Cache attacks: From side channels to fault attacks

Clémentine Maurice – CNRS, Rennes, France

November 16, 2017 – Cryptacus Workshop, Nijmegen, Netherlands

Cache attacks

- side-channel attacks without physical access to the device

Y. Tsunoo, T. Saito, and T. Suzaki. “Cryptanalysis of DES implemented on computers with cache”. In: *CHES’03*. 2003, pp. 62–76.

Cache attacks

- side-channel attacks without physical access to the device
- the attacker only needs unprivileged execution on the victim's machine

---

Y. Tsunoo, T. Saito, and T. Suzaki. "Cryptanalysis of DES implemented on computers with cache". In: *CHES'03*. 2003, pp. 62–76.

Cache attacks

- side-channel attacks without physical access to the device
- the attacker only needs unprivileged execution on the victim's machine
  - realistic scenario with cloud environments, smartphone apps, JavaScript running on webpages...

Y. Tsunoo, T. Saito, and T. Suzaki. "Cryptanalysis of DES implemented on computers with cache". In: *CHES'03*. 2003, pp. 62–76.

Cache attacks

- side-channel attacks without physical access to the device
- the attacker only needs unprivileged execution on the victim's machine
  - realistic scenario with cloud environments, smartphone apps, JavaScript running on webpages...
- first practical attacks presented in 2003

Y. Tsunoo, T. Saito, and T. Suzaki. "Cryptanalysis of DES implemented on computers with cache". In: *CHES'03*. 2003, pp. 62–76.

Cache attacks

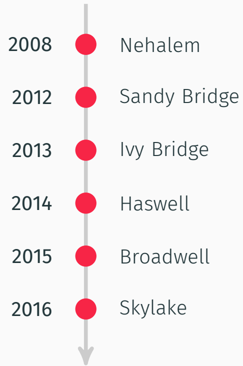
- side-channel attacks without physical access to the device
- the attacker only needs unprivileged execution on the victim's machine
  - realistic scenario with cloud environments, smartphone apps, JavaScript running on webpages...
- first practical attacks presented in 2003
- micro-architecture changed a lot, so did the attacks

Y. Tsunoo, T. Saito, and T. Suzaki. "Cryptanalysis of DES implemented on computers with cache". In: *CHES'03*. 2003, pp. 62–76.

Outline

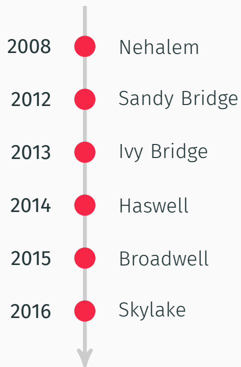
- cache side-channel attacks techniques
- a few words on applications
- challenges in modern architectures
- lessons learned and how to apply this to fault attacks on DRAM

Intel CPUs



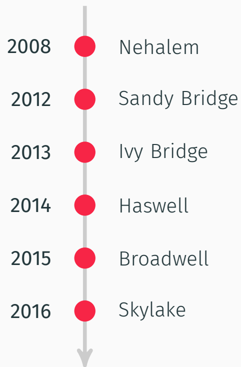
- new microarchitectures yearly

Intel CPUs



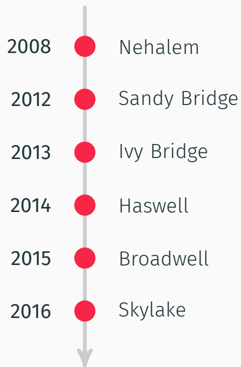
- new microarchitectures yearly
- performance improvement $\approx 5\%$

Intel CPUs



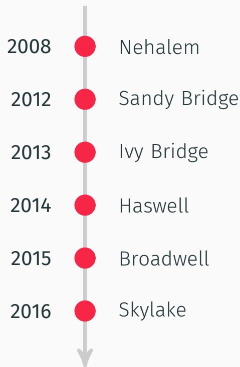
- new microarchitectures yearly
- performance improvement $\approx 5\%$
- very small optimizations: caches, branch prediction...

Intel CPUs



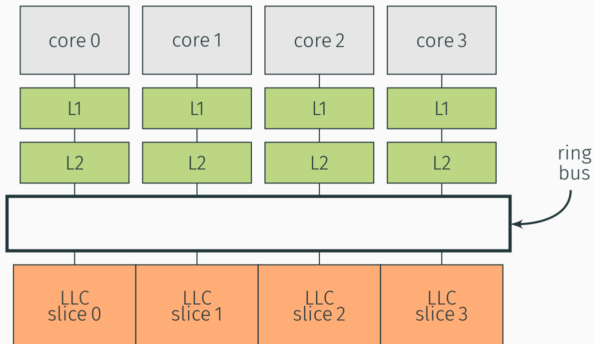
- new microarchitectures yearly
- performance improvement $\approx 5\%$
- very small optimizations: caches, branch prediction...
- no documentation on this intellectual property

Intel CPUs



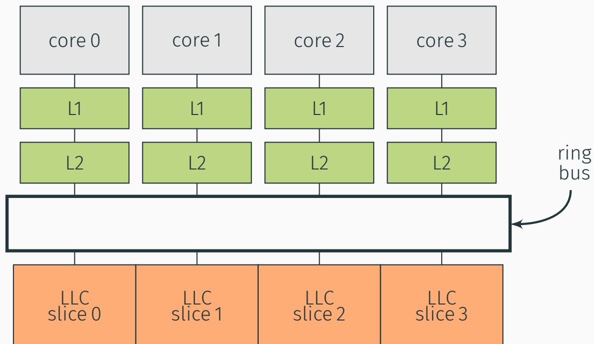
- new microarchitectures yearly
- performance improvement $\approx 5\%$
- very small optimizations: caches, branch prediction...
- no documentation on this intellectual property
- side channels come from these optimizations

Caches on Intel CPUs



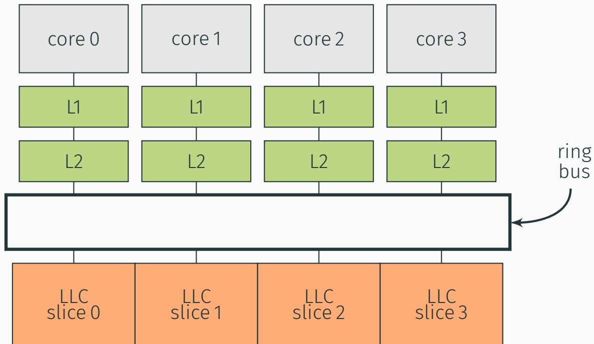
- L1 and L2 are private

Caches on Intel CPUs



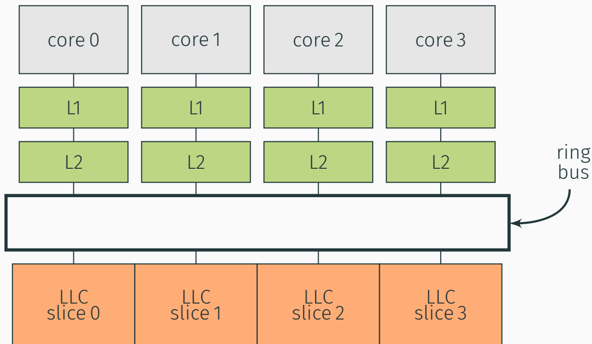
- L1 and L2 are private
- last-level cache

Caches on Intel CPUs



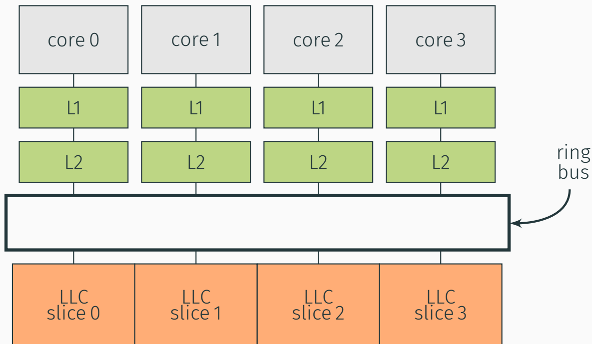
- L1 and L2 are private
- last-level cache
  - divided in slices

Caches on Intel CPUs



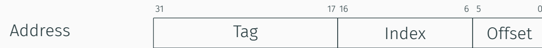
- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores

Caches on Intel CPUs



- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive

Set-associative caches



Cache

Set-associative caches

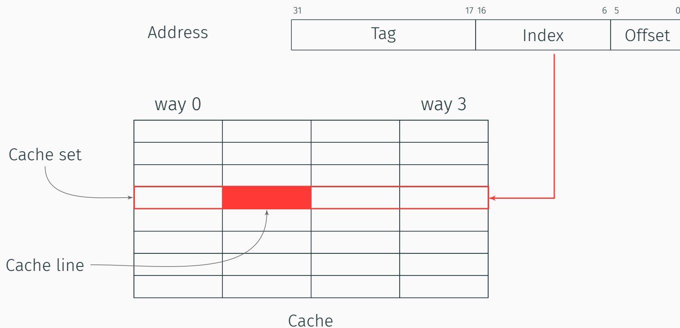
Data loaded in a specific set depending on its address

