Cache attacks:
Software side-channel and fault attacks

Clémentine Maurice, Graz University of Technology

May 11, 2017—Spring school on Security & Correctness in the IoT 2017

Who am I

- Clémentine Maurice
- PhD in computer science, Postdoc @ Graz University Of Technology
- @BloodyTangerine
- clementine.maurice@iaik.tugraz.at

Who am I

- Clémentine Maurice
- PhD in computer science, Postdoc @ Graz University Of Technology
- @BloodyTangerine
- clementine.maurice@iaik.tugraz.at

+ Secure Systems team: Daniel Gruss, Michael Schwarz, Moritz Lipp

Scope

- everyday hardware: servers, workstations, laptops, smartphones...

Scope

- everyday hardware: servers, workstations, laptops, smartphones...
- remote side-channel attacks

Side channels

- safe software infrastructure $\rightarrow$ no bugs, e.g., Heartbleed

Side channels

- safe software infrastructure → no bugs, e.g., Heartbleed
- does not mean safe execution

Side channels

- safe software infrastructure → no bugs, e.g., Heartbleed
- does not mean safe execution
- information leaks because of the hardware it runs on
- no “bug” in the sense of a mistake → lots of performance optimizations

Side channels

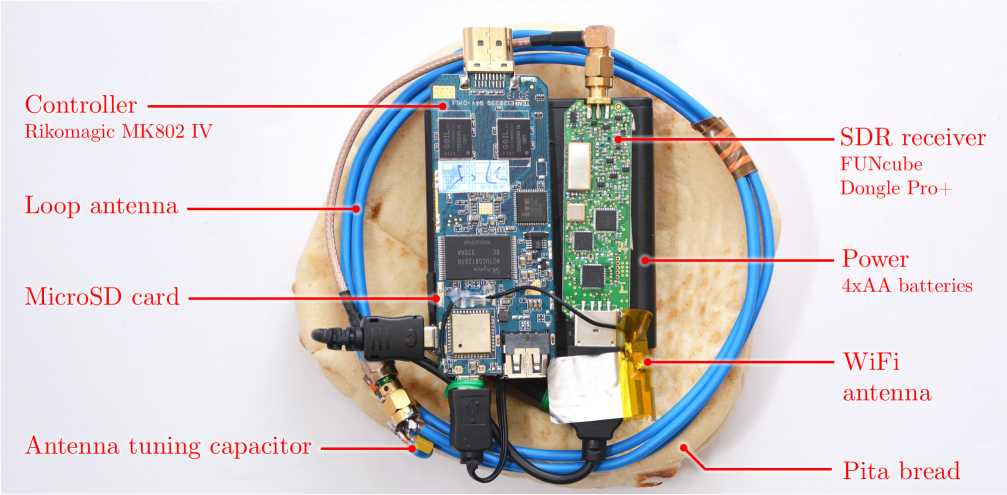
- safe software infrastructure → no bugs, e.g., Heartbleed
- does not mean safe execution
- information leaks because of the hardware it runs on
- no “bug” in the sense of a mistake → lots of performance optimizations

→ crypto and sensitive info., e.g., keystrokes and mouse movements

Sources of leakage

- via power consumption, electromagnetic leaks

Sources of leakage



Sources of leakage

- via power consumption, electromagnetic leaks
  - $\rightarrow$ targeted attacks, physical access

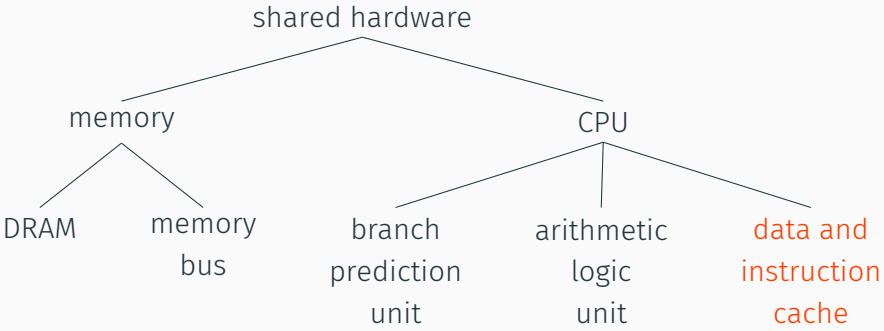
Sources of leakage

- via power consumption, electromagnetic leaks
  - $\rightarrow$ targeted attacks, physical access
- via shared hardware and microarchitecture

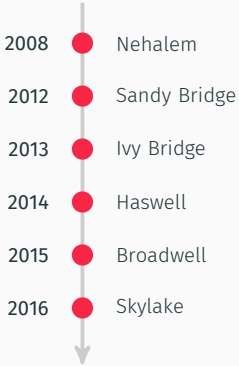
Sources of leakage

- via power consumption, electromagnetic leaks
  - → targeted attacks, physical access
- via shared hardware and microarchitecture
  - → remote attacks

Shared hardware

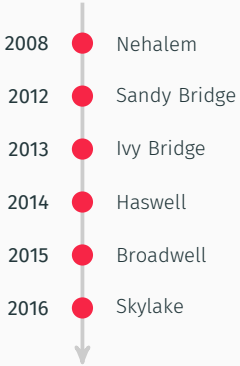


From small optimizations to side channels



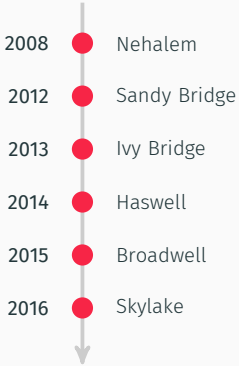
- new microarchitectures yearly

From small optimizations to side channels



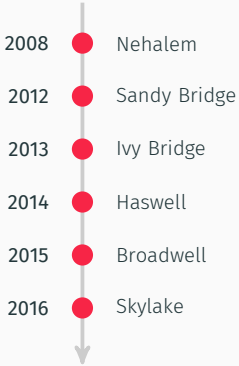
- new microarchitectures yearly
- performance improvement $\approx 5\%$

From small optimizations to side channels



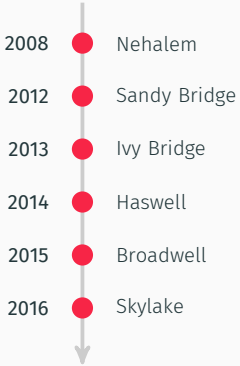
- new microarchitectures yearly
- performance improvement $\approx 5\%$
- very small optimizations: caches, branch prediction...

From small optimizations to side channels



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

From small optimizations to side channels



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

Today's CPU complexity

- "Intel x86 documentation has more pages than the 6502 has transistors"

Ken Shirriff, `http://www.righto.com/2013/09/intel-x86-documentation-has-more-pages.html`

Today's CPU complexity

- "Intel x86 documentation has more pages than the 6502 has transistors"
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...

Ken Shirriff, `http://www.righto.com/2013/09/intel-x86-documentation-has-more-pages.html`

Today's CPU complexity

- "Intel x86 documentation has more pages than the 6502 has transistors"
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...
  - year: 1975 → 3510 transistors

Ken Shirriff, `http://www.righto.com/2013/09/intel-x86-documentation-has-more-pages.html`

Today's CPU complexity

- "Intel x86 documentation has more pages than the 6502 has transistors"
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...
  - year: 1975 $\rightarrow$ 3510 transistors
- 22-core Intel Xeon Broadwell-E5

Ken Shirriff, `http://www.righto.com/2013/09/intel-x86-documentation-has-more-pages.html`

Today's CPU complexity

- "Intel x86 documentation has more pages than the 6502 has transistors"
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...
  - year: 1975 → 3510 transistors
- 22-core Intel Xeon Broadwell-E5
  - year: 2016 → 7.2 billion transistors

Ken Shirriff, `http://www.righto.com/2013/09/intel-x86-documentation-has-more-pages.html`

Today's CPU complexity

- "Intel x86 documentation has more pages than the 6502 has transistors"
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...
  - year: 1975 → 3510 transistors
- 22-core Intel Xeon Broadwell-E5
  - year: 2016 → 7.2 billion transistors
- Intel Software Developer's Manuals (sept. 2016): 4670 pages

---

Ken Shirriff, `http://www.righto.com/2013/09/intel-x86-documentation-has-more-pages.html`

Today's CPU complexity

- "Intel x86 documentation has more pages than the 6502 has transistors"
- 6502: 8-bit microprocessor, used in the Apple II, Commodore 64, Atari 800...
  - year: 1975 → 3510 transistors
- 22-core Intel Xeon Broadwell-E5
  - year: 2016 → 7.2 billion transistors
- Intel Software Developer's Manuals (sept. 2016): 4670 pages
- (there are actually more manuals than just the SDM)

Ken Shirriff, `http://www.righto.com/2013/09/intel-x86-documentation-has-more-pages.html`

