Software-based
Microarchitectural Attacks

Daniel Gruss
May 30, 2018
Graz University of Technology
Fallout following my thesis

MELTDOWN

SPECTRE

KAISER
INTEL REVEALS DESIGN FLAW THAT COULD ALLOW HACKERS TO ACCESS DATA
DEVELOPING STORY

COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES
GLOBAL

COMPUTER CHIP SCARE
The bugs are known as 'Spectre' and 'Meltdown'
<table>
<thead>
<tr>
<th></th>
<th>Intel (Prev)</th>
<th>Intel (After Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>45.26</td>
<td>44.85</td>
</tr>
<tr>
<td>Change</td>
<td>-1.59</td>
<td>-0.41</td>
</tr>
<tr>
<td>Percent Change</td>
<td>[-3.39%]</td>
<td>[-0.91%]</td>
</tr>
</tbody>
</table>

**SHROUT: ISSUE NOT UNIQUE TO INTEL, BUT IT’S AFFECTED THE MOST**
Der Kernschmelzer

Daniel Gruss hat eine schwere Sicherheitslücke in Computerchips entdeckt. Warum gelingt dem Informatiker, woran die Hersteller scheitern?

Von Jens Tönnesmann

Meltdown (security vulnerability)

From Wikipedia, the free encyclopedia

Meltdown is a hardware vulnerability affecting Intel x86 microprocessors and some ARM-based microprocessors.[1][2][3] It allows a rogue process to read all memory, even when it is not authorized to do so.

Meltdown affects a wide range of systems. At the time of disclosure, this included all devices running any but the most recent and patched versions of iOS,[4] Linux,[5][6] macOS,[4] or Windows. Accordingly, many servers and cloud services were impacted,[7] as well as a potential majority of smart devices and embedded devices using ARM based processors (mobile devices, smart TVs and others), including a wide range of networking equipment. A purely software workaround to Meltdown has been assessed as slowing computers between 5 and 30 percent in certain specialized workloads,[8] although companies responsible for software correction of the exploit are reporting minimal impact from general benchmark testing.[9]

Meltdown was issued a Common Vulnerabilities and Exposures ID of CVE-2017-5754[4], also known as Rogue Data Cache Load,[2] in January 2018. It was disclosed in conjunction with another exploit, Spectre, with which it shares some, but not all characteristics. The Meltdown and Spectre vulnerabilities are considered "catastrophic"
Spectre (security vulnerability)

From Wikipedia, the free encyclopedia

Spectre is a vulnerability that affects modern microprocessors that perform branch prediction. On most processors, the speculative execution resulting from a branch misprediction may leave observable side effects that may reveal private data to attackers. For example, if the pattern of memory accesses performed by such speculative execution depends on private data, the resulting state of the data cache constitutes a side channel through which an attacker may be able to extract information about the private data using a timing attack.

Two Common Vulnerabilities and Exposures IDs related to Spectre, CVE-2017-5753 and CVE-2017-5715 (bounds check bypass) and CVE-2017-5715 (branch target injection), have been issued. JIT engines used for JavaScript were found vulnerable. A website can read data stored in the browser for another website, or the browser's memory itself.

Several procedures to help protect home computers and related devices from the Spectre (and Meltdown) security vulnerabilities have been published. Spectre patches have been reported to significantly slow down performance, especially on older computers; on the newer 8th generation Core platforms, benchmark performance drops of 2–14 percent have been measured. Meltdown patches may also produce performance loss. On January 18, 2018, unwanted reboots, even for newer Intel chips, due to
THE MELTDOWN AND SPECTRE EXPLOITS USE "SPECULATIVE EXECUTION?" WHAT'S THAT?

YOU KNOW THE TROLLEY PROBLEM? WELL, FOR A WHILE NOW, CPU's HAVE BASICALLY BEEN SENDING TROLLEYS DOWN BOTH PATHS, QUANTUM-STYLE, WHILE AWAITING YOUR CHOICE. THEN THE UNNEEDED "PHANTOM" TROLLEY DISAPPEARS.

