Zombieload Attack

Michael Schwarz
Moritz Lipp

black hat
Who am I?

Michael Schwarz
Faculty @ CISPA Helmholtz Center for Information Security

@misc0110
michael.schwarz@cispa.saarland
Who am I?

Moritz Lipp
PhD Candidate @ Graz University of Technology

@mlqxyz

moritz.lipp@iaik.tugraz.at
Microarchitecture
Microarchitecture

Fetch

Execution Engine

Memory Subsystem

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microarchitecture

Frontend

Fetch + Decode

Execution Engine

Memory Subsystem

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microarchitecture

Frontend
- Fetch + Decode

Execution Engine
- Scheduler
- Execution Units
  - ALU, AES
  - ALU, FMA
  - ALU, Vect
  - ALU, Branch
- Load data
- Store data
- AGU

Memory Subsystem
- Write Back

Execute

Fetch + Decode

Instruction Queue
- Instruction Fetch & PreDecode
- Instruction Cache
- ITLB
- Branch Predictors
- L1 Instruction Cache
- L1 Data Cache
- DTLB
- L2 Cache
- L3 Cache
- LFB
- STLB
- DRAM

Write Back

Allocate Queue
- MUX
- 4-Way Decode
- Instruction Queue
- Branch Predictor
- L1 Instruction Cache
- ITLB
- Branch Predictors
- L1 Instruction Cache
- L1 Data Cache
- DTLB
- L2 Cache
- L3 Cache
- LFB
- STLB
- DRAM

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microarchitecture

Frontend
- Instruction Fetch & PreDecode
- Instruction Queue
- 4-Way Decode

Execution Engine

Memory Subsystem
Microarchitecture

Frontend
- Branch Predictor
- μOP Cache

Instruction Fetch & PreDecode
- Instruction Queue
- 4-Way Decode

Execution Engine
- Execution Units (ALU, AES, FMA, Vect, Branch)
- Load data
- Store data

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

Fetch + Decode
- Allocation Queue
- 4-Way Decode
- Instruction Queue
- Instruction Fetch & PreDecode
- BranchPredictor
- μOP Cache

MUX
- μOP
- μOP
- μOP
- μOP

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microarchitecture

Frontend
- Branch Predictor
- \( \mu \text{OP Cache} \)
- Allocation Queue

Execution Engine
- L1 Instruction Cache
- Instruction Fetch & PreDecode
- Instruction Queue
- 4-Way Decode
- MUX

Memory Subsystem
- Fetch/Decode
- Instruction Queue
- ITLB

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microarchitecture

Frontend
- Branch Predictor
- \(\mu\)OP Cache
- Instruction Fetch & PreDecode
- 4-Way Decode
- Allocation Queue

Execution Engine
- L1 Instruction Cache
- Reorder buffer
- Scheduler
- Execution Units
- CDB

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

Fetch + Decode
- Instruction Queue
- Instruction Fetch & PreDecode
- ITLB

\(\mu\)OP cache

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microarchitecture

Frontend
- Branch Predictor
- Instruction Fetch & PreDecode
- Instruction Queue
- 4-Way Decode
- Allocation Queue
- μOP Cache

Execution Engine
- L1 Instruction Cache
- 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
- L2 Cache
- L3 Cache
- DRAM

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microarchitecture

Frontend
- Branch Predictor
- Instruction Fetch & PreDecode
- 4-Way Decode
- Allocation Queue

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
- L2 Cache
- STLB
- L3 Cache
- DRAM

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Caching Speeds Up Memory Accesses

- Access time [CPU cycles]
- Number of accesses
- Cache Hits

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Caching Speeds Up Memory Accesses

![Graph showing cache hits and misses over access times in CPU cycles]
How Do Caches Actually Work?

Physical Address

- PPN
- \(n\) bits
- \(b\) bits

Tag \(\equiv\) PPN

Cache

- Tag
- Data

\(2^n\) cache lines

Cache Index
How Do Caches Actually Work?

Physical Address

- PPN
- Tag ≡ PPN

Cache

- Cache Index
- $2^n$ cache sets

- Tag
- Data

- Way 1 Tag
- Way 1 Data
- Way 2 Tag
- Way 2 Data

$\begin{array}{c|c|c}
\text{PPN} & n \text{ bits} & b \text{ bits} \\
\hline
\text{Tag} & & \\
\hline
\text{Cache sets} & & \\
\end{array}$
How Do Caches Actually Work?