Outline

- Background
- Cache covert channels
- Side-channel attacks on crypto, keystrokes, and KASLR
- Fault attacks on DRAM in JavaScript
- Countermeasures and conclusion

Cache Attack Techniques

MOV—Move

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
88 /r	MOV *r/m8,r8*	MR	Valid	Valid	Move *r8* to *r/m8*.
REX + 88 /r	MOV *r/m8****,*r8****	MR	Valid	N.E.	Move *r8* to *r/m8*.
89 /r	MOV *r/m16,r16*	MR	Valid	Valid	Move *r16* to *r/m16*.
89 /r	MOV *r/m32,r32*	MR	Valid	Valid	Move *r32* to *r/m32*.
REX.W + 89 /r	MOV *r/m64,r64*	MR	Valid	N.E.	Move *r64* to *r/m64*.
8A /r	MOV *r8,r/m8*	RM	Valid	Valid	Move *r/m8* to *r8*.
REX + 8A /r	MOV *r8****,*r/m8****	RM	Valid	N.E.	Move *r/m8* to *r8*.
8B /r	MOV *r16,r/m16*	RM	Valid	Valid	Move *r/m16* to *r16*.
8B /r	MOV *r32,r/m32*	RM	Valid	Valid	Move *r/m32* to *r32*.
REX.W + 8B /r	MOV *r64,r/m64*	RM	Valid	N.E.	Move *r/m64* to *r64*.
8C /r	MOV *r/m16,Sreg***	MR	Valid	Valid	Move segment register to *r/m16*.
REX.W + 8C /r	MOV *r/m64,Sreg***	MR	Valid	Valid	Move zero extended 16-bit segment register to *r/m64*.
8E /r	MOV *Sreg,r/m16***	RM	Valid	Valid	Move *r/m16* to segment register.
REX.W + 8E /r	MOV *Sreg,r/m64***	RM	Valid	Valid	Move *lower 16 bits of r/m64* to segment register.
A0	MOV AL,*moffs8**	FD	Valid	Valid	Move byte at (*seg:offset*) to AL.
REX.W + A0	MOV AL,*moffs8**	FD	Valid	N.E.	Move byte at (*offset*) to AL.
A1	MOV AX,*moffs16**	FD	Valid	Valid	Move word at (*seg:offset*) to AX.
A1	MOV EAX,*moffs32**	FD	Valid	Valid	Move doubleword at (*seg:offset*) to EAX.
REX.W + A1	MOV RAX,*moffs64**	FD	Valid	N.E.	Move quadword at (*offset*) to RAX.

mov

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If an attempt is made to load SS register with NULL segment selector when CPL = 3.
	If an attempt is made to load SS register with NULL segment selector when CPL < 3 and CPL ≠ RPL.
#GP(selector)	If segment selector index is outside descriptor table limits.
	If the memory access to the descriptor table is non-canonical.
	If the SS register is being loaded and the segment selector's RPL and the segment descriptor's DPL are not equal to the CPL.
	If the SS register is being loaded and the segment pointed to is a nonwritable data segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is not a data or readable code segment.
	If the DS, ES, FS, or GS register is being loaded and the segment pointed to is a data or nonconforming code segment, but both the RPL and the CPL are greater than the DPL.
#SS(0)	If the stack address is in a non-canonical form.
#SS(selector)	If the SS register is being loaded and the segment pointed to is marked not present.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If attempt is made to load the CS register.
	If the LOCK prefix is used.

`mov`—What could go wrong?

- lots of exceptions for `mov`

`mov`—What could go wrong?

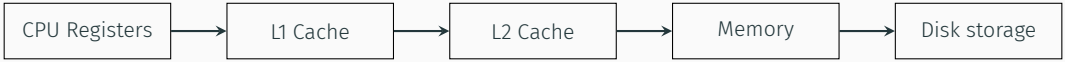
- lots of exceptions for `mov`
- but accessing data loads it to the cache

`mov`—What could go wrong?

- lots of exceptions for `mov`
- but accessing data loads it to the cache

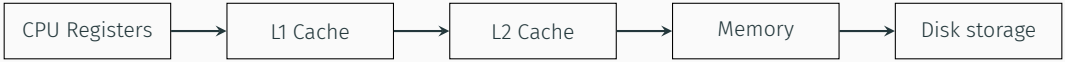
→ side effects on computations!

Memory hierarchy



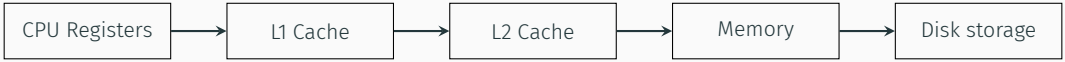
- data can reside in

Memory hierarchy



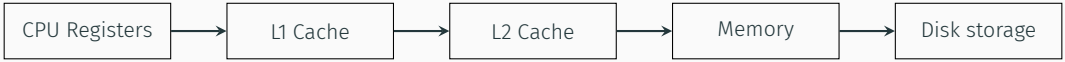
- data can reside in
  - CPU registers

Memory hierarchy



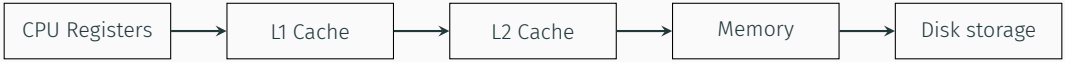
- data can reside in
  - CPU registers
  - different levels of the CPU cache

Memory hierarchy



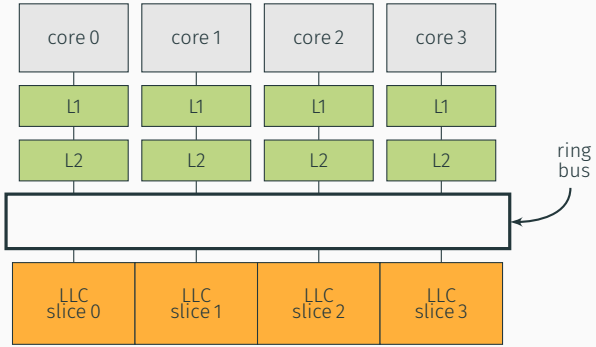
- data can reside in
  - CPU registers
  - different levels of the CPU cache
  - main memory

Memory hierarchy



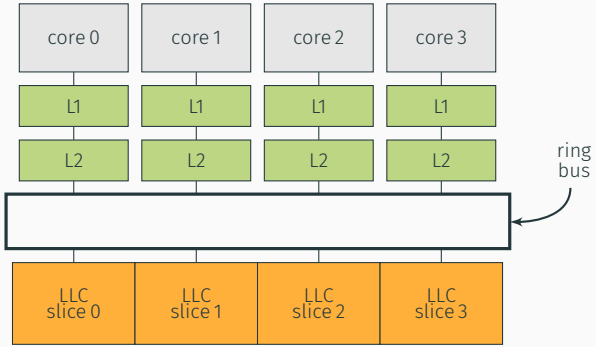
- data can reside in
  - CPU registers
  - different levels of the CPU cache
  - main memory
  - disk storage

Caches on Intel CPUs



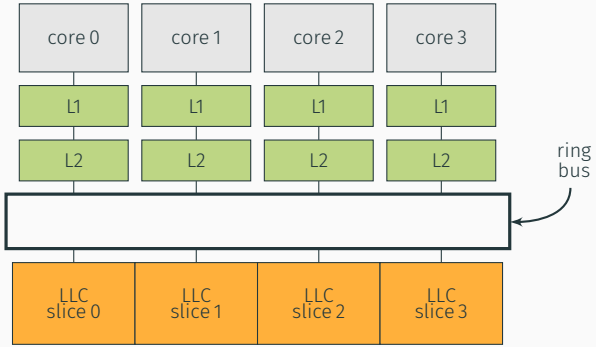
- L1 and L2 are private

Caches on Intel CPUs



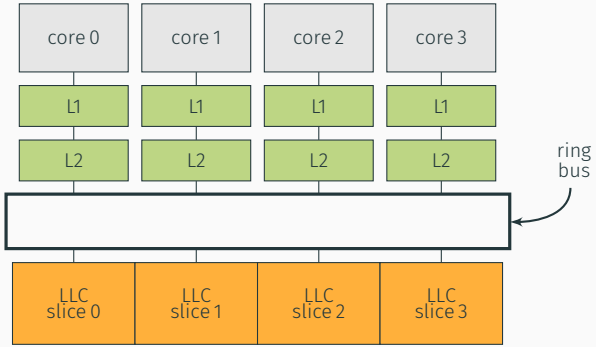
- L1 and L2 are private
- last-level cache