THE PHANTOM TROLLEY ISN'T SUPPOSED TO TOUCH ANYONE. BUT IT TURNS OUT YOU CAN STILL USE IT TO DO STUFF. AND IT CAN DRIVE THROUGH WALLS.

THAT SOUNDS BAD. HONESTLY, I'VE BEEN ASSUMING WE WERE DOOMED EVER SINCE I LEARNED ABOUT ROUHAMMER.

WHAT'S THAT? IF YOU TOGGLE A ROW OF MEMORY CELLS ON AND OFF REALLY FAST, YOU CAN USE ELECTRICAL INTERFERENCE TO FLIP NEARBY BITS AND— DO WE JUST SUCK AT... COMPUTERS? YUP. ESPECIALLY SHARED ONES.

SO YOU'RE SAYING THE CLOUD IS FULL OF PHANTOM TROLLEYS ARMED WITH HAMMERS... YES. THAT IS EXACTLY RIGHT.

OKAY. I'LL, UH... INSTALL UPDATES? GOOD IDEA.
Edward Snowden
@Snowden

You may have heard about @Intel's horrific #Meltdown bug. But have you watched it in action? When your computer asks you to apply updates this month, don't click "not now." (via spectreattack.com & @misc0110)


152
6.547
6.512
Plan (how it worked out)

- Page Dedup.
- Page Dedup. in JS
- P+P
- P+P in JS
- F+R on Memory
- F+R in JS
- F+R on ARM
- CTA

Daniel Gruss — Graz University of Technology
Plan (how it worked out)

Page Dedup.

P+P → P+P in JS

F+R → F+R in JS

CTA

F+R on Memory

F+R on ARM

Daniel Gruss — Graz University of Technology
Plan (how it worked out)

- Page Dedup.
- Page Dedup. in JS
- P+P
- P+P in JS
- F+R
- F+R on Memory
- F+R on JS
- CTA
- F+R on ARM

Daniel Gruss — Graz University of Technology
Plan (how it worked out)

- Page Dedup.
- Page Dedup. in JS
- P+P
- P+P in JS
- F+R
- F+R on Memory
- Rowhammer.js
- CTA
- F+R on ARM
Plan (how it worked out)

Page Dedup. → Page Dedup. in JS

P+P → P+P in JS → F+R on Memory

F+R → Rowhammer.js

CTA → ARMageddon
Plan (how it worked out)

Page Dedup. → Page Dedup. in JS

P+P → P+P in JS → DRAMA

F+R → Rowhammer.js → ARMageddon

CTA
Influence of the Cache on the Access Latency

![Graph showing the number of cache hits versus access time in CPU cycles. The x-axis represents access time in CPU cycles, ranging from 50 to 400. The y-axis represents the number of accesses on a logarithmic scale, ranging from $10^1$ to $10^7$. The graph shows a distribution of cache hits with a peak around 100 CPU cycles.]
Influence of the Cache on the Access Latency

The graph shows the number of accesses on the y-axis and the access time in CPU cycles on the x-axis. The data is represented in a log-log scale.

- **Cache Hits** (blue bars)
- **Cache Misses** (red bars)

The access time ranges from 50 to 400 CPU cycles, and the number of accesses ranges from $10^1$ to $10^7$. The graph indicates that cache misses occur more frequently than cache hits, especially at lower access times.

Daniel Gruss — Graz University of Technology
Flush+Reload

Attacker Address Space

Cache

Victim Address Space
Flush+Reload

Attacker Address Space

Cache

Victim Address Space

cached

cached
Flush+Reload

Attacker Address Space

Cache

(slow) access

Victim Address Space
Flush + Reload

Attacker Address Space

Cache

(fast) access

Victim Address Space
Relation of the papers

- minimization of requirements
- novel side channels
- automation of attacks
Relation of the papers

- minimization of requirements
- novel side channels
- automation of attacks
- CTA
Relation of the papers

- minimization of requirements
- Dedup.js
- novel side channels
- CTA
- automation of attacks
Relation of the papers

minimization of requirements

Dedup.js

RH.js

F+F

CTA

novel side channels

automation of attacks

Daniel Gruss — Graz University of Technology
Cache Template Attacks
1. how to locate secret-dependent accesses?
   - binaries are large, numerous, closed-source, self-compiled
1. how to locate secret-dependent accesses?
   • binaries are large, numerous, closed-source, self-compiled
2. how to find all exploitable addresses?
1. how to locate secret-dependent accesses?
   - binaries are large, numerous, closed-source, self-compiled
2. how to find all exploitable addresses?