Physical Address

- PPN
- n bits
- b bits

Tag ≡ PPN

Cache

- 2^n cache sets
- Way 1 Tag
- Way 2 Tag
- Way 1 Data
- Way 2 Data

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
How Do Caches Actually Work?

Physical Address

\[ \text{Tag} \equiv \text{PPN} \]

Cache

\[ 2^n \text{ cache sets} \]

\[ \text{Way 1 Tag} \]
\[ \text{Way 2 Tag} \]
\[ \text{Way 1 Data} \]
\[ \text{Way 2 Data} \]

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
char value = kernel[0]
char value = kernel[0]

Page fault (Exception)
char value = kernel[0]

Page fault (Exception)

mem[value]

Out of order
char value = kernel[0]

Page fault (Exception)

mem[value]

Out of order
Meltdown in the Microarchitecture

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

Core Memory
- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- LFB

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

Physical Page Number
Ignored
X
D
S
G
Ignored
P
RW
US
WT
UC
R
D
S
G
Ignored

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown in the Microarchitecture

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown in the Microarchitecture

... mov al, byte [rcx] ...
Meltdown in the Microarchitecture

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown in the Microarchitecture

mov al, byte [rcx]
Meltdown in the Microarchitecture

CDB

Reorder buffer

Scheduler

Execution Units

Execution Engine

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

Physical Page Number

Ignored

X

D

S

G

Ignored

P

RW

US

WT

UC

R

D

S

G

Ignored

mov al, byte [rcx]

...
Meltdown in the Microarchitecture

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown in the Microarchitecture

mov al, byte [rcx]
Meltdown in the Microarchitecture

Execution Engine

Reorder buffer

Scheduler

Execution Units

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

Core Memory

CDB

Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

not used for L1/SB/LFB

mov al, byte [rcx]

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown in the Microarchitecture

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown in the Microarchitecture

```
... mov al, byte [rcx] ...
...
Meltdown in the Microarchitecture

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

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Store data

AGU

CDB

Core Memory

#n-1 ...

#n=ppn vpn offset reg.no.

#n+1 ...

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

P RW US WT UC R D S G Ignored

Physical Page Number

Ignored X

data can go to register

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Performance

Meltdown
Leakage Rate

552.4 kB/s

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Performance

Meltdown Leakage Rate

552.4 kB/s

Meltdown Error Rate

0.003 %

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown Experiment

L1 Cacheline

⋯ XXXXXXXXXXXXXXXX ⋯

Kernel Memory
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)
Meltdown Experiment

L1 Cacheline

XXXXXXXXXXXXXXXXXX

Kernel Memory

Leak (Meltdown)

X  X
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)

X X X
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)

X X X X X
Meltdown Experiment

L1 Cacheline

XXXXXXXXXXXXXXXXXXXX

Kernel Memory

Leak (Meltdown)

X X X X X X

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)

X X X X X X X X X
Meltdown Experiment

L1 Cacheline

Leak (Meltdown)

Kernel Memory
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)

X X X X X X X P X

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)
Meltdown Experiment

L1 Cacheline

XXXXXXXXXXXXXXXX

Kernel Memory

Leak (Meltdown)

XXXXXXXXXXXXXXX
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)
Meltdown Experiment

L1 Cacheline

Leak (Meltdown)

Kernel Memory

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown Experiment

L1 Cacheline

XXX

Kernel Memory

Leak (Meltdown)

X X X X X X X X P X X X X X X X P

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)
Meltdown Experiment

L1 Cacheline

XXXXXXXXXXXXXXXX

Kernel Memory

Leak (Meltdown)

X X X X X X X P X X X X X X X P X X X
Meltdown Experiment

L1 Cacheline

Kernel Memory

Leak (Meltdown)
How to get rid of the noise?
There is no noise.

Noise is just someone else’s data.
Lemma 1: Noise is someone else’s data

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Lemma 1: Noise is someone else’s data

$$\lim_{\rightarrow} \emptyset =$$
Lemma 1: Noise is someone else’s data

\[ \lim_{\rightarrow} = 0 \]
Deep Dive: Intel Analysis of Microarchitectural Data Sampling

Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer.
Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer. Under certain conditions, the fill buffer may speculatively forward data, including stale data,
Deep Dive: Intel Analysis of Microarchitectural Data Sampling

Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer. Under certain conditions, the fill buffer may speculatively forward data, including stale data, to a load operation that will cause a fault/assist.
ZombieLoad Cache-line Conflicts

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Cache-line Conflicts

Mapping $v_2$

Cache line

Page

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Cache-line Conflicts

faulting load

Mapping $v_2$

Page

cache line

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Cache-line Conflicts

faulting load

Mapping $v_2$

Page

cache line

Mapping $v_1$

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Cache-line Conflicts

faulting load
Mapping $v_2$

Page

cache line

flush
Mapping $v_1$

“certain condition”
ZombieLoad Cache-line Conflicts

- Faulting load
- Mapping $v_2$
- Cache line
- Page
- Flush
- Mapping $v_1$

"Certain condition"
Complex Load Situations

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

- Core Memory
  - Load Buffer
  - Store Buffer
  - L1 Data Cache
  - DTLB
  - LFB

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Complex Load Situations

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Complex Load Situations

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Complex Load Situations

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Complex Load Situations

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Complex Load Situations

... mov al, byte [rcx] ...
Complex Load Situations

