Transient Execution Attacks

Daniel Gruss
June 20, 2019

Graz University of Technology
Another flip in the row

ROWHAMMER

FANTASTIC TIMERS

AND WHERE TO FIND THEM

HIGH-RESOLUTION MICROARCHITECTURAL ATTACKS IN JAVASCRIPT

JavaScript zero

REAL JAVASCRIPT AND ZERO SIDE-CHANNEL ATTACKS
Why do you have a website?
- Inform journalists and the general public
- Otherwise: completely misleading presentation of your work in the media
- Defend yourself against misleading presentations!

Why do you have fancy names?
- What was CVE-2017-5754 again?
- People will throw things together that don’t belong together
- Names enable unambiguous communication

Why do you need a logo?
- Otherwise: media makes their own
- No control over how inappropriate these are
• Why do you have a website?

Inform journalists and the general public otherwise: completely misleading presentation of your work in the media! Defend yourself against misleading presentations!

What was CVE-2017-5754 again? People will throw things together that don’t belong together! Names enable unambiguous communication!

Otherwise: media makes their own! No control over how inappropriate these are!

Daniel Gruss — Graz University of Technology
• Why do you have a website? → Inform journalists and the general public
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
    → defend yourself against misleading presentations!
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
    → defend yourself against misleading presentations!

• Why do you have fancy names?
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
    → defend yourself against misleading presentations!
• Why do you have fancy names? → what was CVE-2017-5754 again?
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
  → defend yourself against misleading presentations!
• Why do you have fancy names? → what was CVE-2017-5754 again?
  • People will throw things together that don’t belong together
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
    → defend yourself against misleading presentations!

• Why do you have fancy names? → what was CVE-2017-5754 again?
  • People will throw things together that don’t belong together
    → Names enable unambiguous communication
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
    → defend yourself against misleading presentations!

• Why do you have fancy names? → what was CVE-2017-5754 again?
  • People will throw things together that don’t belong together
    → Names enable unambiguous communication

• Why do you need a logo?
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
    → defend yourself against misleading presentations!

• Why do you have fancy names? → what was CVE-2017-5754 again?
  • People will throw things together that don’t belong together
    → Names enable unambiguous communication

• Why do you need a logo?
  • Otherwise: media makes their own
• Why do you have a website? → Inform journalists and the general public
  • Otherwise: completely misleading presentation of your work in the media
    → defend yourself against misleading presentations!

• Why do you have fancy names? → what was CVE-2017-5754 again?
  • People will throw things together that don’t belong together
    → Names enable unambiguous communication

• Why do you need a logo?
  • Otherwise: media makes their own → no control over how inappropriate these are
side channel
= obtaining meta-data and deriving secrets from it

CHANGE MY MIND
Side Channel or not?

- Profiling cache utilization with performance counters?
  - No
- Observing cache utilization with performance counters and using it to infer a crypto key?
  - Yes
- Measuring memory access latency with Flush+Reload?
  - No
- Measuring memory access latency with Flush+Reload and using it to infer keystroke timings?
  - Yes
• Profiling cache utilization with performance counters?
Profiling cache utilization with performance counters? → No
Side Channel or not?

- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key?
• Profiling cache utilization with performance counters? → No
• Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
• Profiling cache utilization with performance counters? → No
• Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
• Measuring memory access latency with Flush+Reload?
- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
- Measuring memory access latency with Flush+Reload? → No
• Profiling cache utilization with performance counters? → No
• Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
• Measuring memory access latency with Flush+Reload? → No
• Measuring memory access latency with Flush+Reload and using it to infer keystroke timings?
• Profiling cache utilization with performance counters? → No
• Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
• Measuring memory access latency with Flush+Reload? → No
• Measuring memory access latency with Flush+Reload and using it to infer keystroke timings? → Yes
Speculative Side-Channel Attacks?

Let’s avoid the term Speculative Side-Channel Attacks!

Let’s be more precise!

