Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript

Daniel Gruss, Clémentine Maurice, and Stefan Mangard
Graz University of Technology

July 8, 2016
Overview

- Rowhammer: bit flip at a random location in DRAM
- exploitable \(\rightarrow\) gain root privileges

We are the first to

- evaluate performance of cache eviction
- perform Rowhammer attacks without `clflush` on many platforms
- perform fault attacks from a website using JavaScript
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

![DRAM bank and row buffer diagram](image-url)
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

[Diagram of DRAM bank with active and copied rows highlighted]
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice
Rowhammer

“It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after” – Motherboard Vice

[Diagram of a DRAM bank with row buffer and bit flips in row 2 highlighted]
Impact of the CPU cache

- only non-cached accesses reach DRAM
- original attacks use `clflush` instruction
  → flush line from cache
  → next access will be served from DRAM
Rowhammer (with clflush)
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- DRAM bank

Daniel Gruss, Graz University of Technology
July 8, 2016
Rowhammer (with \texttt{clflush})

\begin{itemize}
  \item cache set 1
  \item cache set 2
  \item DRAM bank
\end{itemize}
Rowhammer (with clflush)
Rowhammer (with clflush)

Cache set 1

Cache set 2

DRAM bank

reload
Rowhammer (with clflush)
Rowhammer (with clflush)

Cache set 1

Cache set 2

DRAM bank

clflush

clflush
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- DRAM bank

reload reload
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank
Rowhammer (with clflush)
Rowhammer (with clflush)
Rowhammer (with clflush)
Rowhammer (with \texttt{clflush})

- cache set 1
- cache set 2
- DRAM bank

wait for it...
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

bit flip!

reload

reload
Flush, reload, flush, reload. . .

- the core of Rowhammer is essentially a Flush+Reload loop
- as much an attack on DRAM as on cache
Rowhammer without `clflush`?

- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript
Rowhammer without `clflush`?

- **idea**: avoid `clflush` to be independent of specific instructions
  - → no `clflush` in JavaScript

- **our approach**: use regular memory accesses for eviction
  - → techniques from cache attacks!
Rowhammer without `clflush`?

- **idea**: avoid `clflush` to be independent of specific instructions
  - → no `clflush` in JavaScript

- **our approach**: use regular memory accesses for eviction
  - → techniques from cache attacks!
  - → Rowhammer, Prime+Probe style!
Rowhammer without `clflush`

cache set 1

cache set 2

DRAM bank
Rowhammer without clflush

cache set 1

load

DRAM bank

cache set 2
Rowhammer without `clflush`

- **Cache Set 1**: Parts loaded from DRAM bank.
- **Cache Set 2**: Parts loaded from DRAM bank.

`Daniel Gruss, Graz University of Technology
July 8, 2016`
Rowhammer without `clflush`
Rowhammer without `clflush`

cache set 1

cache set 2

DRAM bank

load
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank
Rowhammer without clflush

 cache set 1

 cache set 2

 DRAM bank

 load

 load
Rowhammer without `clflush`
Rowhammer without `clflush`
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

reload

reload
Rowhammer without clflush

cache set 1

cache set 2

repeat!

DRAM bank
Rowhammer without clflush

wait for it...
Rowhammer without `clflush`

Cache set 1

Cache set 2

DRAM bank

bit flip!
Requirements for Rowhammer

1. **uncached** memory accesses: need to reach DRAM
2. **fast** memory accesses: race against the next row refresh
Requirements for Rowhammer

1. *uncached* memory accesses: need to reach DRAM
2. *fast* memory accesses: race against the next row refresh

→ optimize the eviction rate and the timing
Rowhammer.js: the challenges

1. how to get accurate timing in JS?
2. how to get physical addresses in JS?
3. which physical addresses to access?
4. in which order to access them?
Rowhammer.js: the challenges

1. how to get accurate timing in JS? → easy
2. how to get physical addresses in JS?
3. which physical addresses to access?
4. in which order to access them?
Rowhammer.js: the 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?
4. in which order to access them?
Rowhammer.js: the 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?
Rowhammer.js: the 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
Challenge #1: accurate timing in JavaScript?