→ automate cache attacks
% sleep 2; ./spy 300 7f05140a4008-7f051417b000 r-xp 8x20000 08:02 268050/usr/lib/x86_64-linux-gnu/gedit/libgedit.so

shark% ./spy
2018: Meltdown

Simple Template over an Array

Secret value can be read from the kernel
All data in the system can be read
• Simple Template over an Array
• Simple Template over an Array

![Graph showing Zugriffszeit (CPU Zyklen) vs. Seite]
• Simple Template over an Array

![Graph showing Zugriffszeit vs. Seiten. The graph indicates a pattern of CPU cycles that are accessed.]
2018: Meltdown

- Simple Template over an Array

- Secret value can be read from the kernel
• Simple Template over an Array

• Secret value can be read from the kernel
• All data in the system can be read
Page Deduplication Attacks in JavaScript
Virtual Address Space

Physical Address Space
Page Deduplication Attack

JavaScript

Virtual Address Space

Physical Address Space
Page Deduplication Attack

Victim Virtual Address Space

JavaScript

Physical Address Space
Page Deduplication Attack

JavaScript

Virtual Address Space

Victim

Physical Address Space
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

JavaScript

Victim
Page Deduplication Attack

Victim's Virtual Address Space

Attacker generates a page suspected in victim process

Physical Address Space
Page Deduplication Attack

Victim

Virtual Address Space

JavaScript

Victim

Physical Address Space

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

JavaScript  

Virtual Address Space  

Attacker waits for deduplication  

Victim  

Physical Address Space
Victim Virtual Address Space

Attacker waits for deduplication

```
t = time();
p[0] = p[0];
Δ = time() - t;
```
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Victim

JavaScript

measure $\Delta$

Time

0

$\Delta$ in $\mu$s

4
Page Deduplication Attack

Victim Virtual Address Space

Physical Address Space

measure $\Delta$

$\Delta$ in $\mu$s

Time

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

JavaScript

measure $\Delta$

$\Delta$ in $\mu$s

Time

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

JavaScript

Physical Address Space

measure $\Delta$

Time

$\Delta$ in $\mu$s

0

4

Victim

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Virtual Address Space

JavaScript

Victim

Physical Address Space

measure $\Delta$

Time

%2$s' in %2$us

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim Virtual Address Space

Physical Address Space

JavaScript

measure $\Delta$

Time

0

4

$\Delta$ in $\mu$s

Victim
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

JavaScript

Time

measure $\Delta$

$\Delta$ in $\mu$s

$0$

$4$

$\text{Victim}$

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

- Victim
- Virtual Address Space
- JavaScript
- Physical Address Space

\[ \Delta \text{ in } \mu s \]

Time

\[ \text{measure } \Delta \]

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim Virtual Address Space

Victim

JavaScript

Physical Address Space

measure $\Delta$

Time

$\Delta$ in $\mu$s

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

JavaScript

Time

0 4

µs

measure ∆

physical address space

Victim

Physical Address Space

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim Virtual Address Space

JavaScript

Physical Address Space

measure $\Delta$

Time


Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

JavaScript

Victim

measure $\Delta$

Time

$\Delta$ in $\mu$s

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

JavaScript

Time

measure $\Delta$

$\Delta$ in $\mu$s

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Physical Address Space

Virtual Address Space

JavaScript

measure $\Delta$

Time

$\Delta$ in $\mu$s

$0$

$4$

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

Victim

JavaScript

measure $\Delta$

$\Delta$ in $\mu s$

Time

$0$

$4$

$\Delta$ in $\mu s$

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

JavaScript

Time

measure $\Delta$

$\Delta$ in $\mu$s

Victim

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Victim

JavaScript

measure $\Delta$

Time

$\Delta$ in $\mu$s

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim Virtual Address Space

Physical Address Space

Victim

JavaScript

measure $\Delta$

$\Delta$ in $\mu s$

Time

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

Time

measure $\Delta$

$\Delta$ in $\mu$s

0

4

$\neq$

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

- **Victim**
- **Virtual Address Space**
- **Physical Address Space**
- **JavaScript**