Set-associative caches

Data loaded in a specific set depending on its address

Several ways per set

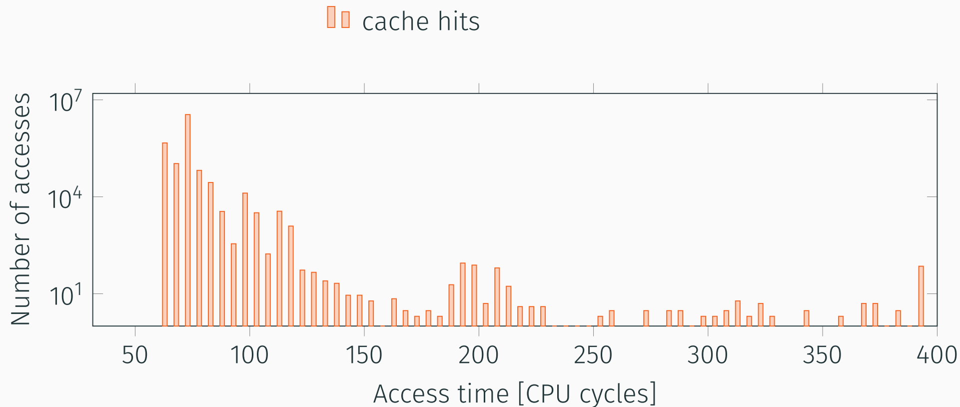
Set-associative caches



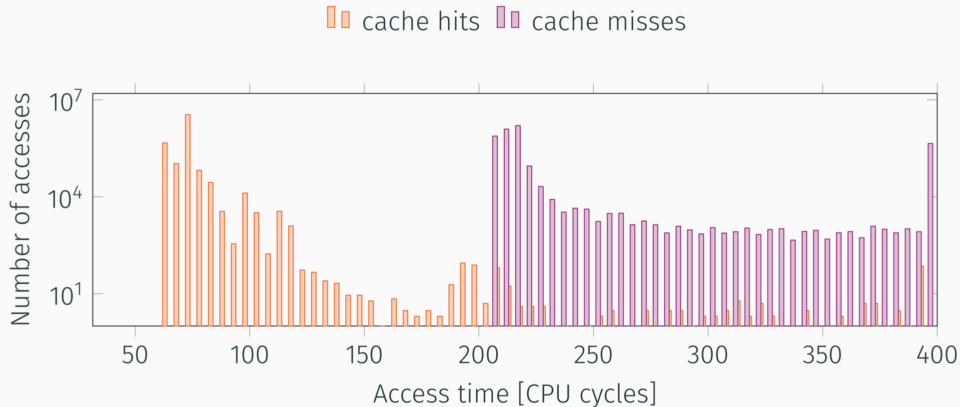
Data loaded in a specific set depending on its address

Several ways per set

Cache line loaded in a specific way depending on the replacement policy


cache hits
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]

Timing differences



Cache attacks

- cache attacks $\rightarrow$ exploit timing differences of memory accesses

Cache attacks

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content

Cache attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
    - not allowed to do so, e.g., across VMs

Cache attacks

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- side-channel attack: one malicious process spies on benign processes
  - e.g., steals crypto keys, spies on keystrokes

Cache attacks techniques

- two (main) techniques
  1. Flush+Reload (Gullasch et al., Osvik et al., Yarom et al.)
  2. Prime+Probe (Percival, Osvik et al., Liu et al.)
- exploitable on x86 and ARM

D. Gullasch, E. Bangerter, and S. Krenn. “Cache Games – Bringing Access-Based Cache Attacks on AES to Practice”. In: *S&P’11*. 2011.

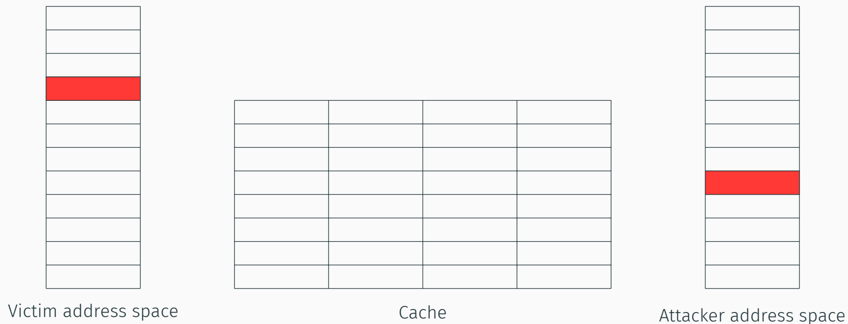
Y. Yarom and K. Falkner. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014.

D. A. Osvik, A. Shamir, and E. Tromer. “Cache Attacks and Countermeasures: the Case of AES”. In: *CT-RSA 2006*. 2006.

C. Percival. “Cache missing for fun and profit”. In: *Proceedings of BSDCan*. 2005.

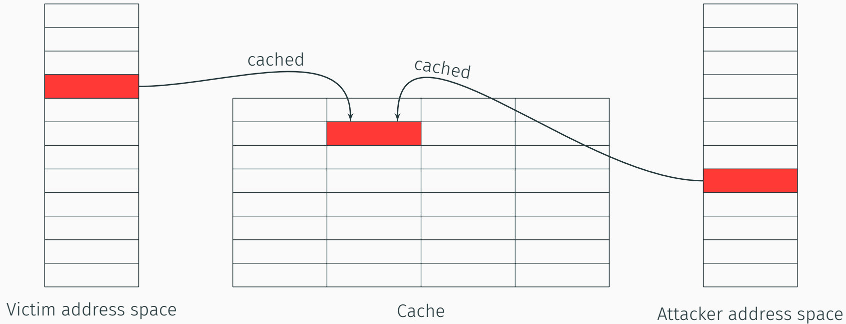
F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. “Last-Level Cache Side-Channel Attacks are Practical”. In: *S&P’15*. 2015.

Cache attacks: Flush+Reload



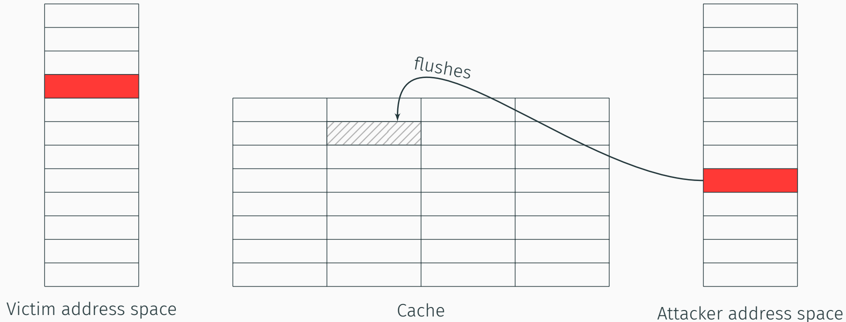
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attacks: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

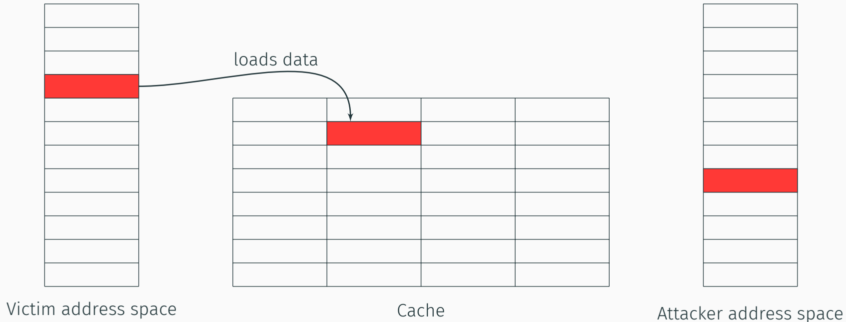
Cache attacks: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Cache attacks: Flush+Reload

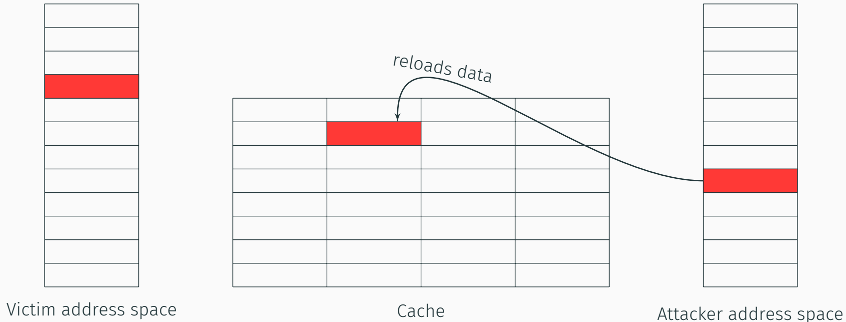


Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Cache attacks: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Step 4: Attacker reloads the data

What did the attacker learn?

- the victim accessed a particular cache line

What did the attacker learn?

- the victim accessed a particular cache line
- *i.e.*, every bit of the address except the lower 6
- with almost no false positives

Flush+Reload: Applications

- cross-VM side channel attacks on crypto algorithms
  - RSA: 96.7% of secret key bits in a single signature
  - AES: full key recovery in 30000 dec. (a few seconds)

Y. Yarom and K. Falkner. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014

B. Gülmezoglu, M. S. Inci, T. Eisenbarth, and B. Sunar. “A Faster and More Realistic Flush+Reload Attack on AES”. In: *COSADE’15*. 2015

D. Gruss, R. Spreitzer, and S. Mangard. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015

`https://github.com/IAIK/cache_template_attacks`

Flush+Reload: Applications

- cross-VM side channel attacks on crypto algorithms
  - RSA: 96.7% of secret key bits in a single signature
  - AES: full key recovery in 30000 dec. (a few seconds)
- Cache Template Attacks: automatically finds information leakage

  → side channel on keystrokes and AES T-tables implementation

---

Y. Yarom and K. Falkner. "Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack". In: *USENIX Security Symposium*. 2014

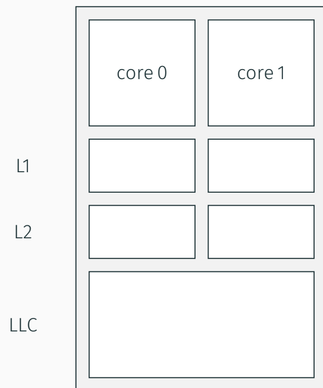
B. Gülmezoglu, M. S. Inci, T. Eisenbarth, and B. Sunar. "A Faster and More Realistic Flush+Reload Attack on AES". In: *COSADE'15*. 2015

D. Gruss, R. Spreitzer, and S. Mangard. "Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches". In: *USENIX Security Symposium*. 2015

`https://github.com/IAIK/cache_template_attacks`

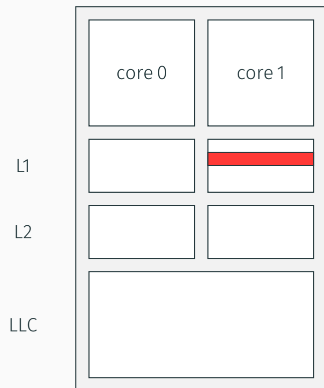
What if there is no shared memory?