Then we can think about actual mitigations.
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

Let's avoid the term Speculative Side-Channel Attacks. Let's be more precise!
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
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**
- Let’s be more precise

Daniel Gruss — Graz University of Technology
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
- Let’s be more precise
- → then we can think about actual mitigations

Daniel Gruss — Graz University of Technology
Speculative Side-Channel Attacks?

side channels

Spectre

Meltdown

Daniel Gruss — Graz University of Technology
Speculative Side-Channel Attacks?

side channels  Spectre  Bug fixing
Back to Work
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
```c
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 %dx%d = %d\n", width, height, area);
*(volatile char*) 0;
array[84 * 4096] = 0;
• Flush+Reload over all pages of the array
Building Meltdown

- Flush+Reload over all pages of the array

- “Unreachable” code line was actually executed
• 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*) 0xfffffffff81a000e0;
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**
- Flush+Reload over all pages of the array

- **Index** of cache hit reveals **data**
• 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
used with authorization from Sili
con Graphics, Inc. However, the authors make no claim that M
esa. is in any way a compatible replacement for OpenGL or associ-
ated with. Silicon Graphics, Inc

... This versi
on of Mesa provi
des GLX and DRI capabilities: it is capable of.
both direct and indirec
rendering. For direct
rendering, it ca
use DRI. modu
les from the libg
mschwarz@lab06:~/Documents$
attacker@meltdown ~/exploit %

victim@meltdown ~ %
• Basic Meltdown code leads to a crash (segfault)
• Basic Meltdown code leads to a crash (segfault)
• How to prevent the crash?
• Basic Meltdown code leads to a crash (segfault)
• How to prevent the crash?

Fault Handling
Fault Suppression
Fault Prevention
• Intel TSX to suppress exceptions instead of signal handler

```c
if (xbegin() == XBEGIN_STARTED) {
    char secret = *(char*) 0xffffffff81a000e0;
    array[secret * 4096] = 0;
    xend();
}

for (size_t i = 0; i < 256; i++) {
    if (flush_and_reload(array + i * 4096) == CACHE_HIT) {
        printf("%c\n", i);
    }
}
```
**Speculative execution to prevent exceptions**

```c
int speculate = rand() % 2;
size_t address = (0xffffffff81a000e0 * speculate) +
((size_t)&zero * (1 - speculate));

if (!speculate) {
    char secret = *(char*) address;
    array[secret * 4096] = 0;
}

