitigated (●), partially mitigated (○), not mitigated (〇), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (■), or out of scope (◇).
Table 2: Reported performance impacts of countermeasures

<table>
<thead>
<tr>
<th>Defense</th>
<th>Impact</th>
<th>Performance Loss</th>
<th>Benchmark</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvisiSpec</td>
<td>22%</td>
<td>SPEC</td>
<td></td>
</tr>
<tr>
<td>SafeSpec</td>
<td>3% (improvement)</td>
<td>SPEC2017 on MARSSx86</td>
<td></td>
</tr>
<tr>
<td>DAWG</td>
<td>2–12%, 1–15%</td>
<td>PARSEC, GAPBS</td>
<td></td>
</tr>
<tr>
<td>RSB Stuffing</td>
<td>no reports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retpoline</td>
<td>5–10%</td>
<td>real-world workload servers</td>
<td></td>
</tr>
<tr>
<td>Site Isolation</td>
<td>only memory overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLH</td>
<td>36.4%, 29%</td>
<td>Google microbenchmark suite</td>
<td></td>
</tr>
<tr>
<td>YSNB</td>
<td>60%</td>
<td>Phoenix</td>
<td></td>
</tr>
<tr>
<td>IBRS</td>
<td>20–30%</td>
<td>two sysbench 1.0.11 benchmarks</td>
<td></td>
</tr>
<tr>
<td>STIPB</td>
<td>30– 50%</td>
<td>Rodinia OpenMP, DaCapo</td>
<td></td>
</tr>
<tr>
<td>IBPB</td>
<td>no individual reports</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serialization</td>
<td>62%, 74.8%</td>
<td>Google microbenchmark suite</td>
<td></td>
</tr>
<tr>
<td>SSBD/SSBB</td>
<td>2–8%</td>
<td>SYSmark®2014 SE &amp; SPEC integer</td>
<td></td>
</tr>
<tr>
<td>KAISER/KPTI</td>
<td>0–2.6%</td>
<td>system call rates</td>
<td></td>
</tr>
<tr>
<td>L1TF mitigations</td>
<td>-3–31%</td>
<td>various SPEC</td>
<td></td>
</tr>
</tbody>
</table>

© 2014 SE & SPEC integer

Daniel Gruss, Moritz Lipp, Michael Schwarz — Graz University of Technology
FINALLY THE RIGHT SPEED FOR ME
How to find the next big thing ;)
29. **Zombieland: Double Tap** (2019)

Action, Comedy, Horror | Post-production

Columbus, Tallahassee, Wichita, and Little Rock move to the American heartland as they face off against evolved zombies, fellow survivors, and the growing pains of the snarky makeshift family.

**Director:** Ruben Fleischer  |  **Stars:** Emma Stone, Zoey Deutch, Woody Harrelson, Abigail Breslin

Votes: 92,806  |  Gross: $102.09M

---

30. **Love, Death & Robots** (2019–)

TV-MA | 15 min | Animation, Short, Comedy

⭐ 8.7  |  Rate this

A collection of animated short stories that span various genres including science fiction, fantasy, horror and comedy.

**Stars:** Scott Whyte, Nolan North, Matthew Yang King, Michael Benyao

Votes: 58,780

---

31. **iZombie** (2015–)

TV-14 | 42 min | Comedy, Crime, Drama

⭐ 7.9  |  Rate this

A medical resident finds that being a zombie has its perks, which she uses to assist the police.

**Stars:** Rose McIver, Malcolm Goodwin, Rahul Kohli, Robert Buckley

Votes: 54,215
ZOMBIE LOAD ATTACK
When the kernel address is loaded in line 4, it is likely that the CPU already issued the subsequent instructions as part of the out-of-order execution, and that their corresponding $\mu$OPs wait in the reservation station for the content of the kernel address to arrive. As soon as the
fault occurs load operation completed? "intel corp"

Toshiba Boot Error - TechRepublic
https://www.techrepublic.com/.../toshiba-boot-error/ ▼ Diese Seite übersetzen
19.05.2007 - by CaptBilly1Eye · 12 years ago In reply to Toshiba Boot Error ... partition on the floppy disk, hard drive or a CD ROM to load the operating system. ... prior to this situation starting to occur, or if you find that the boot sequence already has the ... Leave the notebook plugged in and undisturbed until completed.