Caches on Intel CPUs



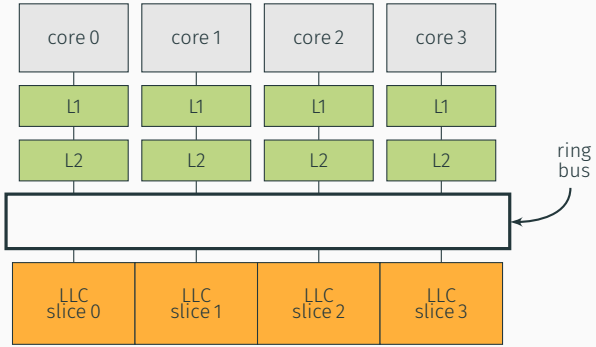
- L1 and L2 are private
- last-level cache
  - divided in slices

Caches on Intel CPUs



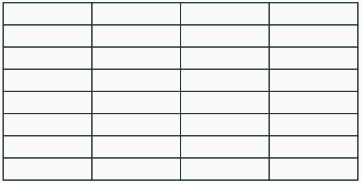
- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores

Caches on Intel CPUs



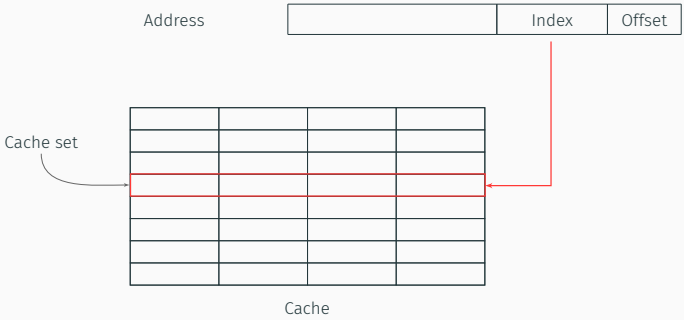
- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive

Set-associative caches



Cache

Set-associative caches



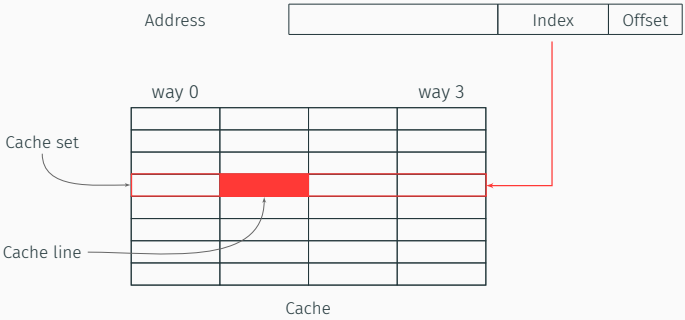
Data loaded in a specific set depending on its address

Set-associative caches

Data loaded in a specific set depending on its address

Several ways per set

Set-associative caches

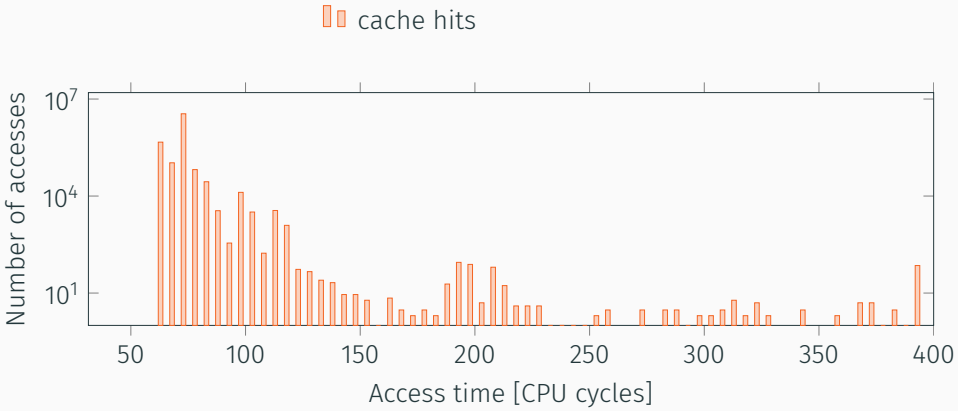


Data loaded in a specific set depending on its address

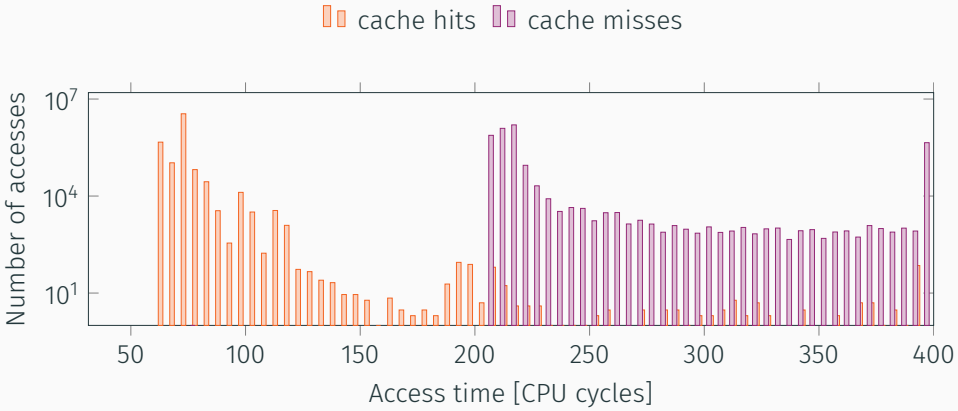
Several ways per set

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

Timing differences



Timing differences



Accurate timings

- very short timings
- `rdtsc` instruction: cycle-accurate timestamps

Accurate timings

- very short timings
- `rdtsc` instruction: cycle-accurate timestamps

```
[...]
rdtsc
function()
rdtsc
[...]
```

Accurate timings

- do you measure what you think you measure?

Accurate timings

- do you measure what you think you measure?
- out-of-order execution

Accurate timings

- do you measure what you think you measure?
- out-of-order execution → what is really executed

```
rdtsc
function()
[...]
rdtsc
```

```
rdtsc
[...]
rdtsc
function()
```

```
rdtsc
rdtsc
function()
[...]
```

Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)

Accurate timings

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

Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures White Paper*, December 2010.

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 $\rightarrow$ 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

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 et al. “Cache Games – Bringing Access-Based Cache Attacks on AES to Practice”. In: *S&P’11*. 2011.

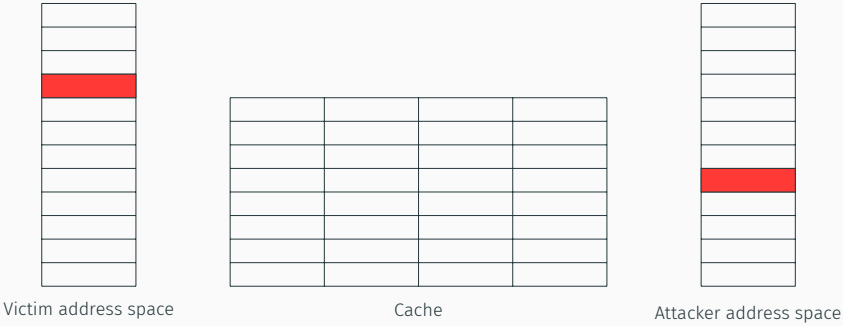
Y. Yarom et al. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014.

D. A. Osvik et al. “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.

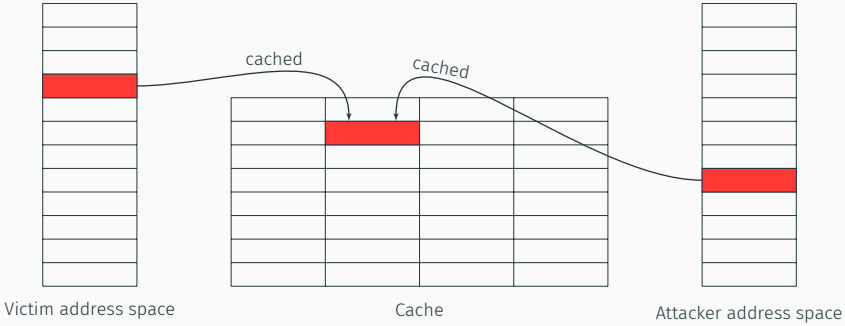
F. Liu et al. “Last-Level Cache Side-Channel Attacks are Practical”. In: *S&P’15*. 2015.

