Rowhammer.js: Root privileges for web apps?

Daniel Gruss (@lavados)\textsuperscript{1}, Clémentine Maurice (@BloodyTangerine)\textsuperscript{2}
\textsuperscript{1}IAIK, Graz University of Technology / \textsuperscript{2}Technicolor and Eurecom

December 28, 2015 — 32c3, Hamburg, Germany
Both of us started to work independently on cache attacks.
Bit flips! — The timeline

2014
June

2015
March
April
June
July

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine), December 28, 2015 — 32c3, Hamburg, Germany
Bit flips! — The timeline

2014

June

Kim et al. paper “Flipping bits in memory without accessing them” at ISCA’14
= Rowhammer with clflush, originally viewed as a reliability issue

2015

March

April

June

July
Bit flips! — The timeline

- 2014
  - June
    - Kim et al. paper "Flipping bits in memory without accessing them" at ISCA'14
      = Rowhammer with clflush, originally viewed as a reliability issue

- 2015
  - March
    - Mark Seaborn and Halvar Flake build exploits using Rowhammer with clflush
Bit flips! — The timeline

2014
June
Kim et al. paper “Flipping bits in memory without accessing them” at ISCA’14
= Rowhammer with clflush, originally viewed as a reliability issue

2015
March
Mark Seaborn and Halvar Flake build exploits using Rowhammer with clflush

April
Daniel starts working on Rowhammer without clflush

June

July
Kim et al. paper “Flipping bits in memory without accessing them” at ISCA’14 = Rowhammer with clflush, originally viewed as a reliability issue

Mark Seaborn and Halvar Flake build exploits using Rowhammer with clflush

Daniel starts working on Rowhammer without clflush

Clémentine visits TU Graz to work on it with Daniel
Bit flips! — The timeline

2014
June
Kim et al. paper “Flipping bits in memory without accessing them” at ISCA’14 = Rowhammer with clflush, originally viewed as a reliability issue

2015
March
Mark Seaborn and Halvar Flake build exploits using Rowhammer with clflush
April
Daniel starts working on Rowhammer without clflush
June
Clémentine visits TU Graz to work on it with Daniel
First bit flips without clflush on Ivy Bridge and Haswell a few days later!

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine), December 28, 2015 — 32c3, Hamburg, Germany
Bit flips! — The timeline

2014
June  
Kim et al. paper “Flipping bits in memory without accessing them” at ISCA’14
= Rowhammer with clflush, originally viewed as a reliability issue

2015
March  
Mark Seaborn and Halvar Flake build exploits using Rowhammer with clflush

April  
Daniel starts working on Rowhammer without clflush

June  
Clémentine visits TU Graz to work on it with Daniel

First bit flips without clflush on Ivy Bridge and Haswell a few days later!

July  
Bit flips in JavaScript!

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine),
December 28, 2015 — 32c3, Hamburg, Germany
DRAM organization example
DRAM organization example

channel 0

channel 1
DRAM organization example

back of DIMM: rank 1

front of DIMM: rank 0
DRAM organization example

channel 0
back of DIMM: rank 1
front of DIMM: rank 0
channel 1
chip
DRAM organization example

- bits in cells in rows
- access: activate row, copy to row buffer
- cells leak $\rightarrow$ refresh necessary
- cells leak faster upon proximate accesses
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

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine), December 28, 2015 — 32c3, Hamburg, Germany
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

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine), December 28, 2015 — 32c3, Hamburg, Germany
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

| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

bit flips in row 2!

Row buffer
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
Cache background

- L1 and L2 are private
- last-level cache:
  - divided in slices
  - shared across cores
  - inclusive
Rowhammer (with clflush)

![Diagram showing cache sets and DRAM bank with highlighted rows indicating rowhammer vulnerability]
Rowhammer (with clflush)
Rowhammer (with clflush)


cache set 1

DRAM bank

cache set 2
Rowhammer (with clflush)
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

reload
Rowhammer (with clflush)

- Cache set 1
- Cache set 2
- DRAM bank

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine),
December 28, 2015 — 32c3, Hamburg, Germany
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank
Rowhammer (with clflush)
Rowhammer (with clflush)

Cache set 1

Cache set 2

DRAM bank

Daniel Gruss (@lavados), Clémantine Maurice (@BloodyTangerine),
December 28, 2015 — 32c3, Hamburg, Germany
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank

reload

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine),
December 28, 2015 — 32c3, Hamburg, Germany
Rowhammer (with clflush)

cache set 1

cache set 2

DRAM bank
Rowhammer (with clflush)
Rowhammer (with 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
Wait a second, Flush+Reload?