Measure $\Delta$ in $\mu$s

Time

$\Delta \neq 13$
Page Deduplication Attack

Victim Virtual Address Space

Physical Address Space

JavaScript

measure $\Delta$

$\Delta$ in $\mu$s

Time

$0 \leq \Delta < 4$

$\neq$

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

\[
\Delta \text{ in } \mu s
\]

\[\Delta \neq 13\]

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Virtual Address Space

Physical Address Space

Victim

JavaScript

measure $\Delta$

$\Delta$ in $\mu$s

Time

$0 \neq 13$
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

JavaScript

Victim

\[ \Delta \text{ in } \mu s \]

\[ \Delta \text{ in } \mu s \]

Time

\[ 0 \]

\[ 4 \]

\[ \not= \]

measure \( \Delta \)
Page Deduplication Attack

Virtual Address Space

Physical Address Space

JavaScript

Victim

$\Delta$ in µs

Time

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

Virtual Address Space

Physical Address Space

JavaScript

\[ \Delta \text{ in } \mu s \]

Time

0
4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim Virtual Address Space

Time

write and measure $\Delta$

$\Delta$ in $\mu$s

Victim

JavaScript

Physical Address Space

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Victim

JavaScript

Virtual Address Space

Physical Address Space

Victim

copy

write and measure $\Delta$

$\Delta$ in $\mu$s

0

4

Time

0

4

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Virtual Address Space

JavaScript

Victim

Physical Address Space

\[ \Delta \text{ in } \mu s \]

Time

\[ 0 \]

\[ 4 \]

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Attacker learns that another process had an identical page

JavaScript

Virtual Address Space

Physical Address Space

Victim

Δ in µs

0 4

Time

0 4
Page Deduplication Attack

Attacker learns that another process had an identical page.
Page Deduplication Attack

Attacker learns that another process had an identical page
Page Deduplication Attack

Victim Virtual Address Space

Attacker learns that another process had an identical page

Victim

Physical Address Space

JavaScript

\( \Delta \text{ in } \mu s \)

Time

0

4
Page Deduplication Attack

JavaScript  

Virtual Address Space

Victim

Attacker learns that another process had an identical page

Physical Address Space

Daniel Gruss — Graz University of Technology
Attacker learns that another process had an identical page
Attacker learns that another process had an identical page.
Attacker learns that another process had an identical page.
Attacker learns that another process had an identical page

Delta in microseconds over time:

Virtual Address Space:
- JavaScript
- Victim

Physical Address Space:
- Victim

Daniel Gruss — Graz University of Technology
Page Deduplication Attack

Attacker learns that another process had an identical page.
Page Deduplication Attack

Attacker learns that another process had an identical page.

Virtual Address Space

Physical Address Space
Page Deduplication Attack

Attacker learns that another process had an identical page

JavaScript

Virtual Address Space

Victim

Physical Address Space

Daniel Gruss — Graz University of Technology
Attacker learns that another process had an identical page
Page Deduplication Attack

Attacker learns that another process had an identical page.
Page Deduplication Attack

Attacker learns that another process had an identical page.
Attacker learns that another process had an identical page
Detect Image (JavaScript, Cross-VM, KVM)

Image not loaded

Image loaded

Nanoseconds

Page

Daniel Gruss — Graz University of Technology
Rowhammer.js
Rowhammer

• DRAM bug that causes bit flips
• used in security exploits
• only non-cached accesses reach DRAM
• very similar to Flush+Reload
Rowhammer

• DRAM bug that causes bit flips
• used in security exploits
• only non-cached accesses reach DRAM
• very similar to Flush+Reload
Rowhammer