Cache attacks: Flush+Reload



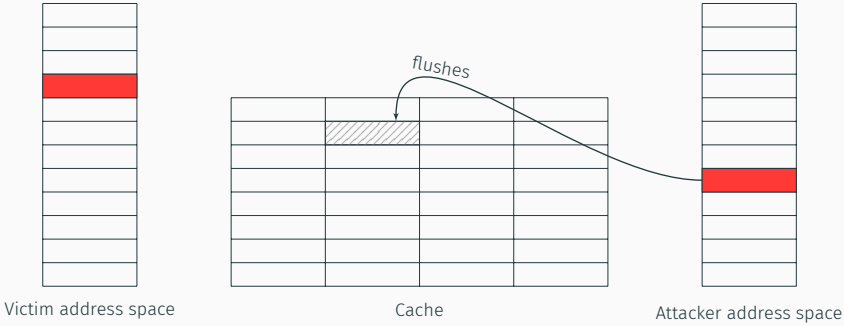
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attacks: Flush+Reload



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

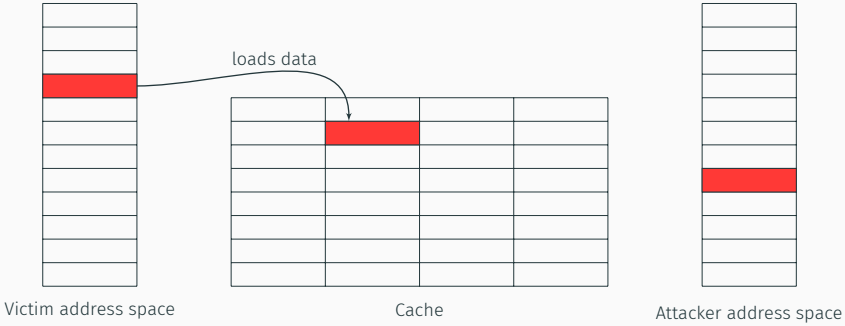
Cache attacks: Flush+Reload



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

Step 2: Attacker flushes the shared cache line

Cache attacks: Flush+Reload

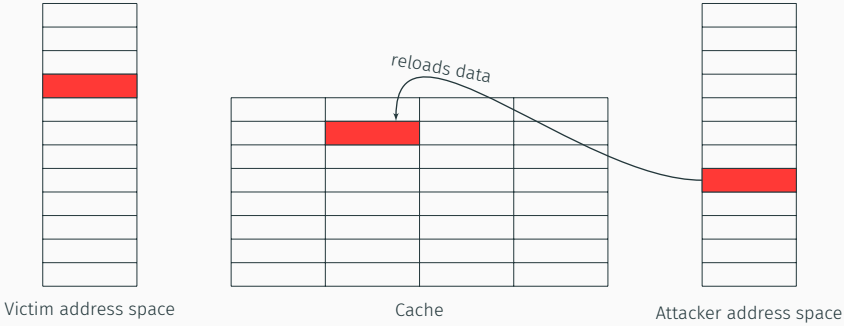


Step 1: Attacker maps shared library (shared memory, in cache)

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)

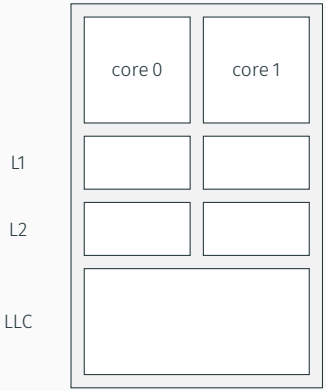
Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Step 4: Attacker reloads the data

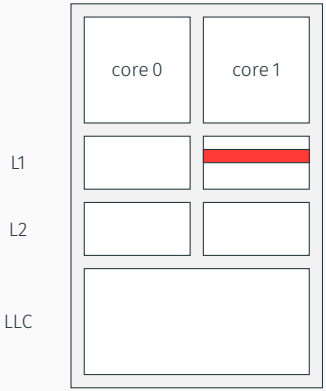
What if there is no shared memory?

Inclusive property



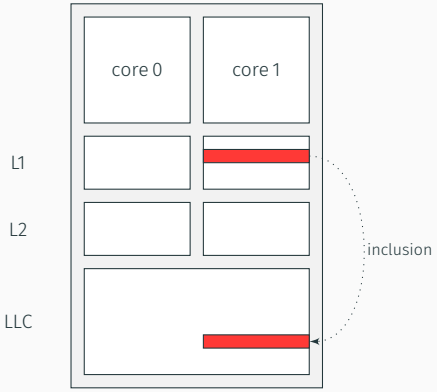
- inclusive LLC: superset of L1 and L2

Inclusive property



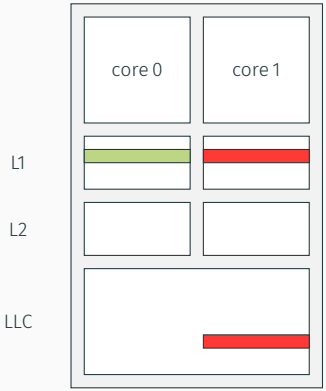
- inclusive LLC: superset of L1 and L2

Inclusive property



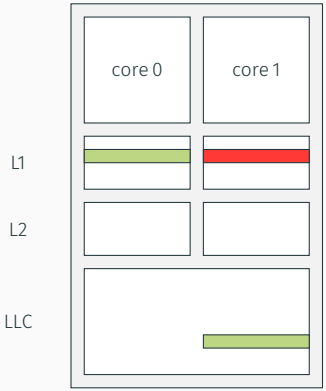
- inclusive LLC: superset of L1 and L2

Inclusive property



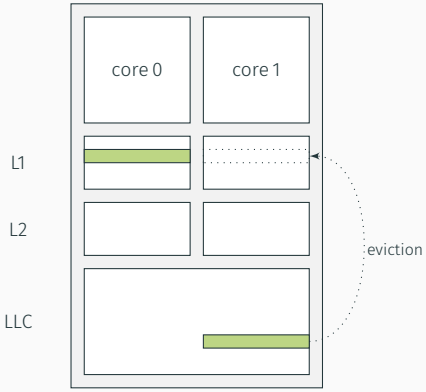
- inclusive LLC: superset of L1 and L2

Inclusive property



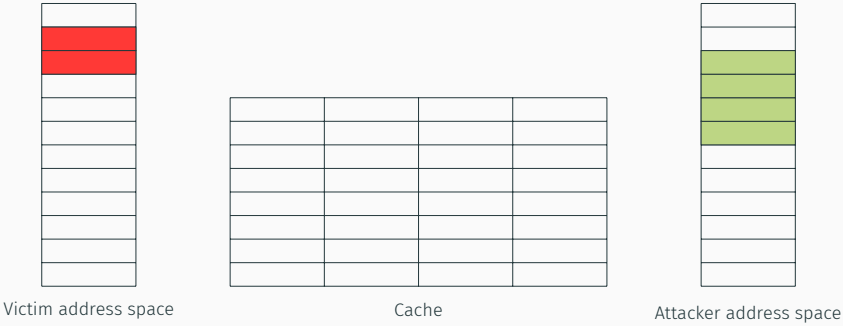
- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2

Inclusive property

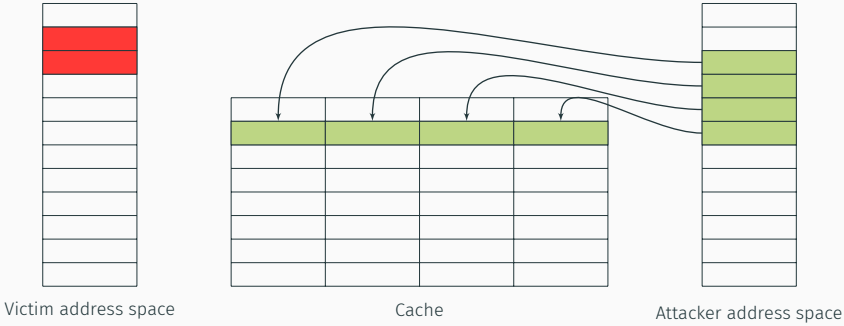


- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
- a core can evict lines in the private L1 of another core

Cache attacks: Prime+Probe

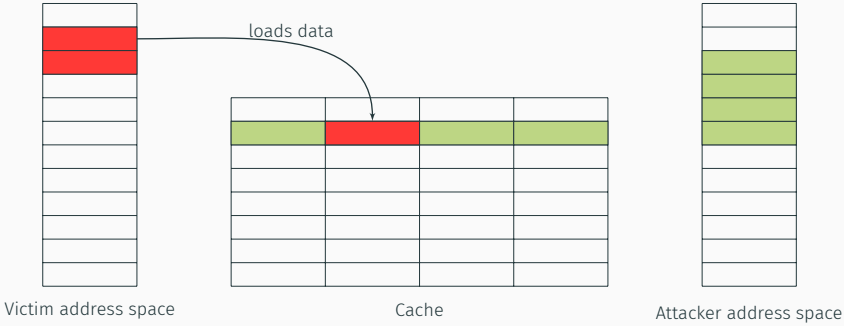


Cache attacks: Prime+Probe



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

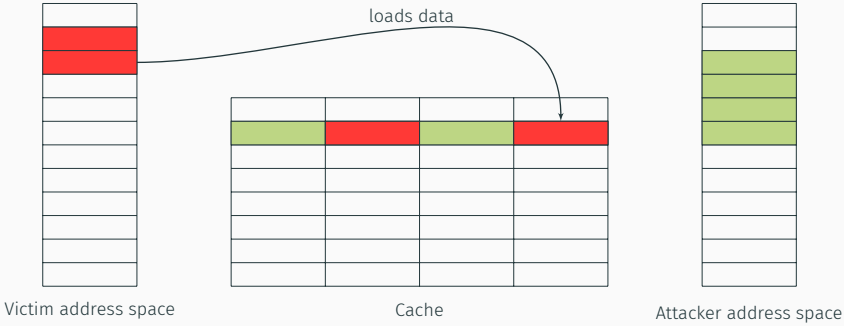
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