- “Flush+Reload” – a cache attack
- exactly what we just did, but ...
Wait a second, Flush+Reload?

- “Flush+Reload” – a cache attack
- exactly what we just did, but ...
- measure timing (cache hit vs. miss)
- run on shared libraries to spy on other processes
Wait a second, Flush+Reload?

- “Flush+Reload” – a cache attack
- exactly what we just did, but ...
- measure timing (cache hit vs. miss)
- run on shared libraries to spy on other processes
- automated attacks [Gruss et al., 2015c]
- crypto keys, keylogging, cross VM attacks, ...
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

cache set 1

cache set 2

DRAM bank
Rowhammer without clflush

![Diagram showing DRAM bank and cache sets with load operations]
Rowhammer without `clflush`

```plaintext
cache set 1

cache set 2

DRAM bank
```

Daniel Gruss (@lavados), Clémentine Maurice (@BloodyTangerine),
December 28, 2015 — 32c3, Hamburg, Germany
Rowhammer without clflush
Rowhammer without clflush

cache set 1

load

DRAM bank

load

cache set 2
Rowhammer without clflush

cache set 1

load

DRAM bank

cache set 2

load
Rowhammer without clflush

cache set 1

load

cache set 2

load
Rowhammer without clflush

cache set 1

DRAM bank

load

cache set 2
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

load

load
Rowhammer without clflush

cache set 1

cache set 2

DRAM bank

reload

reload
Rowhammer without clflush

cache set 1

DRAM bank

repeat!

cache set 2
Rowhammer without `clflush`

![Diagram showing cache sets and DRAM bank](image-url)

- **Cache set 1**
- **Cache set 2**
- **DRAM bank**

Wait for it...
Rowhammer without `clflush`

Cache set 1

Cache set 2

DRAM bank

bit flip!
Rowhammer.js: the challenges

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

1. How to get physical addresses in JS?
2. Which physical addresses to access?
3. In which order to access them?
4. How to get accurate timing in JS?
#1.1: Physical addresses and DRAM

- fixed map: physical addresses $\rightarrow$ DRAM cells
- undocumented for Intel
- reverse-engineering by [Seaborn, 2015b] for Sandy Bridge
- and by us [Pessl et al., 2015] for Sandy, Ivy, Haswell, Skylake, ...
#1.2: Physical addresses and JavaScript

- OS optimization: use 2MB pages
  - last 21 bits (2MB) of physical address
  - = last 21 bits (2MB) of virtual address
#1.2: 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., 2015a]
#1.2: 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 [Grus
- several DRAM rows per 2MB page
- several congruent addresses per 2MB page
- use timing for cross-page information
#2.1: Which physical addresses to access?

“LRU eviction”:

- assume that cache uses LRU replacement
- accessing \( n \) addresses from the same cache set to evict an \( n \)-way set
  
  [Percival, 2005, Liu et al., 2015, Oren et al., 2015, Maurice et al., 2015b]
- eviction from last level → from whole hierarchy (it’s inclusive!)
#2.2: Which addresses map to the same set?

- function H that maps slices is undocumented
- reverse-engineered by
  [Maurice et al., 2015a, Inci et al., 2015, Yarom et al., 2015]
#2.2: Which addresses map to the same set?

- function $H$ that maps slices is undocumented
- reverse-engineered by [Maurice et al., 2015a, Inci et al., 2015, Yarom et al., 2015]
- hash function basically an XOR of address bits
#2.2: Which addresses map to the same set?

- function $H$ that maps slices is undocumented
- reverse-engineered by [Maurice et al., 2015a, Inci et al., 2015, Yarom et al., 2015]
- hash function basically an XOR of address bits
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

![Diagram of cache set with one entry highlighted.](https://example.com/diagram.png)
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

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

```
cache set:        2  5  8  1  7  6  3  4
```
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first
- Timestamps for every cache line
- Access updates timestamp
#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

- no LRU replacement
#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

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

“LRU eviction” memory accesses

- no LRU replacement
- only 75% success rate on Haswell
- more accesses $\rightarrow$ higher success rate, but too slow
#3.3: Cache eviction strategy

Figure: Fast and effective on Haswell. Eviction rate >99.97%.
#4: How to get accurate timing in JavaScript?

- **native code:** `rdtsc`
- **JavaScript:** `window.performance.now()`
#4: How to get 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
Evaluation on Haswell

![Graph showing bit flips over refresh interval]

Figure: Number of bit flips within 15 minutes. [Gruss et al., 2015b]
Exploits?