• DRAM bug that causes bit flips
• used in security exploits
• only non-cached accesses reach DRAM

very similar to Flush+Reload

Daniel Gruss — Graz University of Technology
• DRAM bug that causes bit flips
• DRAM bug that causes bit flips
• used in security exploits
• DRAM bug that causes bit flips
• used in security exploits
• DRAM bug that causes bit flips
• used in security exploits
• only non-cached accesses reach DRAM
• DRAM bug that causes bit flips
• used in security exploits
• only non-cached accesses reach DRAM
  • very similar to Flush+Reload
Rowhammer (with clflush)
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank
Rowhammer (with clflush)

DRAM bank

cache set 1

cache set 2

clflush

clflush
Rowhammer (with clflush)
Rowhammer (with clflush)

DRAM bank

cache set 1

reload

reload

reload

cache set 2
Rowhammer (with clflush)

DRAM bank

cache set 1

cache set 2
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

reload

reload

Daniel Gruss — Graz University of Technology
Rowhammer (with clflush)

- DRAM bank
- cache set 1
- cache set 2

Daniel Gruss — Graz University of Technology
Rowhammer (with clflush)

- cache set 1
- cache set 2
- DRAM bank

reload
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

clflush

Daniel Gruss — Graz University of Technology
Rowhammer (with clflush)
Rowhammer (with `clflush`)
Rowhammer (with clflush)

Cache set 1

Cache set 2

DRAM bank

Reload

Reload

Bit flip!
Rowhammer without `clflush`

- Cache set 1
- Cache set 2
- DRAM bank
Rowhammer without clflush

cache set 1

DRAM bank

cache set 2
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

Daniel Gruss — Graz University of Technology
Rowhammer without clflush

cache set 1

load

DRAM bank

cache set 2
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank
Rowhammer without clflush

Copyright © 2016. All rights reserved.
Rowhammer without clflush

Cache set 1

Cache set 2

DRAM bank
Rowhammer without clflush
Rowhammer without clflush

repeat!

cache set 1

cache set 2

DRAM bank

Daniel Gruss — Graz University of Technology
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

wait for it...
Rowhammer without `clflush`
Rowhammer.js: Challenges

1. How to get accurate timing (in JS)?
   - Easy

2. How to get physical addresses (in JS)?
   - Easy

3. Which physical addresses to access?
   - Already solved

4. In which order to access them?
   - Our contribution

Daniel Gruss — Graz University of Technology
1. how to get accurate timing (in JS)?
1. how to get accurate timing (in JS)? → easy
1. how to get accurate timing (in JS)? → easy
2. how to get physical addresses (in JS)?
1. how to get accurate timing (in JS)? → easy
2. how to get physical addresses (in JS)? → easy
1. how to get accurate timing (in JS)? → easy
2. how to get physical addresses (in JS)? → easy
3. which physical addresses to access?
1. how to get accurate timing (in JS)? → easy
2. how to get physical addresses (in JS)? → easy
3. which physical addresses to access? → already solved
1. how to get accurate timing (in JS)? → easy
2. how to get physical addresses (in JS)? → easy
3. which physical addresses to access? → already solved
4. in which order to access them?
Rowhammer.js: Challenges

1. how to get accurate timing (in JS)? → easy
2. how to get physical addresses (in JS)? → easy
3. which physical addresses to access? → already solved
4. in which order to access them? → our contribution
Replacement policy on older CPUs

“LRU eviction” memory accesses

cache set
"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
“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
“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
“LRU eviction” memory accesses

- undocumented replacement policies
“LRU eviction” memory accesses

- undocumented replacement policies
“LRU eviction” memory accesses

- undocumented replacement policies
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- undocumented replacement policies
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- undocumented replacement policies
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- undocumented replacement policies
“LRU eviction” memory accesses

- undocumented replacement policies
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- undocumented replacement policies
“LRU eviction” memory accesses

- undocumented replacement policies
“LRU eviction” memory accesses

- undocumented replacement policies
- only 75% success rate on Haswell
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- undocumented replacement policies
- only 75% success rate on Haswell
- more accesses → higher success rate, but too slow

Daniel Gruss — Graz University of Technology
Cache eviction strategy: Notation (1)

Write eviction strategies as: $\mathcal{P} - C - D - L - S$

```
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: $\mathcal{P}$-$C$-$D$-$L$-$S$

$S$: 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: \( \mathcal{P}-C-D-L-S \)

- **S**: total number of different addresses (= set size)
- **D**: different addresses per inner access loop