```
... mov al, byte [rcx] ...
...
Complex Load Situations

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

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

Core Memory
- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- LFB

not used for L1/SB/LFB

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Complex Load Situations

Execution Engine

Scheduler

Execution Units

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

Load data
Store data

AGU

CDB

Reorder buffer

mov al, byte [rcx]

not used for L1/SB/LFB

Load Buffer
Store Buffer

L1 Data Cache

DTLB
LFB

Core Memory

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Complex Load Situations

- Reorder buffer
- Scheduler
- Execution Units: ALU, AES, ALU, FMA, ALU, Vect., ALU, Branch
- Execution Engine: 
  - Execution Units
  - Core Memory: L1 Data Cache, DTLB, LFB
  - Load Buffer, Store Buffer
- CDB

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

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
char value = faulting[0]
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>O</td>
<td>P</td>
</tr>
<tr>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
</tbody>
</table>

\[
\text{char value} = \text{faulting}[0]
\]

Fault
char value = faulting[0]
mem[value]

Fault
Out of order

User Memory

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D E</td>
</tr>
<tr>
<td>F</td>
<td>G H</td>
</tr>
<tr>
<td>I</td>
<td>J K</td>
</tr>
<tr>
<td>L</td>
<td>M N</td>
</tr>
<tr>
<td>O</td>
<td>P Q</td>
</tr>
<tr>
<td>R</td>
<td>S T</td>
</tr>
<tr>
<td>U</td>
<td>V W</td>
</tr>
<tr>
<td>X</td>
<td>Y Z</td>
</tr>
</tbody>
</table>
char value = faulting[0]

mem[value]
ZombieLoad Variant 1

Physical memory

User 2^{47} Kernel –2^{47} –1

0 max

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Variant 1

Physical memory

User

Kernel

flush

0

max

0

$2^{47}$

$-2^{47}$

-1

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Variant 1

Physical memory

User

Kernel

flush

load

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Variant 3

Physical memory

User

Kernel

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Variant 3

Physical memory

User

Kernel

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Variant 3

Physical memory

User

Kernel

flush

load

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microcode Assist (Variant 3)

Instructions

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microcode Assist (Variant 3)
Microcode Assist (Variant 3)

Instructions

Decoder

MUX

Backend

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microcode Assist (Variant 3)

Microcode ROM → Assist → Decoder → MUX → Backend
Microcode Assist (Variant 3)

Microcode ROM

Instructions

Decoder

Assist

MUX

Backend

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Microcode Assist (Variant 3)

- Microcode assist handles rare cases
Microcode Assist (Variant 3)

- Microcode assist handles rare cases
  → Microarchitectural fault
Microcode Assist (Variant 3)

- Microcode assist handles **rare cases**
- Microarchitectural fault
- Setting **accessed/dirty bit** in page table
Microcode Assist (Variant 3)