Inclusive property



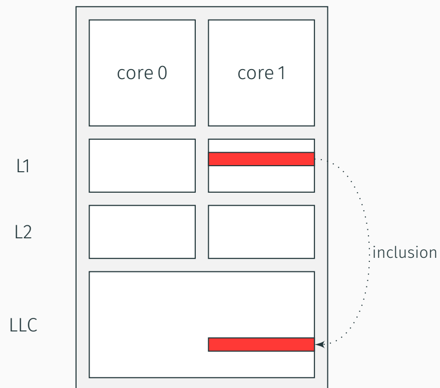
- inclusive LLC: superset of L1 and L2

Inclusive property



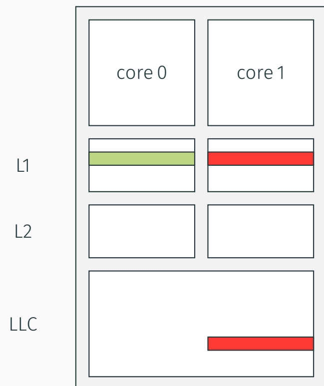
- inclusive LLC: superset of L1 and L2

Inclusive property



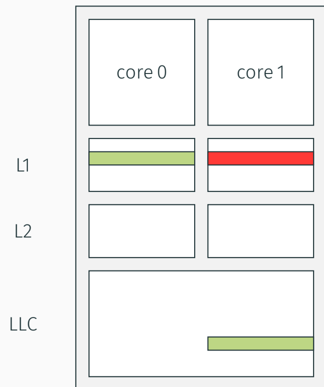
- inclusive LLC: superset of L1 and L2

Inclusive property



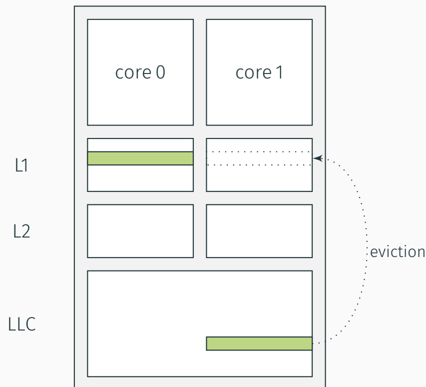
- inclusive LLC: superset of L1 and L2

Inclusive property



- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2

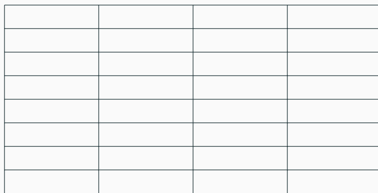
Inclusive property



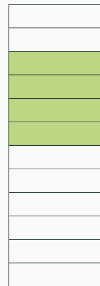
- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
- a core can evict lines in the private L1 of another core

Cache attacks: Prime+Probe

Victim address space

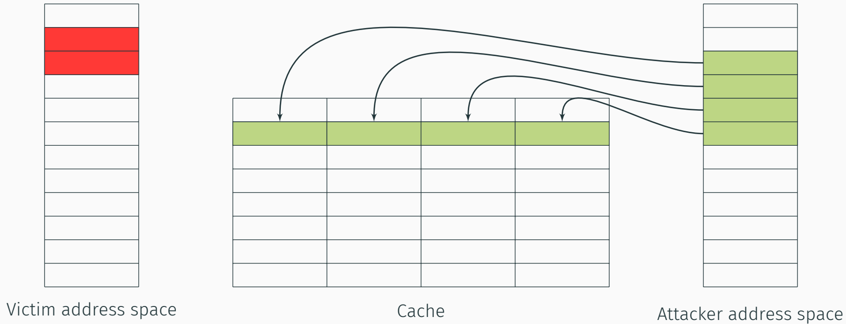


Cache



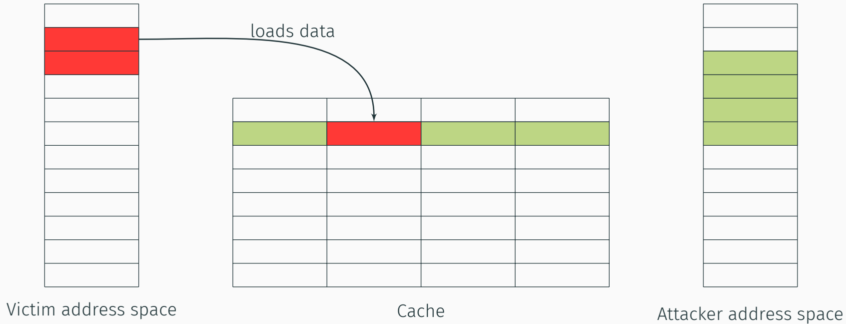
Attacker address space

Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

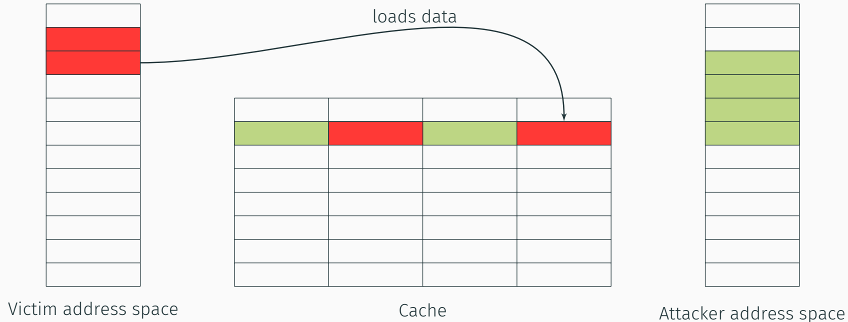
Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

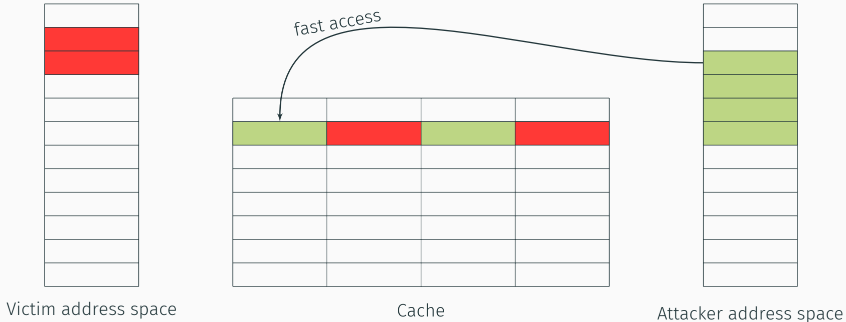
Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

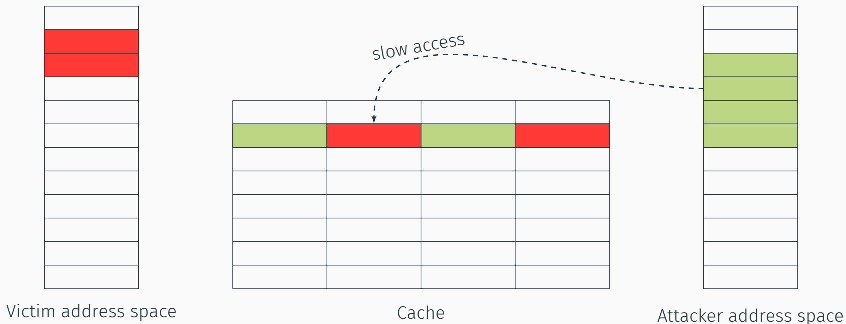


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

Cache attacks: Prime+Probe

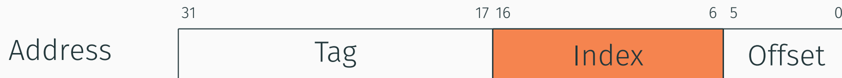


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

What did the attacker learn?



- a program accessed cache lines mapping to the same cache set

What did the attacker learn?

- a program accessed cache lines mapping to the same cache set
- *i.e.*, the index bits, $\approx$ 11 bits in modern last-level caches
- with false positives

Prime+Probe: Applications

- cross-VM side channel attacks on crypto algorithms:
  - El Gamal (sliding window): full key recovery in 12 min.
- tracking user behavior in the browser, in JavaScript
- covert channels between virtual machines in the cloud

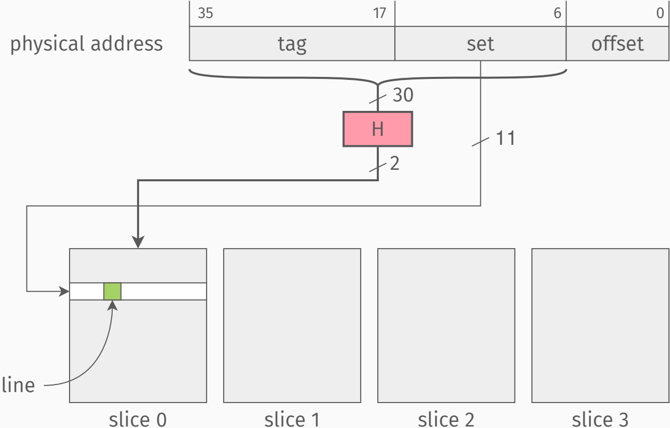
F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. "Last-Level Cache Side-Channel Attacks are Practical". In: *S&P'15*. 2015.

Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications". In: *CCS'15*. 2015.

C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, S. Mangard, and K. Römer. "Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud". In: *NDSS'17*. 2017.

Is that it?

Last-level cache addressing



Prime+Probe technical issues

- no need for e.g., memory deduplication $\rightarrow$ more practical

Prime+Probe technical issues

- no need for e.g., memory deduplication $\rightarrow$ more practical
- but requires:

Prime+Probe technical issues

- no need for e.g., memory deduplication $\rightarrow$ more practical
- but requires:
  1. eviction sets, *i.e.*, addresses in the same cache set, in the same slice

Prime+Probe technical issues

- no need for e.g., memory deduplication $\rightarrow$ more practical
- but requires:
  1. eviction sets, *i.e.*, addresses in the same cache set, in the same slice
  2. actually evicting addresses, *i.e.*, accessing addresses with some strategy

Prime+Probe technical issues

- no need for e.g., memory deduplication $\rightarrow$ more practical
- but requires:
    1. eviction sets, *i.e.*, addresses in the same cache set, in the same slice
    2. actually evicting addresses, *i.e.*, accessing addresses with some strategy
- issues:

Prime+Probe technical issues

- no need for e.g., memory deduplication $\rightarrow$ more practical
- but requires:
  1. eviction sets, *i.e.*, addresses in the same cache set, in the same slice
  2. actually evicting addresses, *i.e.*, accessing addresses with some strategy
- issues:
  1. the last-level cache addressing function is undocumented

Prime+Probe technical issues

- no need for e.g., memory deduplication $\rightarrow$ more practical
- but requires:
  1. eviction sets, *i.e.*, addresses in the same cache set, in the same slice
  2. actually evicting addresses, *i.e.*, accessing addresses with some strategy
- issues:
  1. the last-level cache addressing function is undocumented
  2. the replacement policy is (mostly) undocumented

Reverse-engineering last-level cache addressing

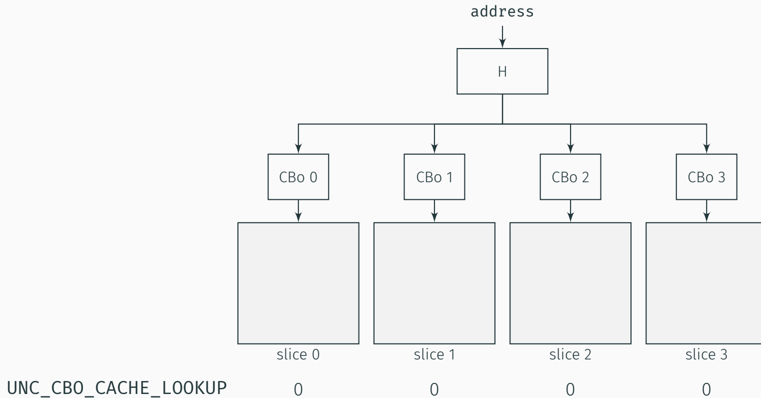
We reverse-engineered this function!

Intuition

1. find some way to map one address to one slice
2. repeat for every address with a 64B stride
3. infer a function out of it

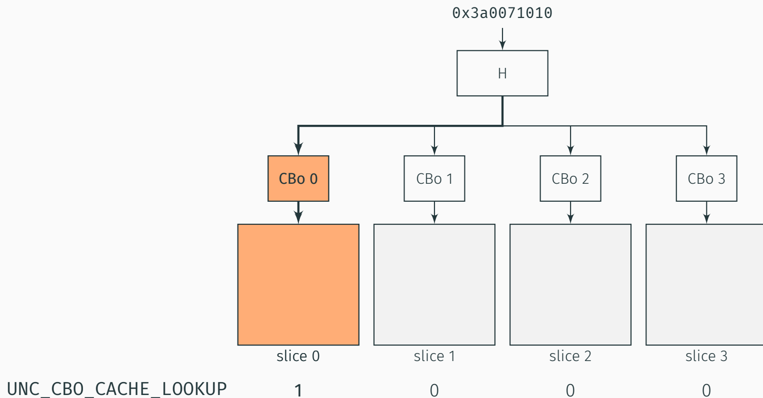
Mapping addresses to slices with performance counters

- event UNC_CBO_CACHE_LOOKUP counts accesses to a slice



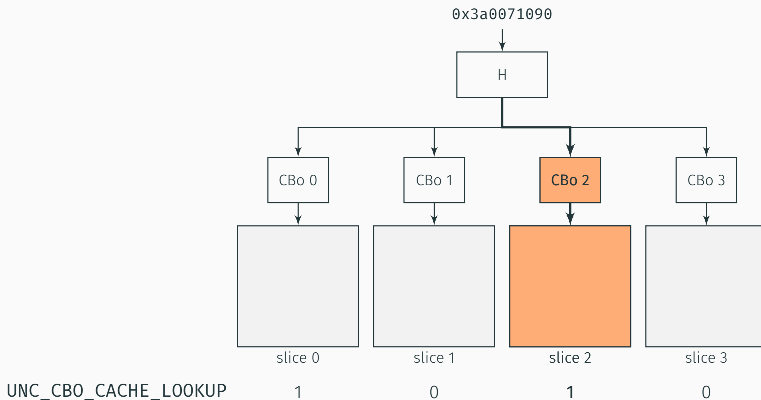
Mapping addresses to slices with performance counters

- event **UNC_CBO_CACHE_LOOKUP** counts accesses to a slice



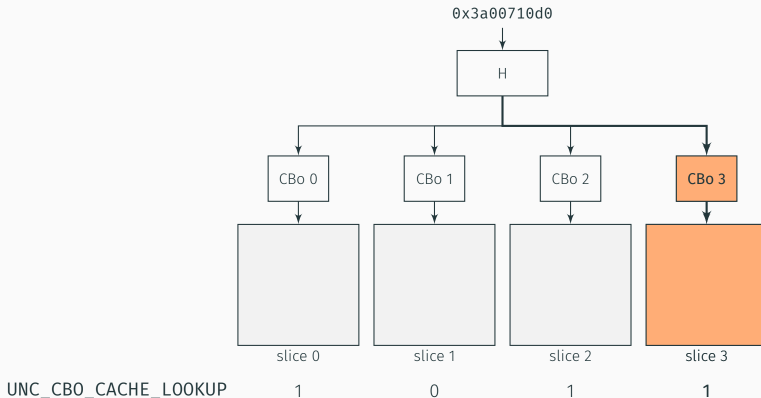
Mapping addresses to slices with performance counters

- event **UNC_CBO_CACHE_LOOKUP** counts accesses to a slice



Mapping addresses to slices with performance counters

- event **UNC_CBO_CACHE_LOOKUP** counts accesses to a slice



Last-level cache linear functions

3 functions, depending on the number of cores

		Address bit																													
		37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06
2 cores	o_0						⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
4 cores	o_0					⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
	o_1				⊕	⊕		⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕		⊕				⊕	
8 cores	o_0		⊕	⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
	o_1	⊕		⊕	⊕	⊕		⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕		⊕				⊕	
	o_2	⊕	⊕	⊕	⊕			⊕	⊕			⊕	⊕			⊕	⊕			⊕			⊕			⊕	⊕				⊕		

- valid for Sandy Bridge, Ivy Bridge, Haswell, Broadwell, whether Core or Xeon
- different for 6, 10, 12... cores $\rightarrow$ non-linear
- different for Skylake

Lessons learned from cache side-channel attacks

- undocumented hardware can be a problem,

Lessons learned from cache side-channel attacks

- undocumented hardware can be a problem, but not for long :)

Lessons learned from cache side-channel attacks

- undocumented hardware can be a problem, but not for long :)
- removing `clflush` does not address the root causes of vulnerabilities

Lessons learned from cache side-channel attacks

- undocumented hardware can be a problem, but not for long :)
- removing `clflush` does not address the root causes of vulnerabilities
- fixing crypto is (relatively) easy, but mitigating all cache attacks is hard

How do we make fault attacks out of that?

DRAM fault attacks

- we're now exploring fault attacks on DRAM

DRAM fault attacks

- we're now exploring fault attacks on DRAM
- attack entirely in software, again no physical access

DRAM fault attacks

- we're now exploring fault attacks on DRAM
- attack entirely in software, again no physical access

→ how can we flip bits without accessing them?

DRAM fault attacks

- we’re now exploring fault attacks on DRAM
- attack entirely in software, again no physical access

→ how can we flip bits without accessing them?

- we’ll conduct attacks on the cache to create the right conditions

DRAM fault attacks

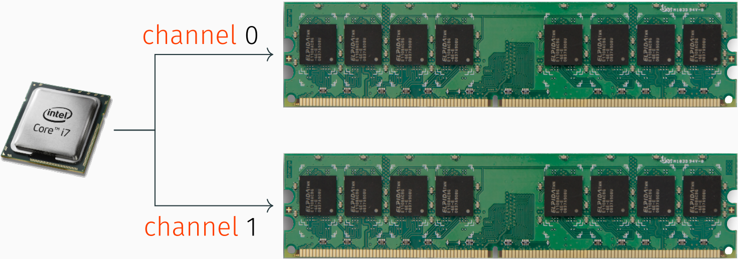
- we're now exploring fault attacks on DRAM
- attack entirely in software, again no physical access

→ how can we flip bits without accessing them?