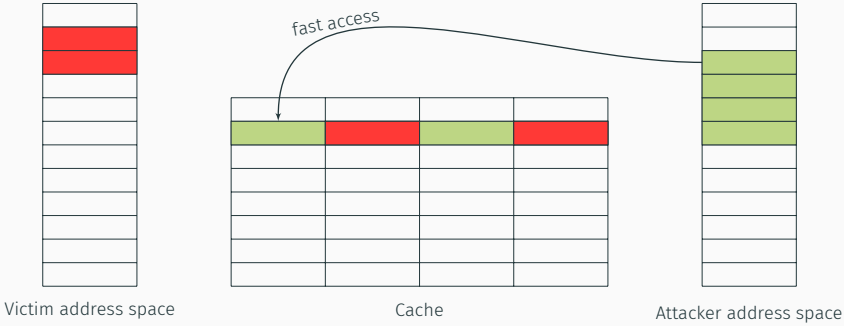
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

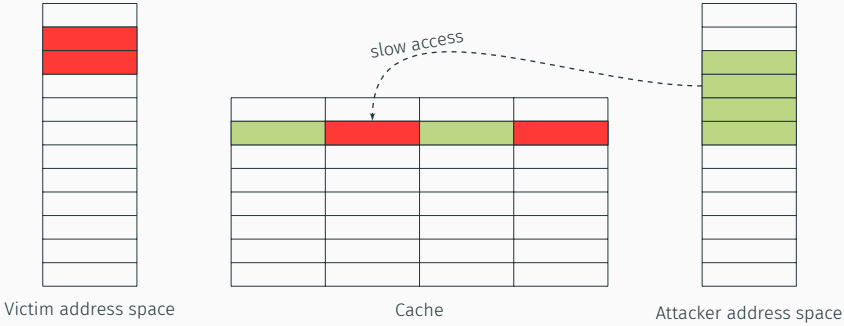


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

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

Challenges with Prime+Probe

We need to evict caches lines without `clflush` or shared memory:

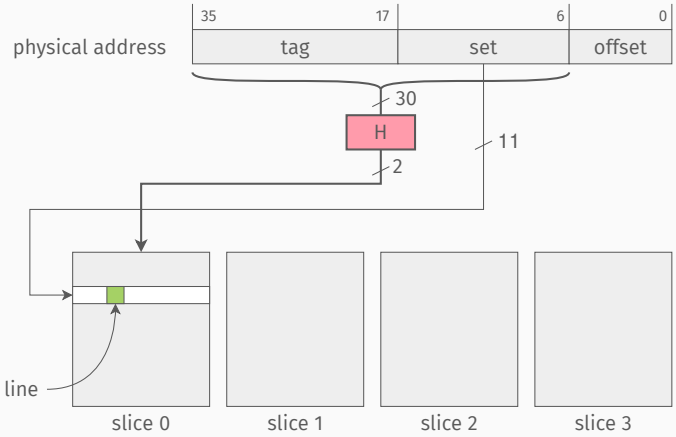
1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?

Challenges with Prime+Probe

We need to evict caches lines without `clflush` or shared memory:

1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?

Last-level cache addressing



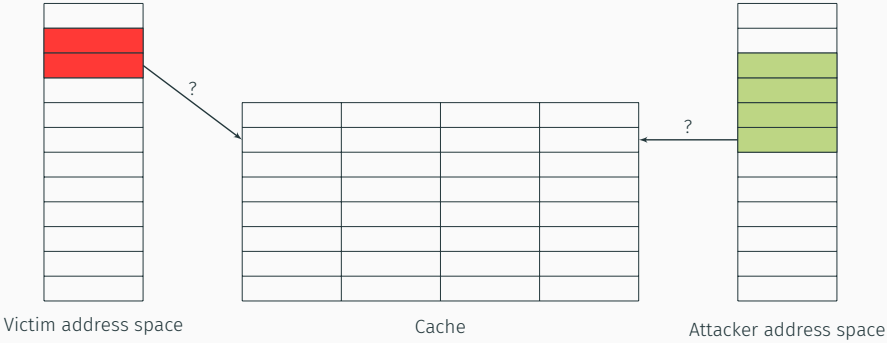
Last-level cache addressing

- last-level cache $\rightarrow$ as many slices as cores
- undocumented hash function that maps a physical address to a slice
- designed for performance

For 2^k slices:

Prime+Probe on recent procesors?

Undocumented function $\rightarrow$ impossible to target the same set in the same slice



Let’s reverse-engineer
the last-level cache!

Reverse engineering method

1. find some way to map one address to one slice

Reverse engineering method

1. find some way to map one address to one slice
2. repeat for every address with a 64B stride

Reverse engineering method

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

How to map addresses to slices?

- with performance counters (Maurice et al., 2015)
- with a timing attack
  - using `clflush` (using Gruss et al., 2016)
  - using memory accesses (Yarom et al., 2015)

C. Maurice et al. “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: *RAID’15*. 2015.

D. Gruss et al. “Flush+Flush: A Fast and Stealthy Cache Attack”. In: *DIMVA’16*. 2016.

Y. Yarom et al. “Mapping the Intel Last-Level Cache”. In: *Cryptology ePrint Archive, Report 2015/905* (2015), pp. 1–12.

How to map addresses to slices?

- with performance counters (Maurice et al., 2015)
- with a timing attack
  - using `clflush` (using Gruss et al., 2016)
  - using memory accesses (Yarom et al., 2015)

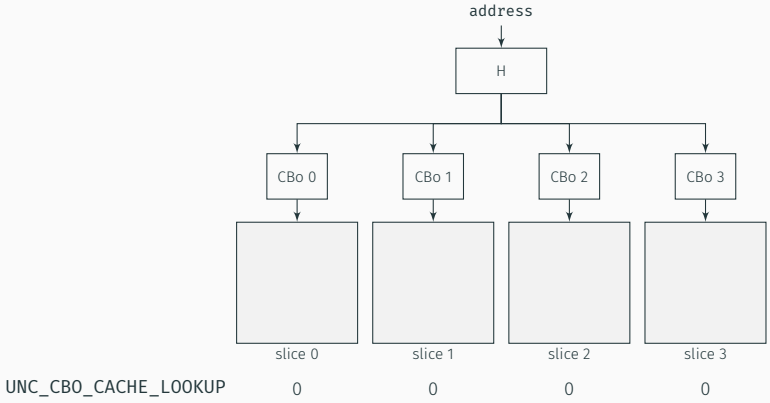
C. Maurice et al. “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: *RAID’15*. 2015.