- Microcode assist handles rare cases
  → Microarchitectural fault
- Setting accessed/dirty bit in page table
  → Regularly reset on Windows

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
• Leak data on same and sibling hyperthread
Attack Targets

- Leak data on same and sibling hyperthread

Applications
Attack Targets

- Leak data on **same** and **sibling** hyperthread

Applications

Operating System

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Attack Targets

- Leak data on same and sibling hyperthread

Applications  Operating System  SGX Enclave
Attack Targets

- Leak data on same and sibling hyperthread

Applications
Operating System
SGX Enclave
Virtual Machine
• Leak data on *same* and *sibling* hyperthread

Applications

Operating System

SGX Enclave

Virtual Machine

Hypervisor
<table>
<thead>
<tr>
<th>Page Number</th>
<th>Page Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meltdown</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Physical</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>12</td>
</tr>
<tr>
<td>Virtual</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Foreshadow</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Physical</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>12</td>
</tr>
<tr>
<td>Virtual</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Fallout</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Physical</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>12</td>
</tr>
<tr>
<td>Virtual</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>Page Number</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Meltdown</strong></td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td>12</td>
</tr>
<tr>
<td>Virtual</td>
<td>12</td>
</tr>
<tr>
<td><strong>Foreshadow</strong></td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td>12</td>
</tr>
<tr>
<td>Virtual</td>
<td>12</td>
</tr>
<tr>
<td><strong>Fallout</strong></td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td>12</td>
</tr>
<tr>
<td>Virtual</td>
<td>12</td>
</tr>
<tr>
<td><strong>ZombieLoad/RIDL</strong></td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td>12</td>
</tr>
<tr>
<td>Virtual</td>
<td>12</td>
</tr>
</tbody>
</table>

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
IMPOSSIBLE
IMPOSSIBLE
key_n (0xD2)

1 1 0 1 0 0 1 0
Control - Domino Attack

key\_n (0x\text{D2})

(4,4)-domino\_n,n+1 (0x21)
**Control - Domino Attack**

<table>
<thead>
<tr>
<th>key(_n) (0xD2)</th>
<th>(4,4)-domino(_{n,n+1}) (0x21)</th>
<th>key(_{n+1}) (0x1C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1</td>
<td>0 0 1 0</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>
Attacker Models

**Variant 1**
Kernel Mapping

- works
- does not work
- can be prevented

**Variant 3**
Microcode-Assisted Page-Table Walk

- works
- does not work
- can be prevented

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Performance

Variant 1
Kernel Mapping

5.30 kB/s
Performance

Variant 1
Kernel Mapping
5.30 kB/s

Variant 3
Microcode-Assisted
Page-Table Walk
7.73 kB/s

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Intel Mitigations

- Disable hyperthreading or group scheduling
Intel Mitigations

- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers

CPU Meltdown Foreshadow RIDL Fallout MLPDS MDSUM ZombieLoad

8th/9th gen. Intel Core Coffee Lake
Intel Xeon Cascade Lake


Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - VERW instruction (microcode update)
Intel Mitigations

- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - VERW instruction (microcode update)
  - Software sequences

8th/9th gen. Intel Core Coffee Lake
Intel Xeon Cascade Lake


Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Intel Mitigations

- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - VERW instruction (microcode update)
  - Software sequences
- New CPUs which are not affected

<table>
<thead>
<tr>
<th>CPU</th>
<th>Meltdown</th>
<th>Foreshadow</th>
<th>RIDL</th>
<th>Fallout</th>
<th>MLPDS</th>
<th>MDSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>8th/9th gen. Intel Core Coffee Lake</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Intel Xeon Cascade Lake</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Intel Mitigations

- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - VERW instruction (microcode update)
  - Software sequences
- New CPUs which are not affected

<table>
<thead>
<tr>
<th>CPU</th>
<th>Meltdown</th>
<th>Foreshadow</th>
<th>RIDL</th>
<th>Fallout</th>
<th>MLPDS</th>
<th>MDSUM</th>
<th>ZombieLoad</th>
</tr>
</thead>
<tbody>
<tr>
<td>8th/9th gen. Intel Core Coffee Lake</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Intel Xeon Cascade Lake</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>