for (size_t i = 0; i < 256; i++) {
    if (flush_and_reload(array + i * 4096) == CACHE_HIT) {
        printf("%c\n", i);
    }
}
```
Foreshadow / Foreshadow-NG

Boot from ROM...
early console in extract_kernel
input_data: 0x0000000001e0a276
input_len: 0x00000000003d48f8
output: 0x0000000000100000
output_len: 0x00000000001bc258
kernel_total_size: 0x0000000000dec000
booted via startup_32()
Physical KASLR using RDTSC...
Virtual KASLR using RDTSC...

Decompressing Linux... Parsing ELF... Performing relocations... done.
Booting the kernel.

L1 Terminal Fault

Run reader <pfn> [<cache miss threshold>] to leak hypervisor data from the L1
Kernel Address Isolation to have Side channels Efficiently Removed
Kernel Address Isolation to have Side channels Efficiently Removed

KAISER /ˈkʌɪzə/
1. [german] Emperor, ruler of an empire
2. largest penguin, emperor penguin
Without KAISER:

Shared address space

User memory \[ \text{context switch} \] Kernel memory
**Without KAISER:**

Shared address space

User memory → Kernel memory

0 → −1

circuit switch

**With KAISER:**

User address space

User memory → Not mapped

Address space

Interrupt dispatcher

SMAP + SMEP

Kernel address space

0 → −1

circuit switch

Daniel Gruss — 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
Our patch

Adopted in

Linux
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
Our patch
Adopted in Linux

Adopted in Windows

Adopted in OSX/iOS

→ now in every computer
A table for 6 please
»A table for 6 please«
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

Prediction
index = 0;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)
{
    LUT[data[index] * 4096]
}

else
{
    0
}
index = 1;

char* data = "textKEY";

\[
\begin{align*}
\text{if} & \ (\text{index} < 4) \\
\text{then} & \quad \text{LUT}[\text{data[index]} \times 4096] \\
\text{else} & \quad 0 \\
\end{align*}
\]
index = 1;

char* data = "textKEY";

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

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 = 2;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0

Prediction
index = 2;

char* data = "textKEY";

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

Prediction
index = 2;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

0
index = 2;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0

Prediction

Spectre-PHT (v1)
index = 3;

char* data = "textKEY";

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

else
index = 3;

char* data = "textKEY";

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

char* data = "textKEY";

if (index < 4)

 Speculate

 LUT[data[index] * 4096]

then

Prediction

else

0
index = 3;

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)
then
  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)
then

LUT[data[index] * 4096]

else

Prediction

Execute

0

Spectre-PHT (v1)

Daniel Gruss — Graz University of Technology
index = 5;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

then

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)

then

LUT[data[index] * 4096]

else

0
index = 5;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

else

Execute

0
index = 6;

char* data = "textKEY";

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

char* data = "textKEY";

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

Prediction
index = 6;

char* data = "textKEY";

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

char* data = "textKEY";

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

if (index < 4)

LUT[data[index] * 4096]  0

Spectre-STL (v4): Ignore sanitizing write access and use unsanitized old value instead
Animal* a = bird;

a->move()

fly()

swim()

swim()

LUT[data[a->m] * 4096]

0
Animal* a = bird;

LUT[data[a->m] * 4096]
Animal* a = bird;

a->move()

fly()

swim()

Prediction

swim()

LUT[data[a->m] * 4096] 0
Animal* a = bird;

LUT[data[a->m] * 4096]
Spectre v2

```c
Animal* a = bird;
```

```
LUT[data[a->m] * 4096]
```
Animal* a = bird;

LUT[data[a->m] * 4096]

Speculate

fly()

Prediction

0

a->move()

fly()

swim()
\textbf{Spectre v2}

```c
Animal* a = bird;

a->move()

fly()

fly()

swim()

LUT[data[a->m] * 4096]