D. Gruss et al. “Flush+Flush: A Fast and Stealthy Cache Attack”. In: *DIMVA’16*. 2016.

Y. Yarom et al. “Mapping the Intel Last-Level Cache”. In: *Cryptology ePrint Archive, Report 2015/905* (2015), pp. 1–12.

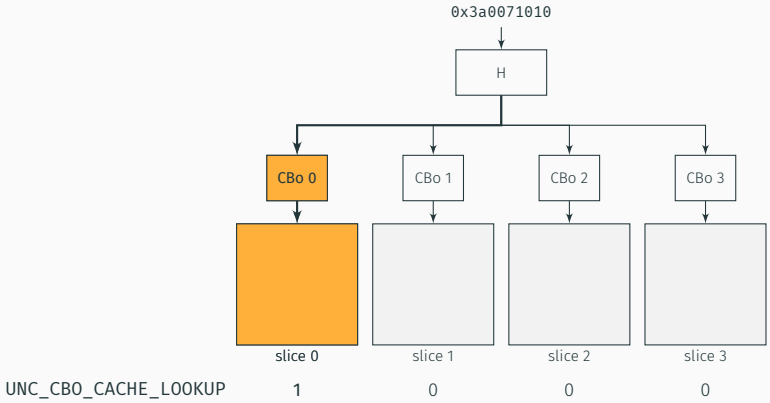
Mapping with performance counters

- event UNC_CBO_CACHE_LOOKUP counts accesses to a slice



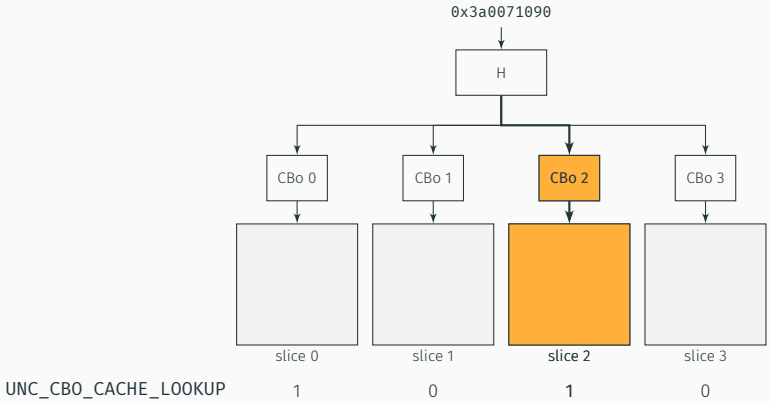
Mapping with performance counters

- event UNC_CBO_CACHE_LOOKUP counts accesses to a slice



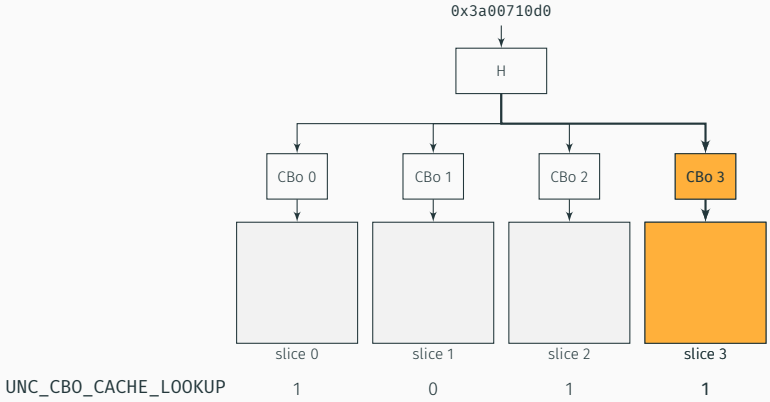
Mapping with performance counters

- event UNC_CBO_CACHE_LOOKUP counts accesses to a slice



Mapping with performance counters

- event UNC_CBO_CACHE_LOOKUP counts accesses to a slice



Mapping a physical address to a slice

1. translate virtual to physical address with `/proc/pid/pagemap`

Mapping a physical address to a slice

1. translate virtual to physical address with `/proc/pid/pagemap`
2. set up monitoring session

Mapping a physical address to a slice

1. translate virtual to physical address with `/proc/pid/pagemap`
2. set up monitoring session
3. repeat access to a single address

Mapping a physical address to a slice

1. translate virtual to physical address with `/proc/pid/pagemap`
2. set up monitoring session
3. repeat access to a single address
   → `clflush` is already counted as an access

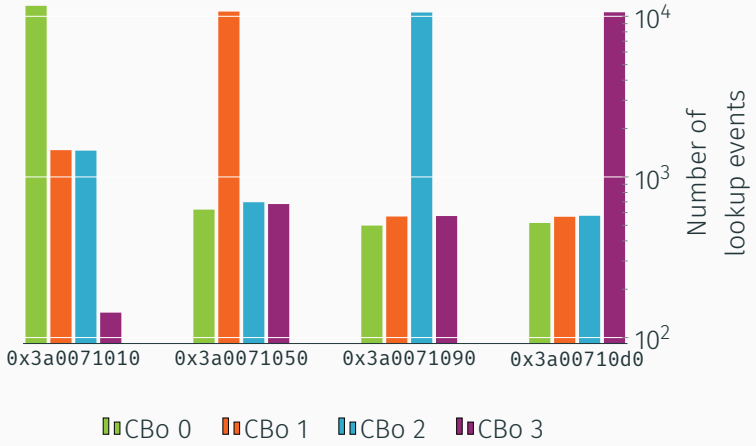
Mapping a physical address to a slice

1. translate virtual to physical address with `/proc/pid/pagemap`
2. set up monitoring session
3. repeat access to a single address
   → `clflush` is already counted as an access
4. read `UNC_CBO_CACHE_LOOKUP` event for each CBo

Mapping a physical address to a slice

1. translate virtual to physical address with `/proc/pid/pagemap`
2. set up monitoring session
3. repeat access to a single address
   → `clflush` is already counted as an access
4. read `UNC_CBO_CACHE_LOOKUP` event for each CBo
5. slice is the one that has the maximum lookup events

Mapping physical addresses to slices



Inferring the function

Two cases:

1. linear function (2^n number of cores): XOR of address bits

Inferring the function

Two cases:

1. linear function (2^n number of cores): XOR of address bits
    - solve equations with linear algebra

Inferring the function

Two cases:

1. linear function (2^n number of cores): XOR of address bits
   - solve equations with linear algebra
   - brute force

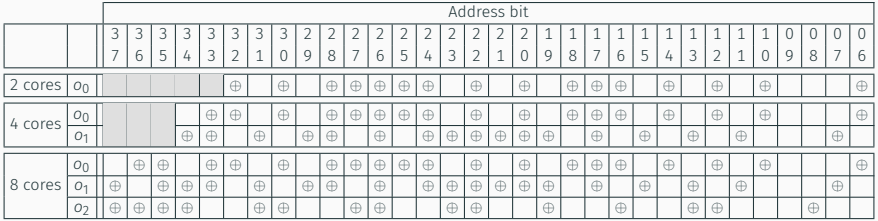
Inferring the function

Two cases:

1. linear function (2^n number of cores): XOR of address bits
   - solve equations with linear algebra
   - brute force
2. non-linear function (the rest): more complicated

Last-level cache linear functions

3 functions, depending on the number of cores



Function valid for Sandy Bridge, Ivy Bridge, Haswell, Broadwell

#1. Cache Covert Channels

Covert channel

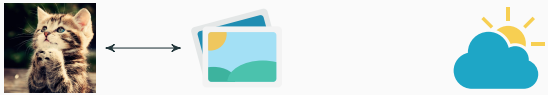
- malicious privacy gallery app

Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images

Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget

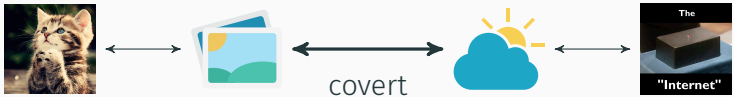


Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
  - no permissions except accessing the Internet

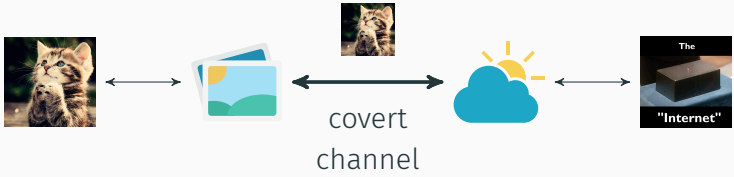
Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
  - no permissions except accessing the Internet



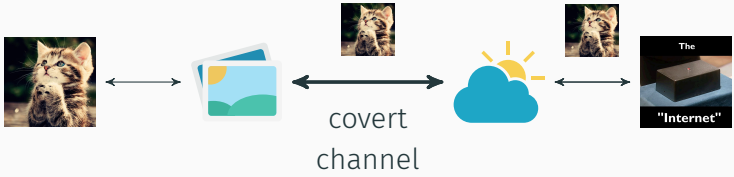
Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
  - no permissions except accessing the Internet



Covert channel

- malicious privacy gallery app
  - no permissions except accessing your images
- malicious weather widget
  - no permissions except accessing the Internet



General principle

- sender transmits bits with cache hits and misses
- receiver monitors a cache line or a cache set to receive 1 bit

General principle

- sender transmits bits with cache hits and misses
- receiver monitors a cache line or a cache set to receive 1 bit
- transmitting on several lines or sets to send several bits

General principle

- sender transmits bits with cache hits and misses
- receiver monitors a cache line or a cache set to receive 1 bit
- transmitting on several lines or sets to send several bits
- Flush+Reload: using cache line from a shared library/executable
  - sender: accesses the line to send '1', stays idle to send '0'
  - receiver: reloads the line, then flushes it

General principle

- sender transmits bits with cache hits and misses
- receiver monitors a cache line or a cache set to receive 1 bit
- transmitting on several lines or sets to send several bits
- Flush+Reload: using cache line from a shared library/executable
  - sender: accesses the line to send '1', stays idle to send '0'
  - receiver: reloads the line, then flushes it
- Prime+Probe: agreeing on a cache set
  - sender: primes the set to send '1', stays idle to send '0'
  - receiver: probes the set

Issues

- programs competing for
  - the CPU cache
  - scheduling

C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: *NDSS’17*. 2017.

Issues

- programs competing for
  - the CPU cache
  - scheduling
- literature: covert channels stop working with noise on the machine

C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: *NDSS’17*. 2017.