Idea: port root exploit by [Seaborn, 2015a] to JavaScript

- page table spraying
- needs shared memory → not available in JavaScript
Exploits?

Idea: port root exploit by [Seaborn, 2015a] to JavaScript

- page table spraying
- needs shared memory → not available in JavaScript
- zero pages are deduplicated
  - “shared memory” accessible from JavaScript
Physical memory access exploit in native code

1. find exploitable bitflip (address bit)
2. release bitflip page
Physical memory access exploit in native code

1. find exploitable bitflip (address bit)
2. release bitflip page
3. put page table there – page table spraying with shared pages
4. trigger bitflip again
Physical memory access exploit in native code

1. find exploitable bitflip (address bit)
2. release bitflip page
3. put page table there – page table spraying with shared pages
4. trigger bitflip again
5. page table mapped instead of shared page?
6. modify page table to access arbitrary memory
Physical memory access exploit in JavaScript

1. find exploitable bitflip (address + writable bit)
2. release bitflip page
3. put page table there – page table spraying with zero pages
4. trigger bitflip again
5. page table mapped instead of zero page?
6. modify page table to access arbitrary memory
Code execution as root

1. search for `/bin/sh` binary pages, ...
2. modify page: add shellcode
Code execution as root

1. search for `/bin/sh` binary pages, ...
2. modify page: add shellcode
3. wait until root executes shellcode
Countermeasures

- patch hardware: dynamic row refreshing

- patch BIOS: increase refresh rate
Countermeasures

- patch hardware: dynamic row refreshing
  → what about legacy hardware?
- patch BIOS: increase refresh rate
Countermeasures

- patch hardware: dynamic row refreshing
  - what about legacy hardware?
- patch BIOS: increase refresh rate
  - might not be sufficient for all machines [Kim et al., 2014]
  - need a BIOS update...
Countermeasures

- patch hardware: dynamic row refreshing
  - what about legacy hardware?
- patch BIOS: increase refresh rate
  - might not be sufficient for all machines [Kim et al., 2014]
  - need a BIOS update... Who does that?
Countermeasures

- patch hardware: dynamic row refreshing
  → what about legacy hardware?
- patch BIOS: increase refresh rate
  → might not be sufficient for all machines [Kim et al., 2014]
  → need a BIOS update... Who does that?
Countermeasures

- patch hardware: dynamic row refreshing
  → what about legacy hardware?
- patch BIOS: increase refresh rate
  → might not be sufficient for all machines [Kim et al., 2014]
  → need a BIOS update... Who does that?
- patch OS: life is not perfect – that’s ok!
Countermeasures

- patch hardware: dynamic row refreshing
  → what about legacy hardware?
- patch BIOS: increase refresh rate
  → might not be sufficient for all machines [Kim et al., 2014]
  → need a BIOS update... Who does that?
- patch OS: life is not perfect – that’s ok!
  - idea: we don’t care about self-destructive processes
  → same-privilege physical memory pools
Conclusions

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions
- hardware-fault attack induced in JavaScript
  → remote attacks through websites
Not there yet, but ...

ROOT privileges for web apps!
Rowhammer.js: Root privileges for web apps?

Daniel Gruss (@lavados)$^1$, Clémentine Maurice (@BloodyTangerine)$^2$

$^1$IAIK, Graz University of Technology / $^2$Technicolor and Eurecom

December 28, 2015 — 32c3, Hamburg, Germany
Practical memory deduplication attacks in sandboxed javascript.

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

Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches.
In USENIX Security Symposium (USENIX Security’15).

Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud.

Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors.


Last-Level Cache Side-Channel Attacks are Practical.

In IEEE Symposium on Security and Privacy (S&P’15).


Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters.

In International Symposium on Research in Attacks, Intrusions and Defenses (RAID).


C5: Cross-Cores Cache Covert Channel.

In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA’15).
The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications.
In ACM Conference on Computer and Communications Security (CCS’15).

Cache Missing for Fun and Profit.
URL: http://daemonology.net/hyperthreading-considered-harmful/.

Reverse Engineering Intel DRAM Addressing and Exploitation.

Exploiting the DRAM rowhammer bug to gain kernel privileges.

How physical addresses map to rows and banks in DRAM.


Mapping the Intel Last-Level Cache.

The research leading to these results has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR).

Furthermore, this work has been supported by the Austrian Research Promotion Agency (FFG) and the Styrian Business Promotion Agency (SFG) under grant number 836628 (SeCoS).

The joint research has been supported by CRYPTACUS COST action.