```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];
```
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

```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];
```
Write eviction strategies as: \( \mathcal{P} - C - D - L - S \)

- \( S \): total number of different addresses (= set size)
- \( D \): different addresses per inner access loop
- \( C \): number of repetitions of the inner access loop
- \( L \): step size of the inner access loop

```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 strategy: Notation (2)

for (s = 0; s <= S - D; s += L)
    for (c = 1; c <= C; c += 1)
        for (d = 1; d <= D; d += 1)
            *a[s+d];
Cache eviction strategy: Notation (2)

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

- \(\mathcal{P} - 2 - 2 - 1 - 4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 4, 3, 4\)

Daniel Gruss — Graz University of Technology
Cache eviction strategy: Notation (2)

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

- $P-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 3, 4, 3, 4$

- $P-1-1-1-4 \rightarrow 1, 2, 3, 4 \rightarrow$ LRU eviction with set size 4
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></td>
<td></td>
</tr>
<tr>
<td>P-1-1-1-20</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
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>✗</td>
</tr>
<tr>
<td>( P_{-1-1-1-20} )</td>
<td>20</td>
<td>99.82%</td>
<td>✓</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
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>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
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 x</td>
</tr>
<tr>
<td>$P$-2-1-1-17</td>
<td>34</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
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>☑</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
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>$P\text{-}1\text{-}1\text{-}1\text{-}17$</td>
<td>17</td>
<td>74.46%</td>
<td>307 ns ✓</td>
</tr>
<tr>
<td>$P\text{-}1\text{-}1\text{-}1\text{-}20$</td>
<td>20</td>
<td>99.82%</td>
<td>934 ns X</td>
</tr>
<tr>
<td>$P\text{-}2\text{-}1\text{-}1\text{-}17$</td>
<td>34</td>
<td>99.86%</td>
<td>191 ns ✓</td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache.
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 X</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>

Executed in a loop, on a Haswell with a 16-way last-level cache.
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% (\times)</td>
<td>307 ns</td>
</tr>
<tr>
<td>(P)-1-1-1-20</td>
<td>20</td>
<td>99.82% (\checkmark)</td>
<td>934 ns (\times)</td>
</tr>
<tr>
<td>(P)-2-1-1-17</td>
<td>34</td>
<td>99.86% (\checkmark)</td>
<td>191 ns (\checkmark)</td>
</tr>
<tr>
<td>(P)-2-2-1-17</td>
<td>64</td>
<td>99.98% (\checkmark)</td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
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>

Executed in a loop, on a Haswell with a 16-way last-level cache
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>

→ more accesses, smaller execution time?

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies (illustration)

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

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

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

$P\text{-}2\text{-}1\text{-}1\text{-}34$ (34 accesses, 191\,ns)

Time in ns
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

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

\(P-2-1-1-34\) (34 accesses, 191\,ns)
Cache eviction strategies (illustration)

Example 1:

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

Example 2:

\( P-2-1-1-34 \) (34 accesses, 191\,ns)
Cache eviction strategies (illustration)

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

\[ P-2-1-1-34 \] (34 accesses, 191\,ns)
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

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

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

$P$-2-1-1-34 (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

\[\Phi-2-1-1-34\) (34 accesses, 191\text{ns})

Time in \text{ns}
Cache eviction strategies (illustration)

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

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)
Cache eviction strategies (illustration)

\(\mathcal{P}-1-1-1-17\) (17 accesses, 307ns)

\(\mathcal{P}-2-1-1-34\) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

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

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

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

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

Time in ns
Cache eviction strategies (illustration)

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

$P\text{-}2\text{-}1\text{-}1\text{-}34$ (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Miss (intended)} & \text{Miss (intended)} & \text{Miss} & \text{Miss} \\
\hline
\end{array}
\]

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

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Miss (intended)} & \text{Miss (intended)} & \text{Miss} & \text{Miss} \\
\hline
\end{array}
\]

Time in ns
Cache eviction strategies (illustration)

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

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

\( \mathcal{P}-1-1-17 \) (17 accesses, 307\,ns)