Issues

- programs competing for
  - the CPU cache
  - scheduling
- literature: covert channels stop working with noise on the machine
- solution? “Just use error-correcting codes”

C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: *NDSS’17*. 2017.

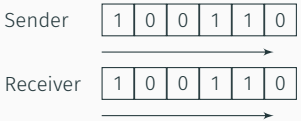
Issues

- programs competing for
  - the CPU cache
  - scheduling
- literature: covert channels stop working with noise on the machine
- solution? “Just use error-correcting codes”

- let’s investigate this!

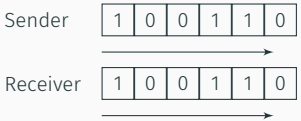
C. Maurice et al. “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: *NDSS’17*. 2017.

Why can’t we just use error correcting codes?

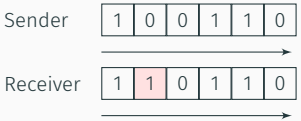


(a) Transmission without errors

Why can’t we just use error correcting codes?

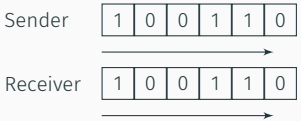


(a) Transmission without errors

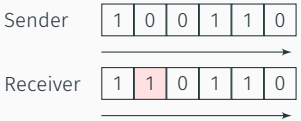


(b) Noise: substitution error

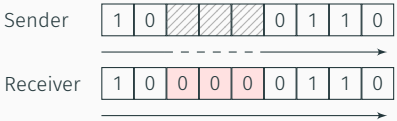
Why can't we just use error correcting codes?



(a) Transmission without errors

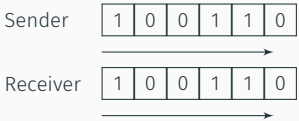


(b) Noise: substitution error

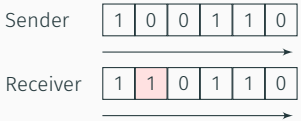


(c) Sender descheduled: insertions

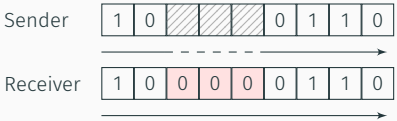
Why can't we just use error correcting codes?



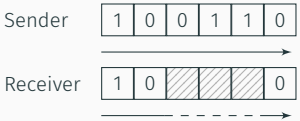
(a) Transmission without errors



(b) Noise: substitution error



(c) Sender descheduled: insertions



(d) Receiver descheduled: deletions

Our robust covert channel

- physical layer:
  - transmits words as a sequence of ‘0’s and ‘1’s
  - deals with synchronization errors
- data-link layer:
  - divides data to transmit into packets
  - corrects the remaining errors

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
- sender transmits ’0’ doing nothing
  → lines of the receiver still in cache → fast access

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
- sender transmits ’0’ doing nothing
  → lines of the receiver still in cache → fast access
- sender transmits ’1’ accessing addresses in the set
  → evicts lines of the receiver → slow access

Eviction set generation

- need a set of addresses in the same cache set and same slice

Eviction set generation

- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address

Eviction set generation

- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address

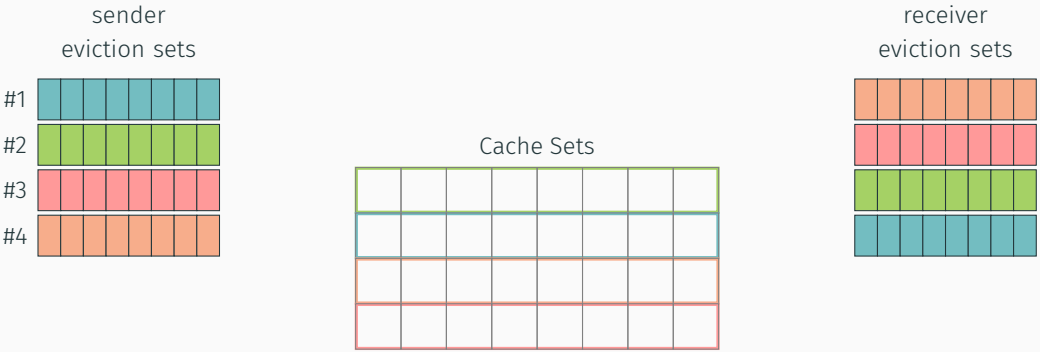
- we can build a set of addresses in the same cache set and same slice

Eviction set generation

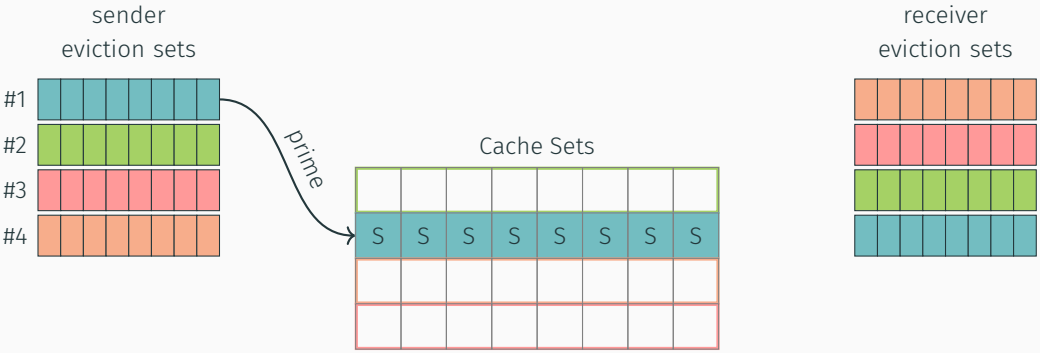
- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address

- we can build a set of addresses in the same cache set and same slice
- without knowing which slice