- native code: `rdtsc`
- JavaScript: `window.performance.now()`
Challenge #1: accurate timing in JavaScript?

- native code: rdtsc
- JavaScript: `window.performance.now()`
- recent patch: time rounded to 5 microseconds
- still works: we measure millions of accesses
Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
- last 21 bits (2MB) of physical address
- = last 21 bits (2MB) of virtual address
Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
  - last 21 bits (2MB) of physical address
  - = last 21 bits (2MB) of virtual address
  - = last 21 bits (2MB) of JS array indices Gruss et al. 2015
Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
  - last 21 bits (2MB) of physical address
  - = last 21 bits (2MB) of virtual address
  - = last 21 bits (2MB) of JS array indices Gruss et al. 2015

- several DRAM rows per 2MB page
- several congruent addresses per 2MB page
Challenge #3: physical addresses and DRAM

- fixed map: physical addresses $\rightarrow$ DRAM cells
- undocumented for Intel CPUs
- reverse-engineered for Sandy Bridge Seaborn 2015
- and by us for Sandy, Ivy, Haswell, Skylake, ... Pessl et al. 2016 (to appear)
Challenge #3: physical addresses and cache sets

- fixed map: physical addresses $\rightarrow$ cache sets
- undocumented for Intel CPUs but reverse-engineered Maurice et al. 2015
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

cache set

[Diagram of cache set with one entry highlighted]
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- LRU replacement policy: oldest entry first
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- LRU replacement policy: oldest entry first
- Timestamps for every cache line
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

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

Cache set:

2 5 8 9 7 6 3 4
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp
Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

- 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 on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
- only 75% success rate on Haswell
Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement on recent CPUs
- only 75% success rate on Haswell
- more accesses $\rightarrow$ higher success rate, but too slow
Cache eviction strategies: The beginning

→ fast and effective on Haswell: eviction rate \(>99.97\%\)
Cache eviction strategy: New representation

- represent accesses as a sequence of numbers: 
  1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, ...

- can be a long sequence

- all congruent addresses are indistinguishable w.r.t eviction strategy
Cache eviction strategy: New representation

- represent accesses as a sequence of numbers: 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, ...
- can be a long sequence
- all congruent addresses are indistinguishable w.r.t eviction strategy
  → adding more unique addresses can increase eviction rate
  → multiple accesses to one address can increase the eviction rate
- indistinguishable → balanced number of accesses
Cache eviction strategy: Notation (1)

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

\[
\begin{align*}
\text{for (s = 0; s <= S - D; s += L)} \\
\text{for (c = 0; c <= C; c += 1)} \\
\text{for (d = 0; d <= D; d += 1)} \\
& *a[s+d] ; 
\end{align*}
\]
Cache eviction strategy: Notation (1)

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];
```
Cache eviction strategy: Notation (1)

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

$S$: total number of different addresses (= set size)

$D$: different addresses per 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 strategy: Notation (1)

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];
```
Cache eviction strategy: Notation (1)

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

$C$: number of repetitions 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 strategy: Notation (2)

\[
\begin{align*}
\text{for} & \ (s = 0; \ s <= \ S - D; \ s += \ L) \\
& \quad \text{for} \ (c = 1; \ c <= \ C; \ c += 1) \\
& \quad \quad \text{for} \ (d = 1; \ d <= \ D; \ d += 1) \\
& \quad \quad \quad *a[s+d];
\end{align*}
\]
Cache eviction strategy: Notation (2)

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

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

- $P - 2 - 2 - 1 - 4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4$  
  \( S = 4 \)
Cache eviction strategy: Notation (2)

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

\[ P-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4 \quad S = 4 \]
Cache eviction strategy: Notation (2)

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

- \(P - 2 - 2 - 1 - 4\) → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

\(S = 4\)

\(D = 2\)
Cache eviction strategy: Notation (2)

```c
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];
```