US5751983A - Out-of-order processor with a memory ...
www.google.com/patents/US5751983 - Diese Seite übersetzen
Application filed by Intel Corp ... Hence, a functional unit may often complete a first instruction (which logically precedes a second instruction in the ...... If a fault occurs with respect to the LOAD operation, it is marked as valid and completed.
Meltdown

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, . . .
ALU, FMA, . . .
ALU, Vect, . . .
ALU, Branch

Load data
Load data
Store data
AGU

Memory Subsystem

Load Buffer
Store Buffer

L1 Data Cache

DTLB
LFB

STLB

L2 Cache
L3 Cache

DRAM
...\n
mov al, byte [rcx]
...
Meltdown

Execution Engine
- Reorder buffer
- Scheduler
  - Execution Units:
    - ALU, AES, ...
    - ALU, FMA, ...
    - ALU, Vect, ...
    - ALU, Branch
  - Load data
  - Store data
  - AGU

Memory Subsystem
- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- LFB
- STLB
- L2 Cache
- L3 Cache
- DRAM

Example instruction:
... mov al, byte [rcx] ...
Meltdown

Execution Engine

- Reorder buffer
- Scheduler
- Execution Units
  - ALU, AES...
  - ALU, FMA...
  - ALU, Vect...
  - ALU, Branch
- Load data
- Store data
- AGU

Memory Subsystem

- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- LFB
- STLB
- L2 Cache
- L3 Cache
- DRAM

CDB

Scheduler

\[ \text{mov al, byte [rcx]} \]

 Nah! STOP EVERYTHING!
...mov al, byte [rcx]...
... mov al, byte [rcx] ...
Meltdown

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch

Load data

Store data

AGU

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

L2 Cache

STLB

L3 Cache

DRAM

CDB

Mov al, byte [rcx]
Nope! STOP EVERYTHING!!!
... mov al, byte [rcx] ...

Nope! STOP EVERYTHING!!!
Foreshadow-VMM

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU

Memory Subsystem

Load Buffer
Store Buffer

L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM

CDB

mov al, byte [rcx]
...

Nope! STOP EVERYTHING!!!
mov al, byte [rcx]
...mov al, byte [rcx]...
Foreshadow-VMM

Execution Engine
- Reorder buffer
- Scheduler
- Execution Units
  - ALU, AES, ...
  - ALU, FMA, ...
  - ALU, Vect, ...
  - ALU, Branch

Memory Subsystem
- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- STLB
- L2 Cache
- LFB
- L3 Cache
- DRAM

CDB

mov al, byte [rcx]
... mov al, byte [rcx] ...
...mov al, byte [rcx]...

---

**Foreshadow-VMM**

CDB → Reorder buffer → Scheduler → Execution Engine

- Scheduler: µOP, µOP, µOP, µOP, µOP, µOP, µOP, µOP
- Execution Units: ALU, AES, ALU, FMA, ALU, Vect, ALU, Branch
- Load data → Storage data → AGU

---

**Memory Subsystem**

- L1 Data Cache
- DTLB
- L2 Cache
- STLB
- L3 Cache
- DRAM

---

**Virtual Memory Addressing**

- #n-1 ...
- #n  ppn, vpn, offset, reg.no.
- #n+1 ...

---

**Guest Physical Page Number**

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Nope! STOP EVERYTHING!!

RW  US  WT  UC  R  D  S  G  Ignored

---

Ignored

X
```
... mov al, byte [rcx] ...
```

Foreshadow-VMM

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch

Load data
Store data
AGU

Execution Engine

CDB

Scheduler

Execution Units

ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch

Load data
Store data
AGU

Load Buffer
Store Buffer

L1 Data Cache

DTLB
LFB

STLB

L2 Cache

L3 Cache

DRAM

Guest Physical Page Number

Ignored

X

data can go to register

...mov al, byte [rcx]...

www.tugraz.at
ZombieLoad

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, . . .
ALU, FMA, . . .
ALU, Vect, . . .
ALU, Branch
Load data
Load data
Store data
AQU

Memory Subsystem

Load Buffer
Store Buffer

L1 Data Cache
DTLB
LFB

STLB

L2 Cache

L3 Cache

DRAM
ZombieLoad

Execution Engine
- Reorder buffer
- Scheduler
- Execution Units
  - ALU, AES, ALU, FMA, ALU, Vect, ALU, Branch
  - Load data
  - Store data
  - AQU

Memory Subsystem
- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- STLB
- L2 Cache
- LFB
- L2 Cache
- L3 Cache
- DRAM

...mov al, byte [rcx]...

...mov al, byte [rcx]...
...
ZombieLoad

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...  ALU, FMA, ...  ALU, Vect, ...  ALU, Branch

Load data  Load data  Store data

AGU

Memory Subsystem

Load Buffer  Store Buffer

L1 Data Cache  DTLB  STLB

L2 Cache  L3 Cache  DRAM

CDB

...  mov al, byte [rcx]  ...