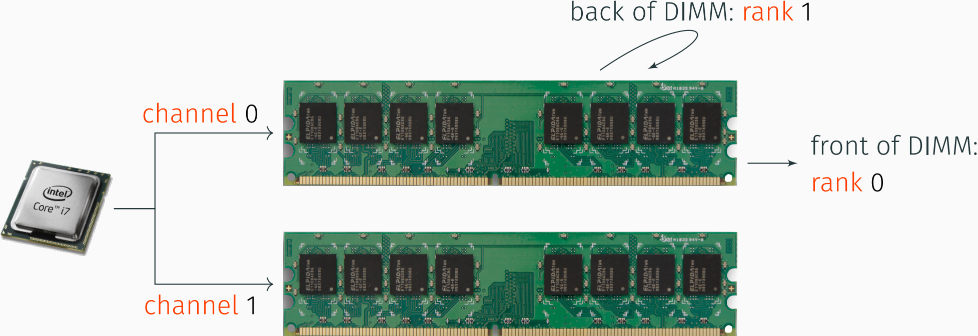
- we'll conduct attacks on the cache to create the right conditions
- (but we're not flipping bits on the cache)

Background: DRAM organization

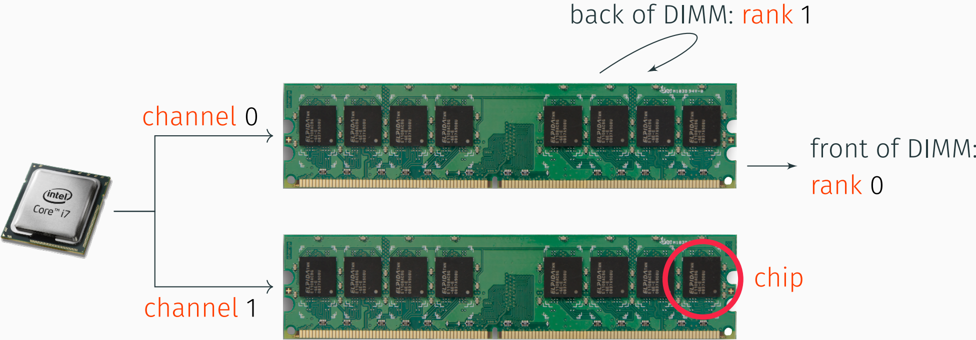
Background: DRAM organization



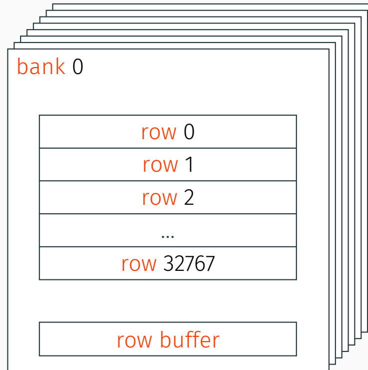
Background: DRAM organization



Background: DRAM organization



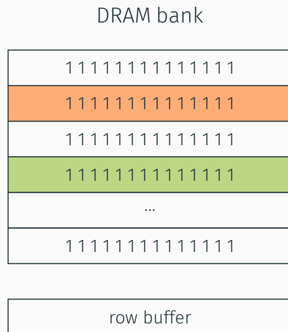
Background: DRAM organization



- bits in cells in rows
- access: activate row, copy to row buffer

Software-Based Fault Attack: Rowhammer

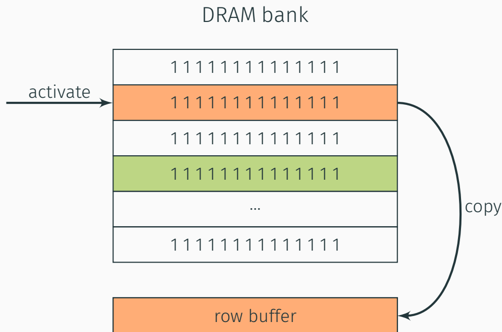
Rowhammer (Kim et al., 2014)
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



Software-Based Fault Attack: Rowhammer

Rowhammer (Kim et al., 2014)

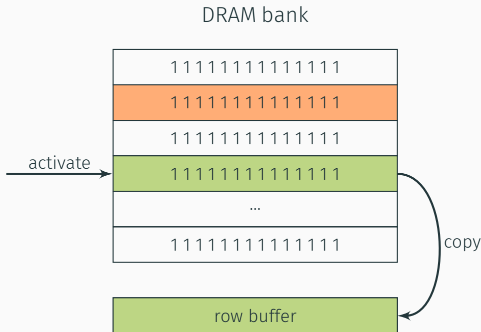
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



Software-Based Fault Attack: Rowhammer

Rowhammer (Kim et al., 2014)

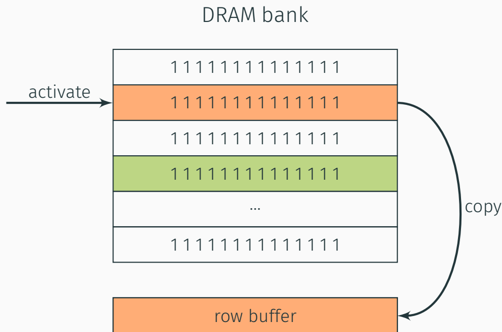
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



Software-Based Fault Attack: Rowhammer

Rowhammer (Kim et al., 2014)

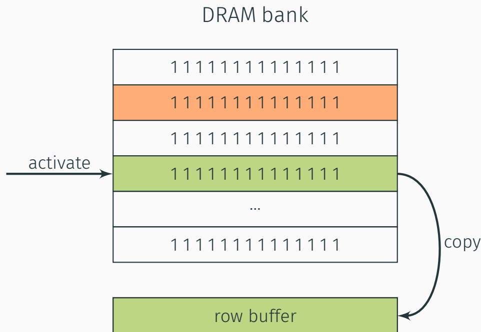
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



Software-Based Fault Attack: Rowhammer

Rowhammer (Kim et al., 2014)

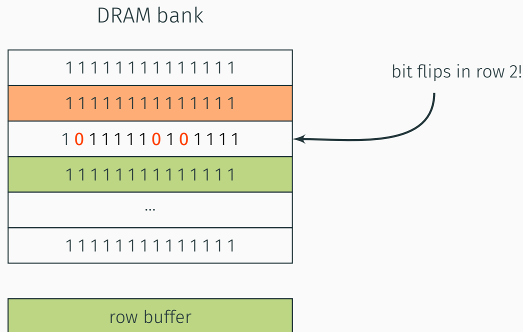
"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



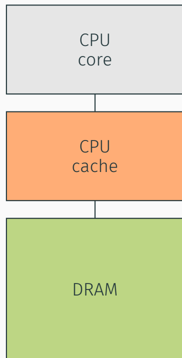
Software-Based Fault Attack: Rowhammer

Rowhammer (Kim et al., 2014)

"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



Impact of the CPU cache

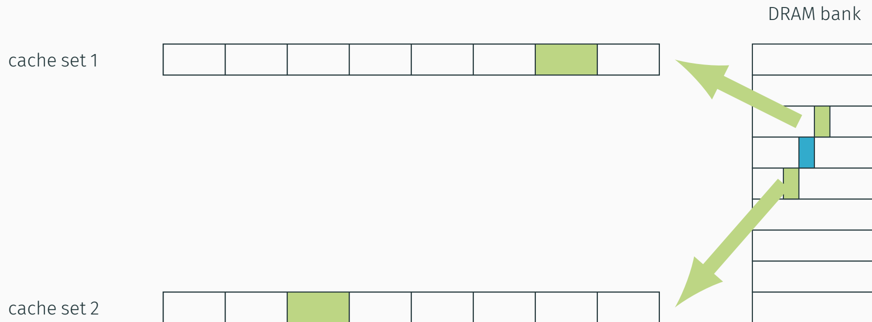


- only non-cached accesses reach DRAM
- original attacks use `clflush` instruction

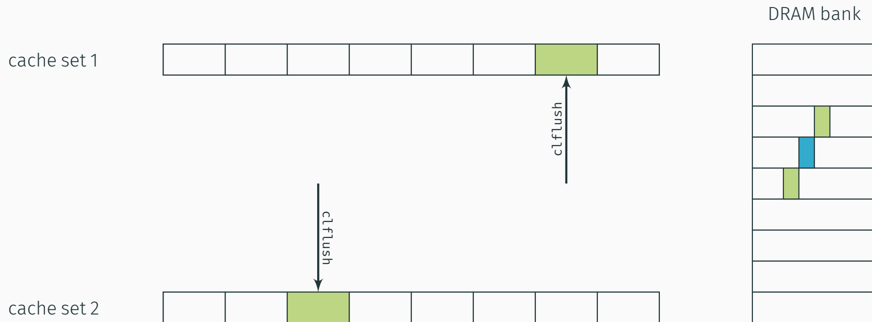
→ flush line from cache

→ next access will be served from DRAM

Rowhammer (with `clflush`)



Rowhammer (with clflush)



Rowhammer (with clflush)