\( \mathcal{P}-2-1-34 \) (34 accesses, 191\,ns)

Time in ns
Cache eviction strategies (illustration)

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

\[ P-2-1-1-34 \] (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

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

\[
\begin{array}{cccccccc}
\text{Miss (intended)} & \text{Miss (intended)} & H & \text{Miss} & \text{Miss} & \text{Miss} & H & \text{Miss}
\end{array}
\]

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

\[
\begin{array}{cccccccc}
\text{Miss (intended)} & \text{Miss (intended)} & \text{H} & \text{H} & \text{H} & \text{H} & \text{Miss} & \text{H}
\end{array}
\]
Cache eviction strategies (illustration)

$P^{-1-1-1-17}$ (17 accesses, 307ns)

$P^{-2-1-1-34}$ (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Miss (intended)} & \text{Miss (intended)} & \text{H} & \text{Miss} & \text{Miss} & \text{H} & \text{Miss}
\end{array}
\]

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Miss (intended)} & \text{Miss (intended)} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{Miss} & \text{H} & \text{H} & \text{H}
\end{array}
\]

Time in ns
Cache eviction strategies (illustration)

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

P-2-1-1-34 (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

$P$-2-1-1-34 (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

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

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

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

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

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

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

\[ \begin{array}{ccccccccc}
\text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} \\
\text{(intended)} & \text{(intended)} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} \\
\end{array} \]

\( \Psi-2-1-1-34 \) (34 accesses, 191ns)

\[ \begin{array}{ccccccccc}
\text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} & \text{Miss} \\
\text{(intended)} & \text{(intended)} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} & \text{H} \\
\end{array} \]
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

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

Time in \text{ns}
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

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

\( \Phi-2-1-34 \) (34 accesses, 191ns)
Cache eviction strategies (illustration)

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

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

Time in ns
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-1-34 \) (34 accesses, 191ns)
Cache eviction strategies (illustration)

\( \mathcal{P}-1-1-17 \) (17 accesses, 307ns)

\( \mathcal{P}-2-1-34 \) (34 accesses, 191ns)

Time in ns
Cache eviction strategies (illustration)

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

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

Time in ns
Rowhammer Requirements

- accurate timing (in JS)
- physical addresses (in JS)
- address selection
- ? order of accesses
Rowhammer Requirements

☑ accurate timing (in JS)
☑ physical addresses (in JS)
☑ address selection
☑ order of accesses
Bitflips in JavaScript

![JavaScript Bitflip Example](image.png)
Evaluation on Haswell

Figure 1: Number of bit flips within 15 minutes.
Flush+Flush
Flush+Flush: Motivation

- cache attacks → many cache misses
- detect via performance counters
Flush+Flush: Motivation

- cache attacks $\rightarrow$ many cache misses
- detect via performance counters
$\rightarrow$ good idea, but is it good enough?

Daniel Gruss — Graz University of Technology
Flush+Flush: Motivation

- cache attacks → many cache misses
- detect via performance counters
→ good idea, but is it good enough?
  - causing a cache flush ≠ causing a cache miss
Flush+Flush

Attacker address space

cached

cached

Cache

Victim address space

30

Daniel Gruss — Graz University of Technology
Flush+Flush

Attacker address space

Cache

Victim address space

loads data
Flush+Flush

Flushes (slow)
• attacker causes no direct cache misses
  → fast
  → stealthy
Flush+Flush: Conclusion

- attacker causes no direct cache misses
  - → fast
  - → stealthy

- same side channel targets as Flush+Reload
Flush+Flush: Conclusion

- attacker causes no direct cache misses
  → fast
  → stealthy
- same side channel targets as Flush+Reload
- 496 KB/s covert channel
ARMageddon
Cache Attacks on Mobile Devices?

- Powerful cache attacks on Intel x86 in the last 10 years
- Nothing like Flush+Reload or Prime+Probe on mobile devices

→ Why not?
Cache Attacks on Mobile Devices?

- powerful cache attacks on Intel x86 in the last 10 years
• powerful cache attacks on Intel x86 in the last 10 years
• nothing like Flush+Reload or Prime+Probe on mobile devices
Cache Attacks on Mobile Devices?