What about second variant?
ZombieLoad Variant 2

faulting load
Mapping $v_2$

Page

cache line

flush
Mapping $v_1$

“certain condition”
ZombieLoad Variant 2

TSX Transaction

faulting load
Mapping v₂

cache line
Page

flush
Mapping v₁

“certain condition”

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Variant 2

“certain condition”

load
Mapping \( \nu_1 \)

flush
Mapping \( \nu_1 \)

cache line

Page

TSX Transaction

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
// Variant 2
flush(mapping);

if (xbegin() == _XBEGIN_STARTED) {
    maccess(lut + 4096 * mapping[0]);
    xend();
}
Abort Transactions

- Data Conflicts

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Abort Transactions

- Data Conflicts
- Limited Transactional Resources
Abort Transactions

- Data Conflicts
- Limited Transactional Resources
- Certain Instructions
  - IO instructions, syscall, . . .
Abort Transactions

- Data Conflicts
- Limited Transactional Resources
- Certain Instructions
  - IO instructions, syscall, ...
- Synchronous Exception Events
  - #BR, #PF, #DB, #BP/INT3, ...
12.2.4.5 Miscellaneous Transactional Aborts

Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution.
12.2.4.5 Miscellaneous Transactional Aborts

Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution. [...] For example, operating systems with timer ticks generate interrupts that can cause transactional aborts.
TAA
TAA

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
NMI, SMI, INTR, …

LOAD Address

LOAD Oracle
TAA

NMI, SMI, INTR, ...

LOAD Address

Leak

LOAD Oracle

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
TAA

XBEGIN

LOAD Address

 Leakage

LOAD Oracle

NMI, SMI, INTR, ...

XEND

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Cache-Line Conflict
Cache-Line Conflict
Cache-Line Conflict

FLUSH Address

LOAD Address

Leak

LOAD Oracle
Attacker Models

Variant 1
Kernel Mapping

Variant 3
Microcode-Assisted Page-Table Walk

- works
- does not work
- can be prevented

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Attacker Models

Variant 1
Kernel Mapping

Variant 2
Transactional Asynchronous Abort

Variant 3
Microcode-Assisted Page-Table Walk

- ⬤ works
- ⬤ does not work
- ⬤ can be prevented

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Performance

Variant 1
Kernel Mapping
5.30 kB/s

Variant 3
Microcode-Assisted Page-Table Walk
7.73 kB/s
Performance

Variant 1
Kernel Mapping
5.30 kB/s

Variant 2
Transactional Asynchronous Abort
39.66 kB/s

Variant 3
Microcode-Assisted Page-Table Walk
7.73 kB/s
Josh Walden @jmw1123 · 19. Nov.
Case of beer on its way/there later this week thanks Daniel! Thanks again for the partnership!

Daniel Gruss @lavados · 13. Nov.
Antwort an @Desertroid und @jmw1123
I'm in favor!
Thanks again Josh!

We already received the case a month ago but only found time this weekend to sit together and enjoy some!

We wish you a merry Christmas and look forward to continue working with Intel next year.

cc @cc0x1f @mlqxyz @misc0110 @tugraz_csbsme tugraz
• Disable Intel TSX
• **Disable Intel TSX**
  • Deactivated by default with new microcode updates on CPUs enumerating MDS_NO
• **Disable Intel TSX**
  - Deactivated by default with new microcode updates on CPUs enumerating MDS_NO

• **VERW** to overwrite affected buffers
Timeline

2019
Timeline

2019
April 12
We report ZombieLoad
2019

April 12  We report ZombieLoad
April 24  Report Variant 2
Timeline

- **2019**
  - April 12: We report ZombieLoad
  - April 24: Report Variant 2
  - May 10: Report TAA on Cascade Lake

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Timeline

2019
April 12  - We report ZombieLoad
April 24  - Report Variant 2
May 10   - Report TAA on Cascade Lake
May 11   - Call with Intel + Embargo

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Timeline

2019

April 12  We report ZombieLoad
April 24  Report Variant 2
May 10  Report TAA on Cascade Lake
May 11  Call with Intel + Embargo
May 14  Disclosure of ZombieLoad (without Variant 2)
Timeline

2019

April 12  •  We report ZombieLoad
April 24  •  Report Variant 2
May 10   •  Report TAA on Cascade Lake
May 11   •  Call with Intel + Embargo
May 14   •  Disclosure of ZombieLoad (without Variant 2)
May 14   •  MDS-resistant CPUs and Mitigations available