µOP  µOP  µOP  µOP  µOP  µOP  µOP  µOP

...  µOP  µOP  µOP  µOP  µOP  µOP  µOP  µOP

complex load situation! need to reissue this load! STOP!!

www.tugraz.at
ZombieLoad

Execution Engine
- Reorder buffer
- Scheduler
  - Execution Units
    - ALU, AES, ...
    - ALU, FMA, ...
    - ALU, Vect, ...
    - ALU, Branch

Memory Subsystem
- Load Buffer
- Store Buffer
  - L1 Data Cache
  - DTLB
  - L2 Cache
  - STLB
  - L3 Cache
  - DRAM

Complex load situation! Need to reissue this load! STOP!!
ZombieLoad

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...

ALU, AES, ...

ALU, AES, ...

ALU, Branch

Load data

Load data

Store data

AGU

Load Buffer

Store Buffer

Load data

Store data

L1 Data Cache

DTLB

STLB

L2 Cache

L3 Cache

DRAM

CDB

Memory Subsystem

#n-1 ...

#n ppn vpn offset reg.no.

#n+1 ...

...mov al, byte[rcx]...

...
ZombieLoad

... mov al, byte [rcx]...

complex load situation! need to reissue this load! STOP!!

#n-1 ...

#n ppn vpn offset reg.no.

#n+1 ...
Zombie Load

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Store data

A GU

CPU

mov al, byte [rcx]

complex load situation! need to reissue this load! STOP!!
ZombieLoad

... mov al, byte [rcx] ...

...
So how did we find it?

- Our Meltdown PoC always worked on non-L1 memory (for us).
- Co-authors confirmed.
- PoCs/reports → Intel - December 2017
- “Can’t reproduce”.
- Works with uncacheable PoC → Intel - March 2018
- “It’s the LFB” → Intel - May 2018.
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
- co-authors confirmed
- PoCs/reports → Intel - December 2017
- "can't reproduce"
- works with uncacheable
- PoC → Intel - March 2018
- "It's the LFB" → Intel - May 2018
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
- co-authors confirmed
- PoCs/reports → Intel - December 2017
  - “can’t reproduce”
- works with uncacheable
- PoC → Intel - March 2018
  - “It’s the LFB” → Intel - May 2018
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
  - co-authors confirmed
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
  - co-authors confirmed
- PoCs/reports → Intel - December 2017
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
  - co-authors confirmed
- PoCs/reports → Intel - December 2017
  - “can't reproduce”
So how did we find it?

• our Meltdown PoC always worked on non-L1 memory (for us)
  • co-authors confirmed
• PoCs/reports → Intel - December 2017
  • “can't reproduce”
• works with uncacheable
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
  - co-authors confirmed
- PoCs/reports → Intel - December 2017
  - “can't reproduce”
- works with uncacheable
  - PoC → Intel - March 2018
So how did we find it?

- our Meltdown PoC always worked on non-L1 memory (for us)
  - co-authors confirmed
- PoCs/reports → Intel - December 2017
  - “can’t reproduce”
- works with uncachable
  - PoC → Intel - March 2018
- “It’s the LFB” → Intel - May 2018
Meltdown Noise?

- Meltdown has noise
- Uncacheable → lower signal to noise ratio
Meltdown Noise?

Meltdown has noise
Uncacheable → lower signal to noise ratio
Meltdown Noise?

Meltdown has noise → Uncacheable → lower signal to noise ratio
• Meltdown has noise
Meltdown Noise?

- Meltdown has noise
- Uncacheable $\rightarrow$ lower signal to noise ratio
NOISE!? ON A COMPLETELY DETERMINISTIC SYSTEM!?
THERE IS NO NOISE

NOISE IS JUST SOMEONE ELSE'S DATA
Unix - Frequently Asked Questions (3/7)

[Frequent posting]

Section - How do I get rid of zombie processes that persevere?

( Part1 - Part2 - Part3 - Part4 - Part5 - Part6 - Part7 - Single Page )

[ Usenet FAQs | Web FAQs | Documents | RFC Index | Houses ]
Why attack research?
We have ignored microarchitectural attacks for many years:
We have ignored microarchitectural attacks for many years:

- attacks on crypto
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer → “only affects cheap sub-standard modules”

for years we solely optimized for performance
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR
How did we get here?

We have ignored microarchitectural attacks for many years:

• attacks on crypto → “software should be fixed”
• attacks on ASLR → “ASLR is broken anyway”
How did we get here?

We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer → “only affects cheap sub-standard modules”
How did we get here?

We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone → “not part of the threat model”
- Rowhammer → “only affects cheap sub-standard modules”

→ for years we solely optimized for performance
• new class of software-based attacks
Conclusions

- new class of software-based attacks
- many problems to solve around microarchitectural attacks and especially transient-execution attacks
Conclusions

- new class of software-based attacks
- many problems to solve around microarchitectural attacks and especially transient-execution attacks
- dedicate more time into identifying problems and not solely in mitigating known problems
Meltdown, Spectre, ZombieLoad

Daniel Gruss, Moritz Lipp, Michael Schwarz
October 1, 2019

Graz University of Technology