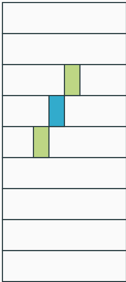
cache set 1

clflush

clflush

cache set 2

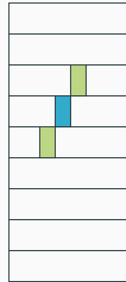
DRAM bank



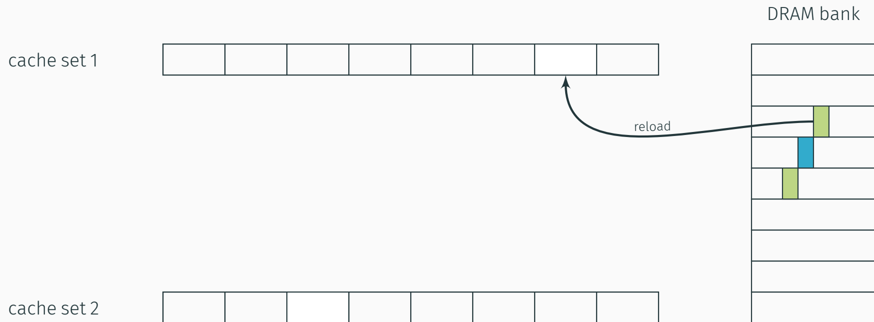
Rowhammer (with `clflush`)

cache set 1

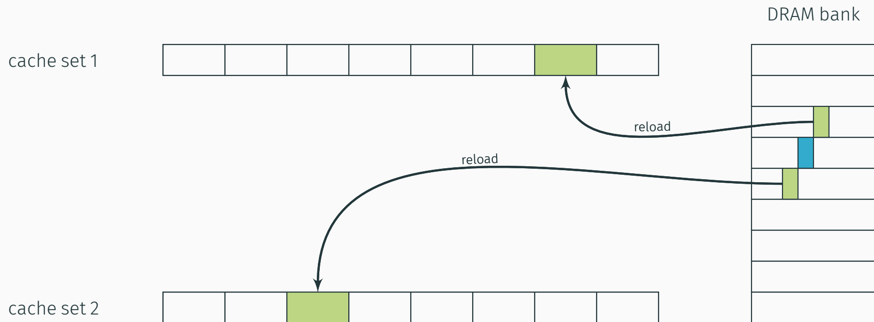
cache set 2



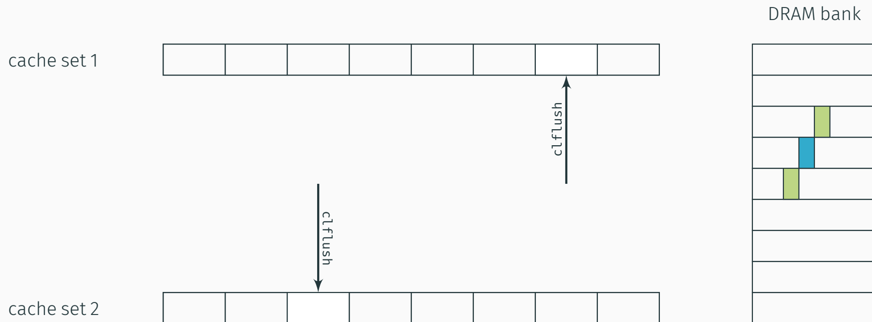
Rowhammer (with `clflush`)



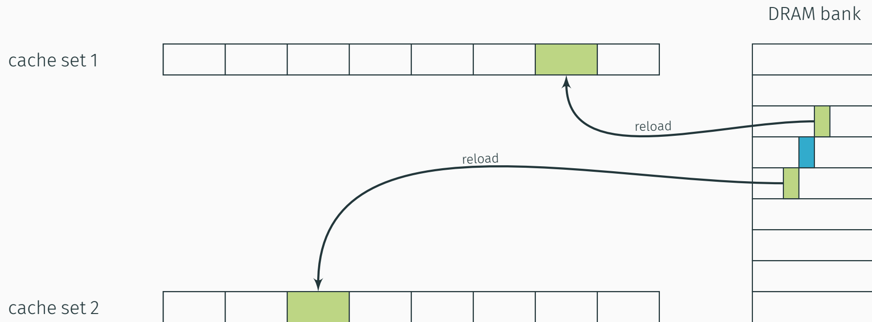
Rowhammer (with `clflush`)



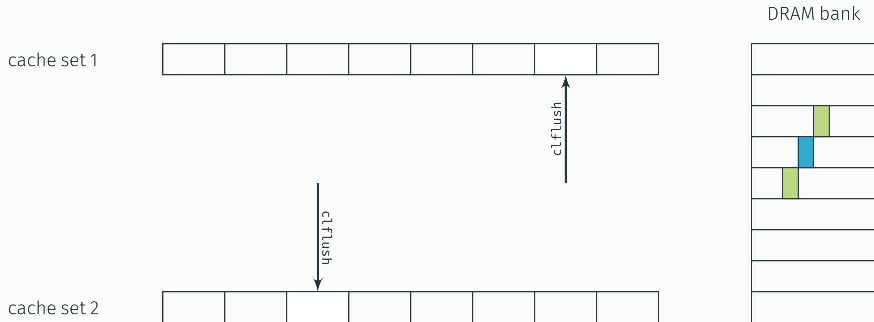
Rowhammer (with `clflush`)



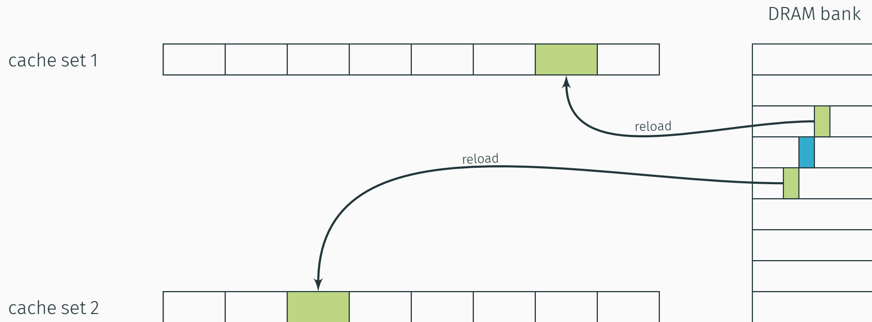
Rowhammer (with `clflush`)



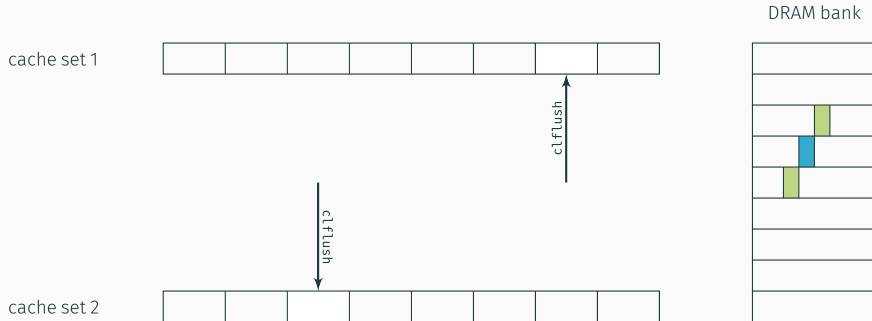
Rowhammer (with `clflush`)



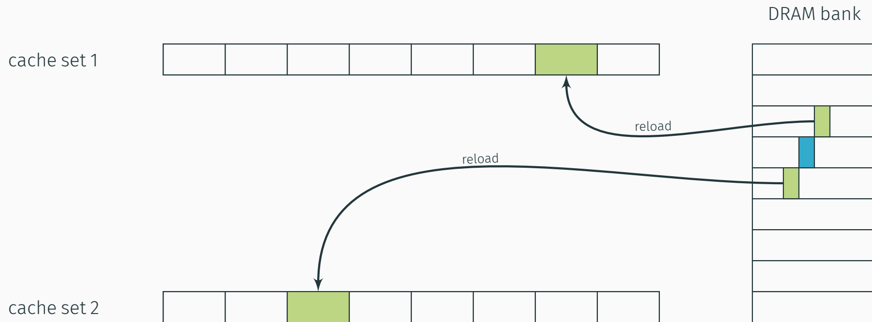
Rowhammer (with `clflush`)



Rowhammer (with `clflush`)



Rowhammer (with `clflush`)



Rowhammer (with clflush)

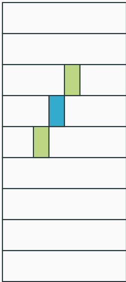
cache set 1

clflush

clflush

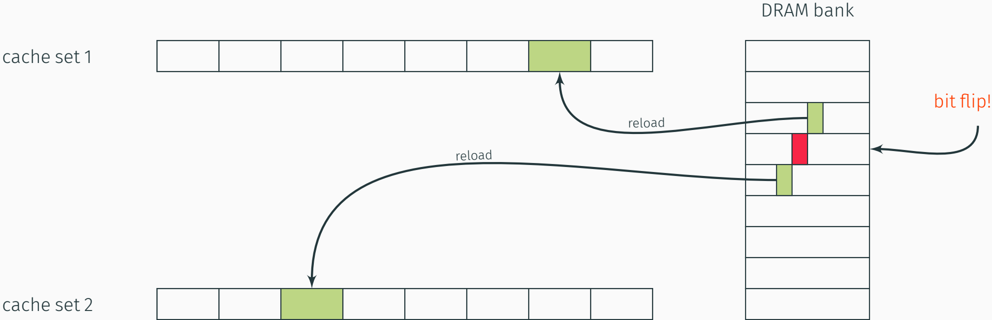
cache set 2

DRAM bank



wait for it...

Rowhammer (with `clflush`)



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`?

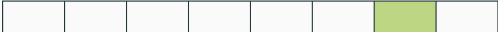
- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript
- our approach: use regular memory accesses for eviction
  → techniques from cache attacks!

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!