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Timeline

2019

April 12  • We report ZombieLoad
April 24  • Report Variant 2
May 10   • Report TAA on Cascade Lake
May 11   • Call with Intel + Embargo
May 14   • Disclosure of ZombieLoad (without Variant 2)
May 14   • MDS-resistant CPUs and Mitigations available
May 16   • Report VERW/Software-Sequences are insufficient
Timeline

2019

April 12  We report ZombieLoad
April 24  Report Variant 2
May 10   Report TAA on Cascade Lake
May 11   Call with Intel + Embargo
May 14   Disclosure of ZombieLoad (without Variant 2)
May 14   MDS-resistant CPUs and Mitigations available
May 16   Report VERW/Software-Sequences are insufficient
Nov 14   Public Disclosure of Variant 2

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
2019

April 12  We report ZombieLoad
April 24  Report Variant 2
May 10  Report TAA on Cascade Lake
May 11  Call with Intel + Embargo
May 14  Disclosure of ZombieLoad (without Variant 2)
May 14  MDS-resistant CPUs and Mitigations available
May 16  Report VERW/Software-Sequences are insufficient
Nov 14  Public Disclosure of Variant 2

2020

January 27  Public Disclosure of L1DES
• Software-sequences and VERW do not work reliably
Intel Mitigations

- Software-sequences and VERW do **not work reliably**
  - Cases where leakage is still visible
L1 Data Eviction Sampling (L1DES)

- L1D
- Fill Buffer
- L2D
L1 Data Eviction Sampling (L1DES)

L1D

Fill Buffer

populate

L2D
L1 Data Eviction Sampling (L1DES)

L1D → Fill Buffer → L2D

populate
L1 Data Eviction Sampling (L1DES)
L1 Data Eviction Sampling (L1DES)

L1D → Fill Buffer (evict) → L2D
L1 Data Eviction Sampling (L1DES)

L1D  🔄

.Fill Buffer

L2D
L1 Data Eviction Sampling (L1DES)

L1D  ⨋

_FILL_BUFFER_

L2D
L1 Data Eviction Sampling (L1DES)

L1D  Fill Buffer

L2D

VERW

zero
L1 Data Eviction Sampling (L1DES)

L1D \(\subseteq\) Fill Buffer

\[
\text{evict} \\
\rightarrow \\
\text{L2D}
\]
L1 Data Eviction Sampling (L1DES)

L1D ➔ Fill Buffer ➔ L2D

evict

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
L1 Data Eviction Sampling (L1DES)

L1D \rightarrow \text{Fill Buffer} \rightarrow L2D

evict
INSIGHTS
Transient Execution Attack Tree

Transient cause

Meltdown-type

- Meltdown-PF
- Meltdown-MCA
Transient Execution Attack Tree

Transient cause

Meltdown-type

Meltdown-PF → Meltdown-US → Meltdown-US-LFB
Variant 1

Meltdown-AD

Meltdown-MCA

Meltdown-AD-LFB
Variant 3

Meltdown-TAA

Meltdown-TAA
Variant 2
ZombieLoad Insights

Address

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Insights

- Instruction Pointer
- Memory-based Side-Channel Attacks
- Address

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
ZombieLoad Insights

Instruction Pointer

Meltdown

Data

Memory-based Side-Channel Attacks

Address

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
Data Sampling (ZombieLoad)

Instruction Pointer

Memory-based Side-Channel Attacks

Data

Meltdown

Address
You can find our proof-of-concept implementation on:

- https://github.com/IAIK/ZombieLoad
Conclusion

• Transient-execution attacks: the gift that keeps on giving
• Transient-execution attacks: the gift that keeps on giving
• Class of Meltdown attacks is larger than expected
Conclusion

- Transient-execution attacks: the gift that keeps on giving
- Class of Meltdown attacks is larger than expected
- CPUs are deterministic - there is no noise
ZombieLoad: Leaking Data on Intel CPUs

https://github.com/IAIK/ZombieLoad

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)
October 2, 2020
• Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. “ZombieLoad: Cross-Privilege-Boundary Data Sampling”. In: CCS. 2019