- $P_{-2-2-1-4} \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4$ (with set size $S = 4$)
  - $D = 2$
  - $C = 2$
Cache eviction strategy: Notation (2)

\[
\text{for } (s = 0; s \leq S - D; s += L) \\
\text{for } (c = 1; c \leq C; c += 1) \\
\text{for } (d = 1; d \leq D; d += 1) \\
\text{ } a[s+d];
\]

\[P \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, S = 4\]

\[L = 1, D = 2, C = 2\]
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];

- $P-2-2-1-4 \rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4 \rightarrow S = 4$
- $L = 1$, $D = 2$, $C = 2$

- $P-1-1-1-4 \rightarrow 1, 2, 3, 4 \rightarrow$ LRU eviction with set size 4
Cache eviction strategies: Evaluation

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
Cache eviction strategies: Evaluation

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
Cache eviction strategies: Evaluation

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
Cache eviction strategies: Evaluation

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></td>
<td></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

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
Cache eviction strategies: Evaluation

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>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

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
Cache eviction strategies: Evaluation

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></td>
</tr>
</tbody>
</table>

Executed in a loop, on a Haswell with a 16-way last-level cache
Cache eviction strategies: Evaluation

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
Cache eviction strategies: Evaluation

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

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

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

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

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

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

Miss (intended)  Miss (intended)

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

Miss (intended)  Miss (intended)

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

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

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

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** (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** (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)

![Diagram showing cache eviction strategy for P-1-1-1-17 with 17 accesses and 307ns latency.]

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

![Diagram showing cache eviction strategy for P-2-1-1-17 with 34 accesses and 191ns latency.]

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)

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

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

```
Miss (intended)  Miss (intended)  Hit  Hit  Hit  Miss  Miss
```
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)](time_in_ns)

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

![Miss (intended)](time_in_ns)
Cache eviction strategies: Illustration

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

![Cache eviction strategy P-1-1-1-17 diagram]

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

![Cache eviction strategy P-2-1-1-17 diagram]

**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)

Miss (intended)  Miss (intended)  Miss  Miss  Miss  H  Miss

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

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

Time in ns
Cache eviction strategies: Illustration

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

![Diagram showing cache eviction strategy for P-1-1-1-17]

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

![Diagram showing cache eviction strategy for P-2-1-1-17]

Time in ns
Cache eviction strategies: Illustration

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

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

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

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

Time in ns
Cache eviction strategies: Illustration

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

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

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

<table>
<thead>
<tr>
<th>Time in ns</th>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss</th>
<th>Miss</th>
<th>Hit</th>
<th>Miss</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Time in ns</th>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Hit</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
</tr>
</thead>
</table>

Daniel Gruss, Graz University of Technology
July 8, 2016
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** (17 accesses, 307ns)

Time in ns

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

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

Miss (intended) Miss Miss Miss Miss Miss Miss

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

Miss (intended) Miss Miss Miss Miss Miss Miss

Time in ns
Cache eviction strategies: Illustration

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

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

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

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

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)

```
Miss (intended)  Miss (intended)  Miss  Miss  Miss  Mem  Miss  Miss
```

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

```
Miss (intended)  Miss (intended)  Mem  Miss  Mem  Mem  Mem  Mem
```

Time in ns
Cache eviction strategies: Illustration

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

<table>
<thead>
<tr>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Hit</th>
<th>Miss</th>
<th>Miss</th>
<th>Miss</th>
<th>Hit</th>
<th>Miss</th>
<th>Miss</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Hit</th>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
<th>Miss (intended)</th>
</tr>
</thead>
</table>

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 (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 (17 accesses, 307ns)**

| Time in ns | Miss (intended) | Miss (intended) | Miss | Miss | Miss | Miss | Miss | Miss | Miss | Miss | Miss |

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

| Time in ns | Miss (intended) | Miss (intended) | Miss | Miss | Miss | Miss | 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)
Execution time vs. bit flips

→ low execution time is better.
Eviction rate vs. bit flips

→ high eviction rate is better. Average: 73.96%.
Eviction strategies on Haswell

Table: The fastest 5 eviction strategies with an eviction rate above 99.75% compared to clflush and LRU eviction on Haswell.