- powerful cache attacks on Intel x86 in the last 10 years
- nothing like Flush+Reload or Prime+Probe on mobile devices
→ why not?
1. no flush instruction
1. no flush instruction → Evict+Reload
1. no flush instruction → Evict+Reload
2. pseudo-random replacement
1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ eviction strategies from Rowhammer.js
1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ eviction strategies from Rowhammer.js
3. cycle counters require root
1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ eviction strategies from Rowhammer.js
3. cycle counters require root $\rightarrow$ new timing methods
ARMageddon in a nutshell

1. no flush instruction → Evict+Reload
2. pseudo-random replacement → eviction strategies from Rowhammer.js
3. cycle counters require root → new timing methods
4. last-level caches not inclusive
1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ eviction strategies from Rowhammer.js
3. cycle counters require root $\rightarrow$ new timing methods
4. last-level caches not inclusive $\rightarrow$ let L1 spill to LLC
ARMageddon in a nutshell

1. no flush instruction → Evict+Reload
2. pseudo-random replacement → eviction strategies from Rowhammer.js
3. cycle counters require root → new timing methods
4. last-level caches not inclusive → let L1 spill to LLC
5. multiple CPUs
1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ eviction strategies from Rowhammer.js
3. cycle counters require root $\rightarrow$ new timing methods
4. last-level caches not inclusive $\rightarrow$ let L1 spill to LLC
5. multiple CPUs $\rightarrow$ remote fetches + flushes
1. no flush instruction → Evict+Reload
2. pseudo-random replacement → eviction strategies from Rowhammer.js
3. cycle counters require root → new timing methods
4. last-level caches not inclusive → let L1 spill to LLC
5. multiple CPUs → remote fetches + flushes
Idea: Would this also work on inaccessible kernel memory?
Prefetch: Locate Kernel Driver (defeat KASLR)

![Graph showing page offset in kernel driver region vs. average execution time](image-url)
Prefetch: Kernel Memory Layout

Virtual address space

Physical memory

User

Kernel

0

max. phys.

0

2^{47}

-2^{47}

-1

Daniel Gruss — Graz University of Technology
Prefetch, Spectre, Meltdown

• Same underlying problem

• Prefetch does not fetch the value into a register

• Meltdown does

• Same countermeasure: KAISER

• Some aspects also related to Spectre
• Same underlying problem
• Same underlying problem
• Prefetch does not fetch the value into a register
• Same underlying problem
• Prefetch does not fetch the value into a register
• Meltdown does
Prefetch, Spectre, Meltdown

- Same underlying problem
- Prefetch does not fetch the value into a register
- Meltdown does
- Same countermeasure: KAISER
• Same underlying problem
• Prefetch does not fetch the value into a register
• Meltdown does
• Same countermeasure: KAISER
• Some aspects also related to Spectre
KAISER (Stronger Kernel Isolation)

Without KAISER:

Shared address space

User memory

Kernel memory

context switch

With KAISER:

User address space

User memory

Not mapped

context switch

SMAP + SMEP

Kernel memory

context switch

Interrupt dispatcher

Daniel Gruss — Graz University of Technology
Conclusions

1. Microarchitectural attacks can be widely automated.
2. Unknown and novel side channels are likely to exist (e.g., Meltdown and Spectre).
3. Minimal requirements enable attacks through websites.
4. Constructing countermeasures is difficult and requires a solid understanding of attacks.
Conclusions

1. microarchitectural attacks can be *widely automated*
Conclusions

1. microarchitectural attacks can be **widely automated**
2. unknown and **novel side channels** are likely to exist (e.g., Meltdown and Spectre)
1. microarchitectural attacks can be **widely automated**
2. unknown and **novel side channels** are likely to exist (e.g., Meltdown and Spectre)
3. **minimal requirements** enable attacks through websites
1. microarchitectural attacks can be widely automated
2. unknown and novel side channels are likely to exist (e.g., Meltdown and Spectre)
3. minimal requirements enable attacks through websites
4. constructing countermeasures is difficult and requires solid understanding of attacks
Software-based Microarchitectural Attacks

Daniel Gruss
May 30, 2018

Graz University of Technology