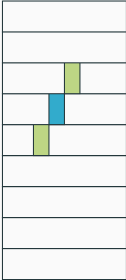
Rowhammer without `clflush`

cache set 1

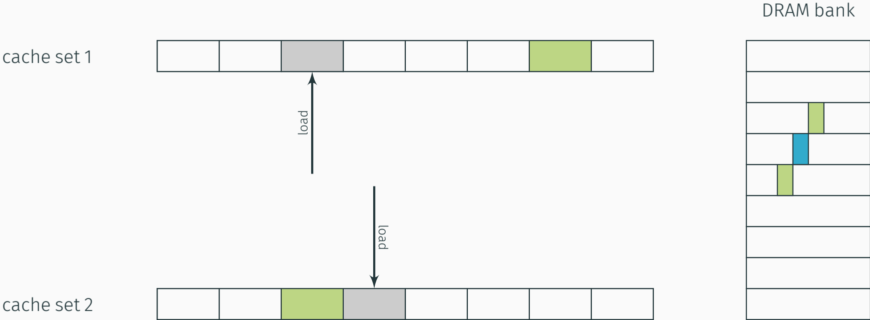


cache set 2

DRAM bank



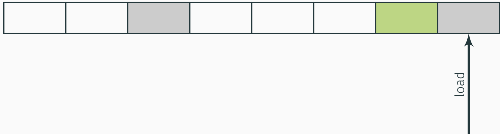
Rowhammer without `clflush`

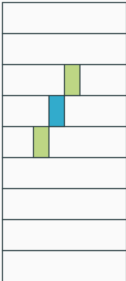


Rowhammer without `clflush`

DRAM bank

cache set 1

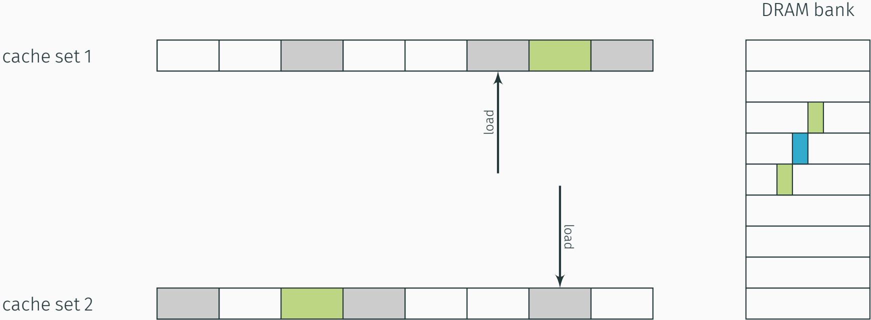




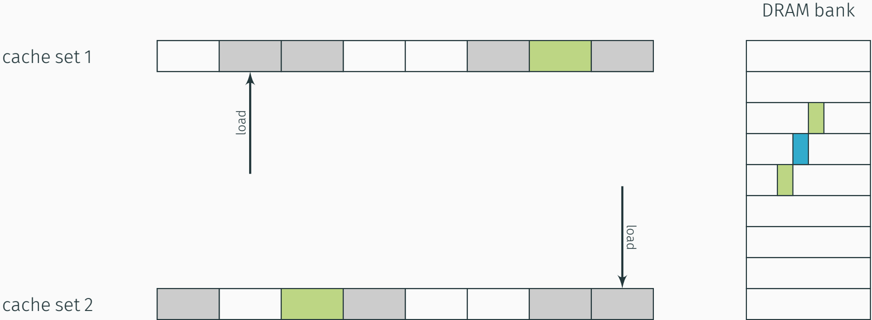
load

cache set 2

Rowhammer without `clflush`



Rowhammer without `clflush`



Rowhammer without `clflush`

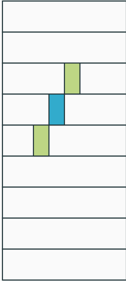
DRAM bank

cache set 1

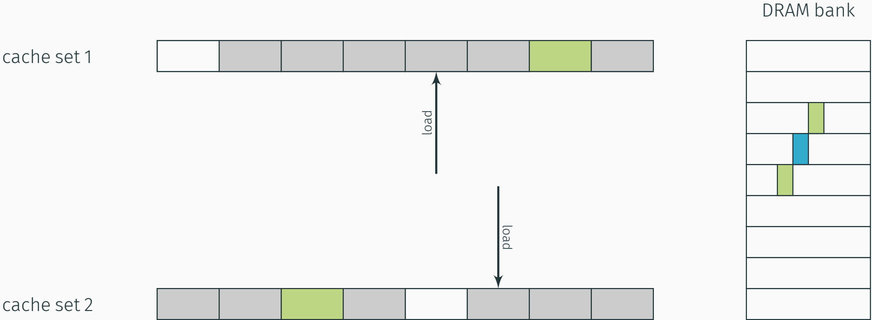
load

load

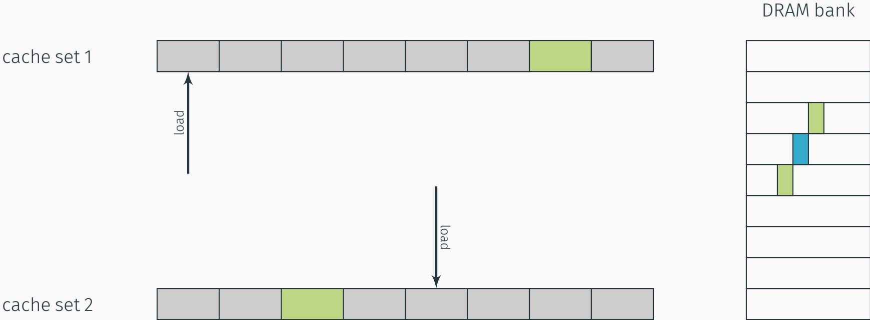
cache set 2



Rowhammer without `clflush`



Rowhammer without clflush



Rowhammer without `clflush`

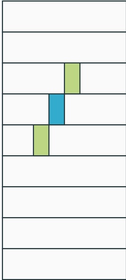
cache set 1

load

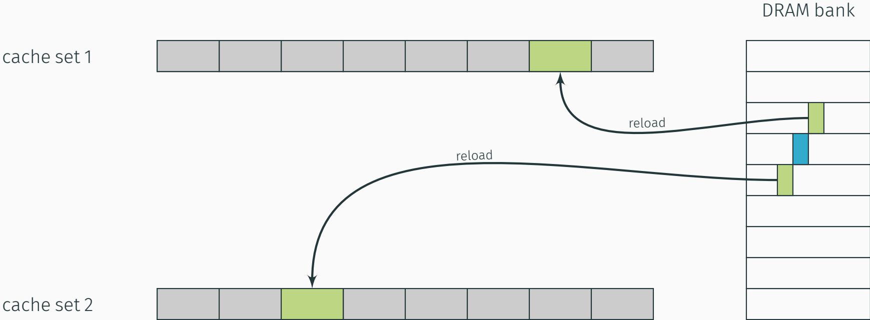
load

cache set 2

DRAM bank



Rowhammer without `clflush`



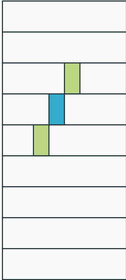
Rowhammer without `clflush`

cache set 1

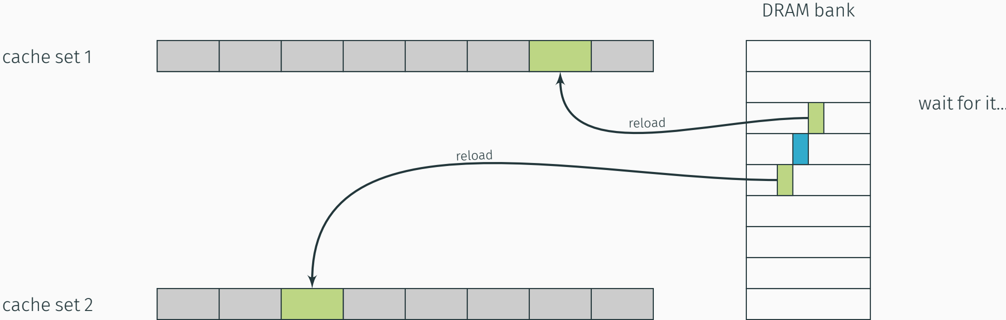
repeat!

cache set 2

DRAM bank



Rowhammer without `clflush`

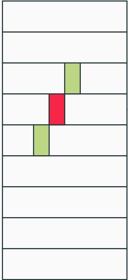


Rowhammer without clflush

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

$\rightarrow$ optimize the eviction rate and the timing

Meanwhile, in reality...

- existing eviction strategies consider that the replacement policy is LRU

Meanwhile, in reality...

- existing eviction strategies consider that the replacement policy is LRU
- it’s not

Meanwhile, in reality...

- existing eviction strategies consider that the replacement policy is LRU
- it's not (and it's undocumented)

Meanwhile, in reality...

- existing eviction strategies consider that the replacement policy is LRU
- it's not (and it's undocumented)
- either we do not evict with high enough probability, or we are too slow

Meanwhile, in reality...

- existing eviction strategies consider that the replacement policy is LRU
- it's not (and it's undocumented)
- either we do not evict with high enough probability, or we are too slow

→ no bit flip

Meanwhile, in reality...