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>L</th>
<th>S</th>
<th>Accesses</th>
<th>Hits</th>
<th>Misses</th>
<th>Time (ns)</th>
<th>Eviction</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>60</td>
<td>99.9999%</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>90</td>
<td>34</td>
<td>4</td>
<td>179</td>
<td>99.9624%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>17</td>
<td>64</td>
<td>35</td>
<td>5</td>
<td>180</td>
<td>99.9820%</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>34</td>
<td>47</td>
<td>5</td>
<td>191</td>
<td>99.8595%</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>108</td>
<td>34</td>
<td>5</td>
<td>216</td>
<td>99.9365%</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td>17</td>
<td>96</td>
<td>13</td>
<td>307</td>
<td>74.4593%</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>20</td>
<td>80</td>
<td>41</td>
<td>23</td>
<td>329</td>
<td>99.7800%</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td>187</td>
<td>78</td>
<td>934</td>
<td>99.8200%</td>
</tr>
</tbody>
</table>
Evaluation on Haswell

![Graph showing the number of bit flips within 15 minutes versus refresh interval in µs (BIOS configuration).]

- clflush
- Evict (Native)
- Evict (JavaScript)

Figure: Number of bit flips within 15 minutes.

Daniel Gruss, Graz University of Technology
July 8, 2016
How to exploit?

- OS groups pages / page tables into 2 MB frames
How to exploit?

- OS groups pages / page tables into 2 MB frames
  - Page tables never in a DRAM row between two code/data pages
How to exploit?

- OS groups pages / page tables into 2 MB frames
  → Page tables never in a DRAM row between two code/data pages
    - unless system is almost out of memory
How to exploit?

- OS groups pages / page tables into 2 MB frames
  → Page tables never in a DRAM row between two code/data pages
    - unless system is almost out of memory
  - hard to get there without crashing the browser
How to exploit?

- OS groups pages / page tables into 2 MB frames
  - Page tables never in a DRAM row between two code/data pages
    - unless system is almost out of memory
  - hard to get there without crashing the browser
- new hammering technique: amplified single-sided hammering
Amplified Single-Sided Hammering

cache set 1

cache set 2

DRAM bank
Amplified Single-Sided Hammering

cache set 1

load

load

cache set 2

DRAM bank
Amplified Single-Sided Hammering

cache set 1

cache set 2

load

load

DRAM bank
Amplified Single-Sided Hammering

cache set 1

load

DRAM bank

cache set 2

load
Amplified Single-Sided Hammering

cache set 1

cache set 2

load

DRAM bank
Amplified Single-Sided Hammering

cache set 1

cache set 2

load

load

DRAM bank
Amplified Single-Sided Hammering

Cache Set 1

Cache Set 2

DRAM Bank

Load
Amplified Single-Sided Hammering

cache set 1

load

cache set 2

load

DRAM bank
Amplified Single-Sided Hammering

- cache set 1
- cache set 2
- load
- DRAM bank
Amplified Single-Sided Hammering

cache set 1

cache set 2

DRAM bank

reload
Amplified Single-Sided Hammering

cache set 1

cache set 2

repeat!

DRAM bank
Amplified Single-Sided Hammering

cache set 1

cache set 2

DRAM bank
Amplified Single-Sided Hammering

cache set 1

cache set 2

DRAM bank

bit flip!
Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
- no near-out-of-memory situation
Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
- no near-out-of-memory situation
- try until memory mappings changed
  - bit flip in your own page tables
Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
- no near-out-of-memory situation
- try until memory mappings changed
  - bit flip in your own page tables
- try until your own page tables are mapped
Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
- no near-out-of-memory situation
- try until memory mappings changed
  - bit flip in your own page tables
- try until your own page tables are mapped
  - full access to all physical memory
Reliable exploits based on Rowhammer.js?

- clever attack exploiting memory deduplication and Rowhammer
- reliable exploit on Microsoft Edge
Conclusions

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions
- first remote fault attack, from a browser
Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript

Daniel Gruss, Clémentine Maurice, and Stefan Mangard
Graz University of Technology

July 8, 2016
Bibliography


Maurice, Clémentine et al. (2015). “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: RAID.