0
```
Spectre v2

Animal* a = fish;

a->move()

fly()
fly()

swim()

Prediction

LUT[data[a->m] * 4096] 0
Animal* a = fish;

a->move()

Speculate

fly()

Prediction

LUT[data[a->m] * 4096]

0

Spectre v2

Spectre-BTB (v2): mistrain BTB

mispredict indirect jump/call

Spectre-RSB (v5): mistrain RSB

mispredict return
Animal* a = fish;

a->move()

fly()

fly()

swim()

LUT[data[a->m] * 4096]
Spectre v2

\[
\text{Animal* } a = \text{fish};
\]

\[
a-&gt;\text{move()}
\]

\[
\text{fly()}
\]

\[
\text{fly()}
\]

\[
\text{swim()}
\]

\[
\text{Prediction}
\]

\[
\text{Execute}
\]

\[
\text{LUT[}\text{data[a-&gt;m] } \times 4096]\]

\[
0
\]
Animal* a = fish;

a->move()

fly()

swim()

Prediction

swim()

LUT[data[a->m] * 4096] 0
Animal* a = fish;

a->move()

fly()

swim()

swim()

LUT[data[a->m] * 4096]

0

Spectre-BTB (v2): mistrain BTB → mispredict indirect jump/call
Animal* a = fish;

a->move()

fly()

swim()

LUT[data[a->m] * 4096]

0

Spectre-BTB (v2): mistrain BTB → mispredict indirect jump/call

Spectre-RSB (v5): mistrain RSB → mispredict return
v1.1: Speculatively write to memory locations

• v1.1: Speculatively write to memory locations
  → Many more gadgets than previously anticipated

v1.1: Speculatively write to memory locations
→ Many more gadgets than previously anticipated
v1.2: Ignore writable bit

• v1.1: Speculatively write to memory locations
  → Many more gadgets than previously anticipated
• v1.2: Ignore writable bit
  → = Meltdown-RW

operation #n

prediction

operation #n+2

possibly architectural transient execution

predict CF/DF

flush pipeline on wrong prediction

time
Meltdown

operation \#n

exception

raise

data

data dependency

possibly
architectural

transient execution

operation \#n+2

retire

retire

retire

Time
Mistraining Location

<table>
<thead>
<tr>
<th>Location</th>
<th>Victim</th>
<th>Attacker</th>
</tr>
</thead>
<tbody>
<tr>
<td>out-of-place/same-address-space</td>
<td>Congruent branch</td>
<td>Congruent branch</td>
</tr>
<tr>
<td>in-place/same-address-space</td>
<td>Victim branch</td>
<td>Shadow branch</td>
</tr>
</tbody>
</table>

Address collision: 

Shared Branch Prediction State

Daniel Gruss — Graz University of Technology
Classification Tree

Transient cause?

Spectre-type
microarchitectural buffer

Meltdown-type
fault type

in-place (IP) vs., out-of-place (OP)
mistraining strategy

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]
RSB-SA-OP [64, 56]

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

Cross-address-space
Same-address-space

Meltdown-US [61]
Meltdown-P [93, 96]
Meltdown-RW [52]
Meltdown-PK ★
Meltdown-XD ★
Meltdown-SM ★
Meltdown-MPX [44]
Meltdown-BND ★
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
## Mitigations

### Table 1: Spectre-type defenses and what they mitigate.

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

Symbols show if an attack is mitigated (●), partially mitigated (○), not mitigated (□), theoretically mitigated (■), theoretically impeded (■), not theoretically impeded (□), or out of scope (◇).

Daniel Gruss — Graz University of Technology
## 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>SYSmrk®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>
Meltdown

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

CDB

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

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

Meltdown

Memory Subsystem

CDB

Execution Engine

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

L1 Data Cache

DTLB

LFB

L2 Cache

STLB

L3 Cache

DRAM

Load Buffer

Store Buffer

Physical Page Number

Ignored
...
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

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

Meltdown

www.tugraz.at
Meltdown

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

www.tugraz.at
Meltdown

Reorder buffer

Scheduler

Execution Units

Load data

AGU

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Store data

AGU

Load Buffer

Store Buffer

L1 Data Cache

DTLB

STLB

L2 Cache

L3 Cache

DRAM

Memory Subsystem

Execution Engine

CDB

mov al, byte [rcx]

...
Meltdown

Execution Engine

CDB

Reorder buffer

Scheduler

Execution Units

Load data

Store data

AGU

L1 Data Cache

DTLB

LFB

L2 Cache

STLB

L3 Cache

DRAM

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

Physical Page Number

Ignored

X

mem

ppn

vpn

offset

reg.no.

#n-1 ...

#n ...

#n+1 ...

P

RW

US

WT

UC

R

D

S

G

Ignored

Physical Page Number

Memory Subsystem

Load Buffer

Store Buffer

Nope! STOP EVERYTHING!!
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

L1 Data Cache

DTLB

LFB

L2 Cache

STLB

L3 Cache

DRAM

Physical Page Number

Ignored

Physical Page Number

Ignored

P RW US WT UC RD SG Ignored

physical Page Number

Ignored

X

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

Nope! STOP EVERYTHING!!!
Meltdown

Execution Engine

Reorder buffer

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Store data

AGU

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Store data

AGU

CDB

Memory Subsystem

L1 Data Cache

DTLB

LFB

L2 Cache

STLB

L3 Cache

DRAM

Physical Page Number

Ignored

Physical Page Number

Ignored

X

data can go to register

mov al, byte [rcx]

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

Foreshadow-VMM
Foreshadow-VMM

Execution Units:
- ALU
- AES
- FMA
- Vect
- Branch

Scheduler

Reorder buffer

Memory Subsystem:
- L1 Data Cache
- DTLB
- LFB
- STLB
- L2 Cache
- L3 Cache
- DRAM

Load Buffer
Store Buffer

Load data
Store data
AGU

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

Foreshadow-VMM

CDB

Execution Engine

Scheduler

Reorder buffer

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

Guest Physical Page Number

Ignored

Ignored

Guest Physical Page Number

X

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

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

Foreshadow-VMM

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

Memory Subsystem
- Load Buffer
- Store Buffer
- L1 Data Cache
- DTLB
- LFB
- STLB
- L2 Cache
- L3 Cache
- DRAM
The diagram illustrates the architecture of a computer system with a focus on the execution engine and memory subsystem. Key components include:

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

- **Memory Subsystem**:
  - L1 Data Cache
  - DTLB
  - STLB
  - L2 Cache
  - LFB
  - L3 Cache
  - DRAM

Detailed instructions shown in the diagram:

- `mov al, byte [rcx]`

The layout indicates the flow of operations through the execution engine and the interaction with the memory subsystem.
Foreshadow-VMM

www.tugraz.at

Execution Engine

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

Reorder buffer

μOP μOP μOP μOP μOP μOP μOP μOP μOP

mov al, byte [rcx]

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

CDB

µOP µOP µOP µOP µOP µOP µOP µOP µOP

Load Buer

Store Buer

Ignored

Guest Physical Page Number

Ignored

X

#n...#n+1...

#n ppn vpn offset reg.no.

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

CDB

memory subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

Nope! STOP EVERYTHING!

Load Buer

Store Buer

Guest Physical Page Number

Ignored

X

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

Foreshadow-VMM

Execution Engine

Scheduler

Reorder buffer

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Load data

Store data

AGU

CDB

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

L2 Cache

STLB

L3 Cache

DRAM

Guest Physical Page Number

Ignored

Guest Physical Page Number

Ignored

X

Nope! STOP EVERYTHING!!!

www.tugraz.at
mov al, byte [rcx]

...
mov al, byte [rcx]
ZombieLoad

Reorder buffer

Scheduler

Execution Units

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

Load buffer
Store buffer

L1 Data Cache

DTLB

L2 Cache

STLB

L3 Cache

DRAM

Load data
Store data

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

Memory Subsystem

Execution Engine
The execution engine contains a scheduler and execution units. The scheduler allocates operations to execution units. Execution units include ALUs, AES, FMA, Vect, and Branch. The reorder buffer manages the order of operations. The memory subsystem includes a load buffer, store buffer, L1 data cache, DTLB, STLB, L2 cache, L3 cache, and DRAM. The diagram shows the flow of instructions, with an example operation `mov al, byte [rcx]`. This operation involves loading a byte from memory into an ALU for further processing. The memory subsystem handles complex load situations where data needs to be reissued or stopped.
ZombieLoad

Execution Engine

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

Load data
Store data
AGU

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

Memory Subsystem

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

Execution Engine

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

CDB

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

STLB

L2 Cache

L3 Cache

DRAM

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

complex load situation! need to reissue this load! STOP!!
... mov al, byte [rcx] ...

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

ZombieLoad
ZombieLoad

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

complex load situation! need to reissue this load! STOP!!
... mov al, byte [rcx] ...

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

Execution Engine

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

L2 Cache

L3 Cache

DRAM

complex load situation! need to reissue this load! STOP!!
data can go to register
AT LEAST IT'S A LOCAL ATTACK
Truly remote attacks...

Just a few examples:

Remote timing attacks on crypto ([Ber04; BB05] and many more)

ThrowHammer and NetHammer

NetSpectre
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
- ThrowHammer and NetHammer
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
- ThrowHammer and NetHammer
- NetSpectre
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”
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR
We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
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”
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
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
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips
→ it’s an optimization problem
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips
  → it’s an optimization problem
    • what if “too aggressive” changes over time?
... and we’re still optimizing for performance

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips
→ it’s an optimization problem
  - what if “too aggressive” changes over time?
  → difficult to optimize with an intelligent adversary
Conclusions

- 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
Transient Execution Attacks

Daniel Gruss
June 20, 2019
Graz University of Technology
References