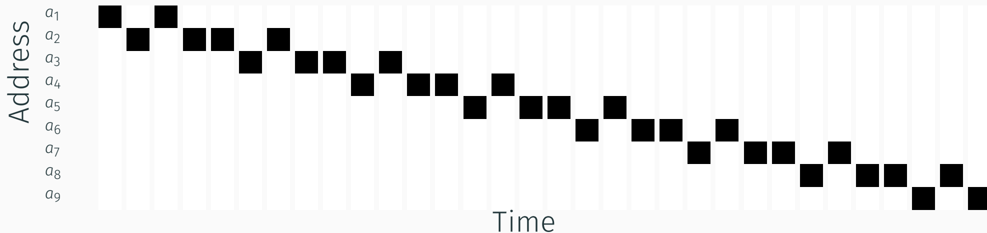
- existing eviction strategies consider that the replacement policy is LRU
- it's not (and it's undocumented)
- either we do not evict with high enough probability, or we are too slow

→ no bit flip

Poking around, we learned two things:

1. adding more unique addresses can increase eviction rate
2. multiple accesses to one address can increase the eviction rate

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: $1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, \ldots$
- 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

Cache eviction strategy: Notation (1)

Write eviction strategies as: $\mathscr{P}$-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];
```

Cache eviction strategy: Notation (1)

Write eviction strategies as: $\mathscr{P}$-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: $\mathscr{P}$-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: $\mathscr{P}$-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];
```

L: step size of the inner access loop

Cache eviction strategy: Notation (1)

Write eviction strategies as: $\mathscr{P}$-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];
```

L: step size of the inner access loop

C: number of repetitions of the inner access loop

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];
```

- $\mathscr{P}$-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];
```

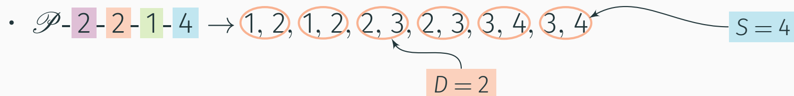
Cache eviction strategy: Notation (2)

```
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];
```

- $\mathscr{P}$-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4 ← $S = 4$

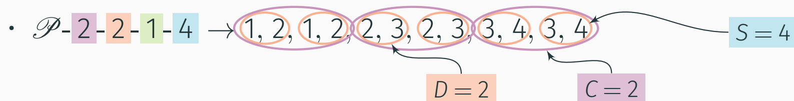
Cache eviction strategy: Notation (2)

```
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];
```



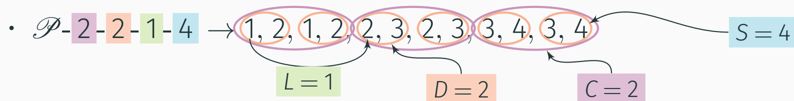
Cache eviction strategy: Notation (2)

```
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];
```



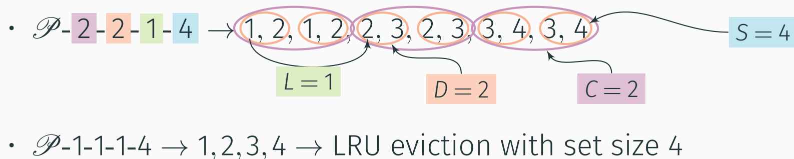
Cache eviction strategy: Notation (2)

```
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];
```



- $\mathscr{P}$-1-1-1-4 → 1, 2, 3, 4 → LRU eviction with set size 4

Better eviction strategies

We evaluated more than 10 000 strategies...

strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17		
$\mathscr{P}$-1-1-1-20	20		

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

Better eviction strategies

We evaluated more than 10 000 strategies...

strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17	74.46 % ✗	
$\mathscr{P}$-1-1-1-20	20	99.82 % ✓	

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

Better eviction strategies

We evaluated more than 10 000 strategies...

strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathscr{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗

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

Better eviction strategies

We evaluated more than 10 000 strategies...

strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathscr{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathscr{P}$-2-1-1-17	34		

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

Better eviction strategies

We evaluated more than 10 000 strategies...

strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathscr{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathscr{P}$-2-1-1-17	34	99.86% ✓	

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

Better eviction strategies

We evaluated more than 10 000 strategies...

strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathscr{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathscr{P}$-2-1-1-17	34	99.86% ✓	191 ns ✓

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

Better eviction strategies

We evaluated more than 10 000 strategies...

strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathscr{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathscr{P}$-2-1-1-17	34	99.86% ✓	191 ns ✓
$\mathscr{P}$-2-2-1-17	64		

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

Better eviction strategies

We evaluated more than 10 000 strategies...

strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathscr{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathscr{P}$-2-1-1-17	34	99.86% ✓	191 ns ✓
$\mathscr{P}$-2-2-1-17	64	99.98% ✓	

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

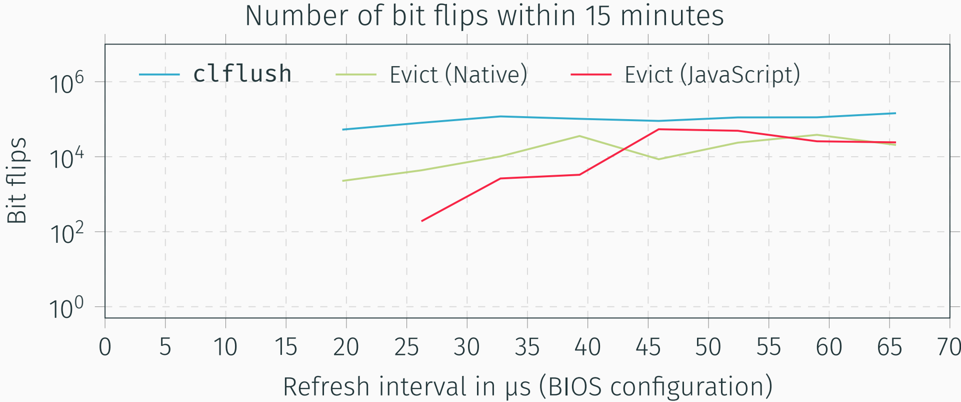
Better eviction strategies

We evaluated more than 10 000 strategies...

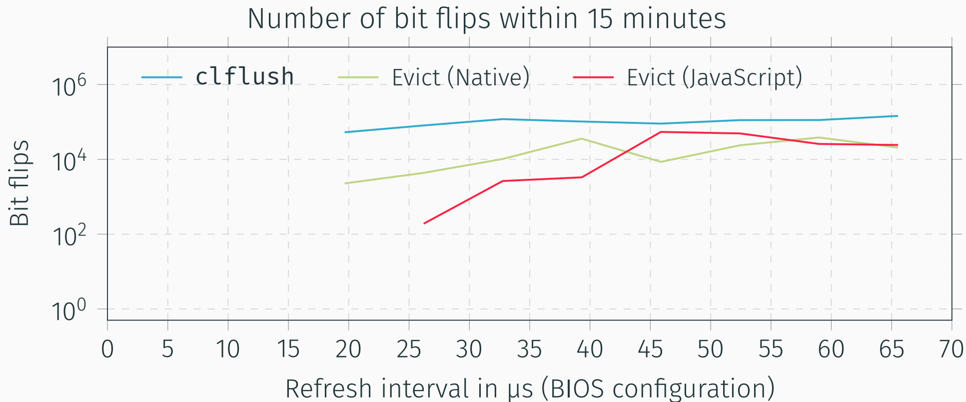
strategy	# accesses	eviction rate	loop time
$\mathscr{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathscr{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathscr{P}$-2-1-1-17	34	99.86% ✓	191 ns ✓
$\mathscr{P}$-2-2-1-17	64	99.98% ✓	180 ns ✓

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

Evaluation on Haswell



Evaluation on Haswell



First remote software-induced fault attack from a browser, in JavaScript!

Lessons learned from Rowhammer.js

- DRAM vulnerable to the native attack $\neq$ vulnerable to the JavaScript version

E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *S&P'16*. 2016.

Lessons learned from Rowhammer.js

- DRAM vulnerable to the native attack $\neq$ vulnerable to the JavaScript version
- most DRAM modules vulnerable to the native attack are also without `clflush`

E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector”. In: *S&P'16*. 2016.

Lessons learned from Rowhammer.js

- DRAM vulnerable to the native attack $\neq$ vulnerable to the JavaScript version
- most DRAM modules vulnerable to the native attack are also without `clflush` $\rightarrow$ removing `clflush` is not a countermeasure

E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *S&P'16*. 2016.

- DRAM vulnerable to the native attack $\neq$ vulnerable to the JavaScript version
- most DRAM modules vulnerable to the native attack are also without `clflush` $\rightarrow$ removing `clflush` is not a countermeasure

What we should have learned from Rowhammer but apparently didn't

E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *S&P'16*. 2016.

Lessons learned from Rowhammer.js

- DRAM vulnerable to the native attack $\neq$ vulnerable to the JavaScript version
- most DRAM modules vulnerable to the native attack are also without `clflush` $\rightarrow$ removing `clflush` is not a countermeasure

What we should have learned from Rowhammer but apparently didn't

- nobody managed to exploit that yet $\neq$ we can't exploit that

E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *S&P'16*. 2016.

Lessons learned from Rowhammer.js

- DRAM vulnerable to the native attack $\neq$ vulnerable to the JavaScript version
- most DRAM modules vulnerable to the native attack are also without `clflush` $\rightarrow$ removing `clflush` is not a countermeasure

What we should have learned from Rowhammer but apparently didn't

- nobody managed to exploit that yet $\neq$ we can't exploit that

$\rightarrow$ exploit by Bosman et al. after we released the code for Rowhammer.js

E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *S&P'16*. 2016.

Conclusions

- lessons from side-channel attacks useful to investigate fault attacks
- micro-architectural attacks are hard to mitigate
- probably more software-based fault attacks to come

Shameless advertisement

I am hiring a postdoc!

Cache attacks: From side channels to fault attacks

Clémentine Maurice – CNRS, Rennes, France

November 16, 2017 – Cryptacus Workshop, Nijmegen, Netherlands