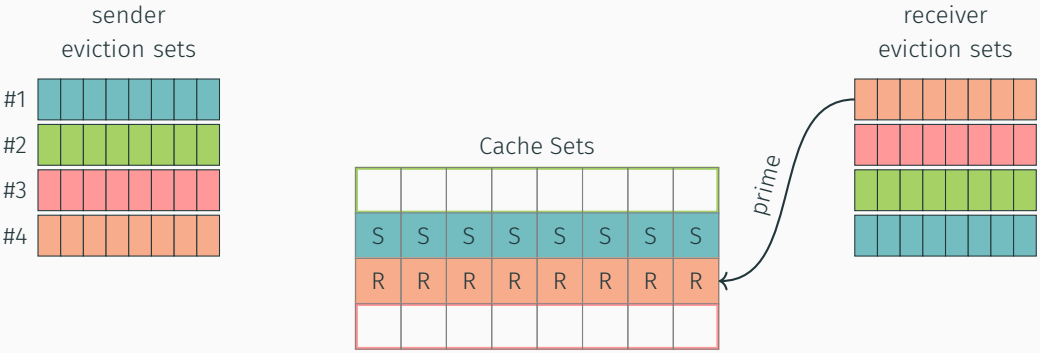
Jamming agreement



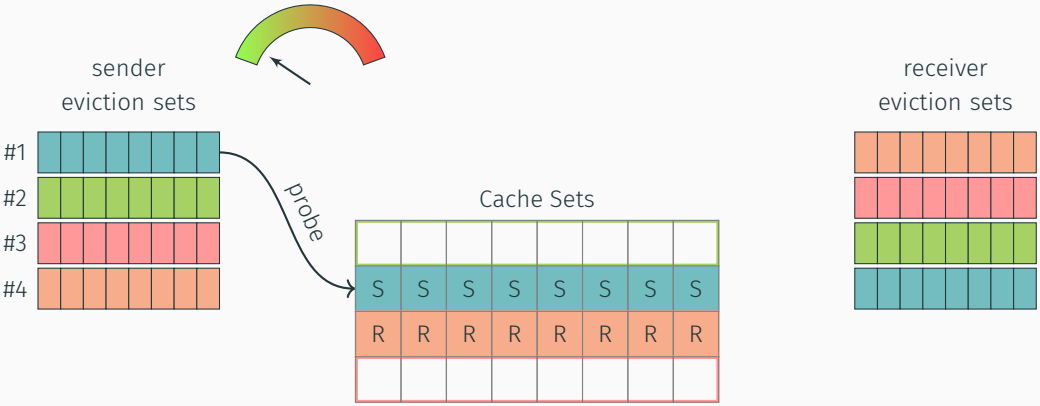
Jamming agreement



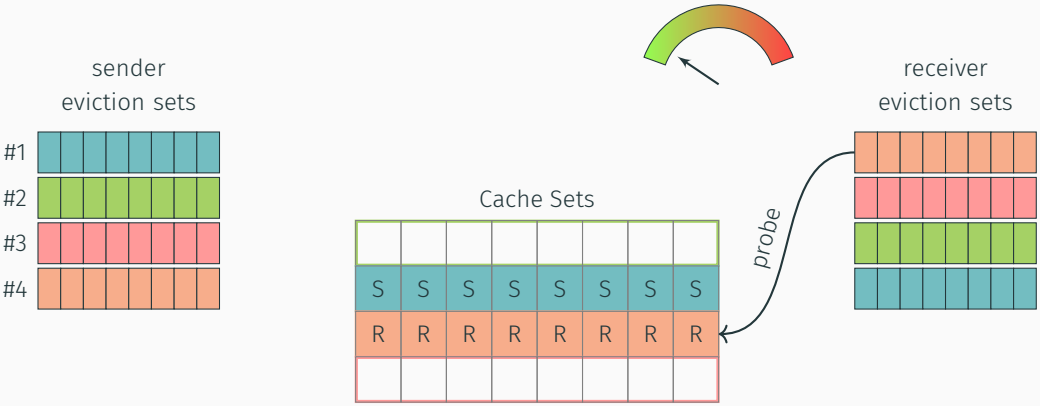
Jamming agreement



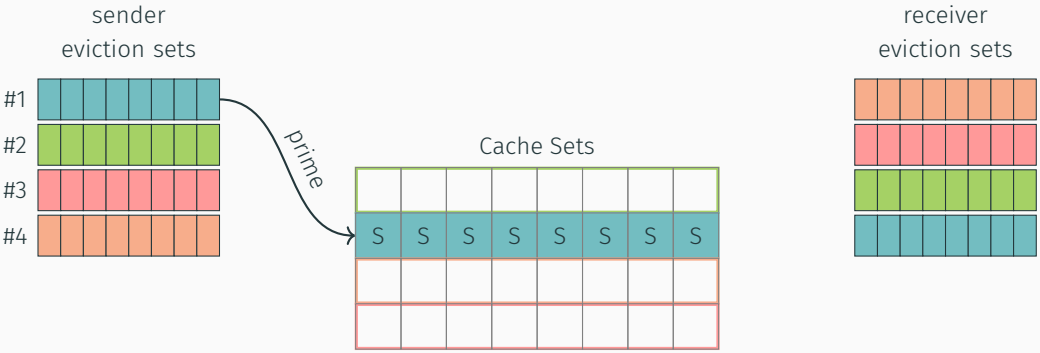
Jamming agreement



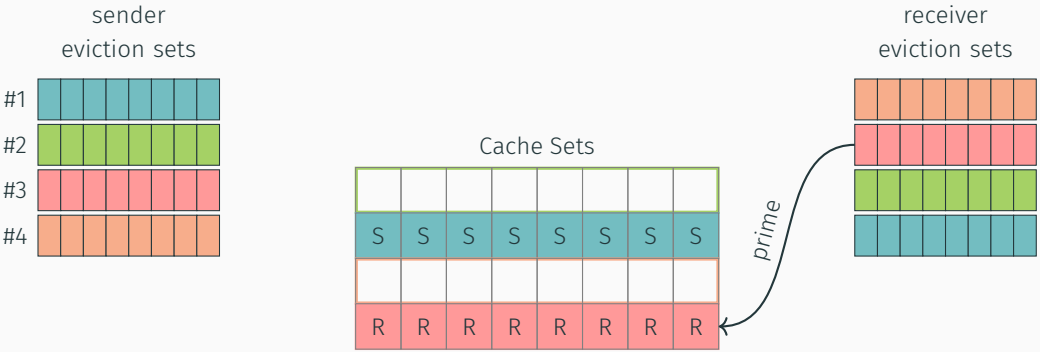
Jamming agreement



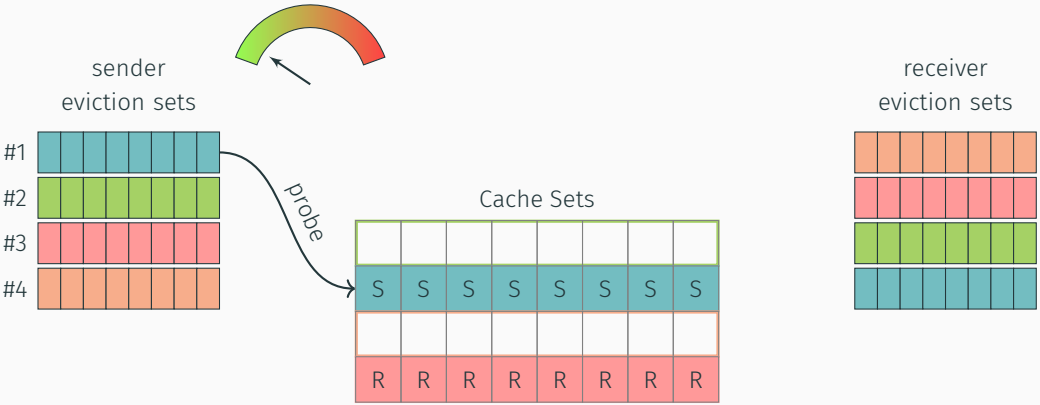
Jamming agreement



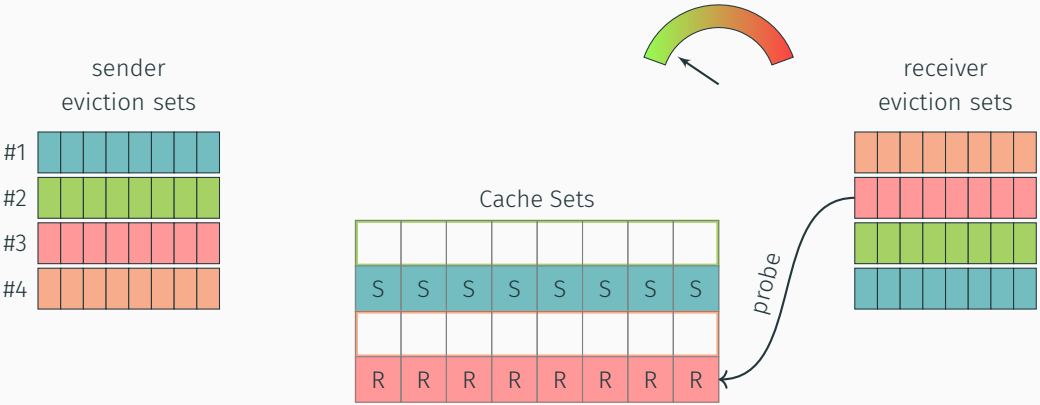
Jamming agreement



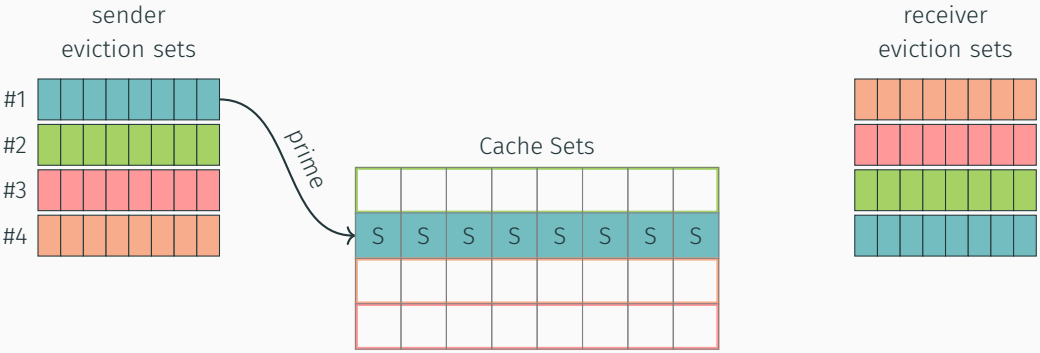
Jamming agreement



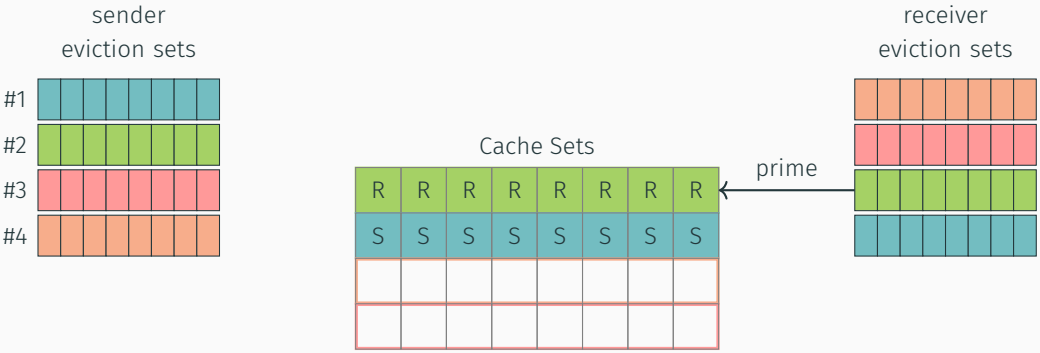
Jamming agreement



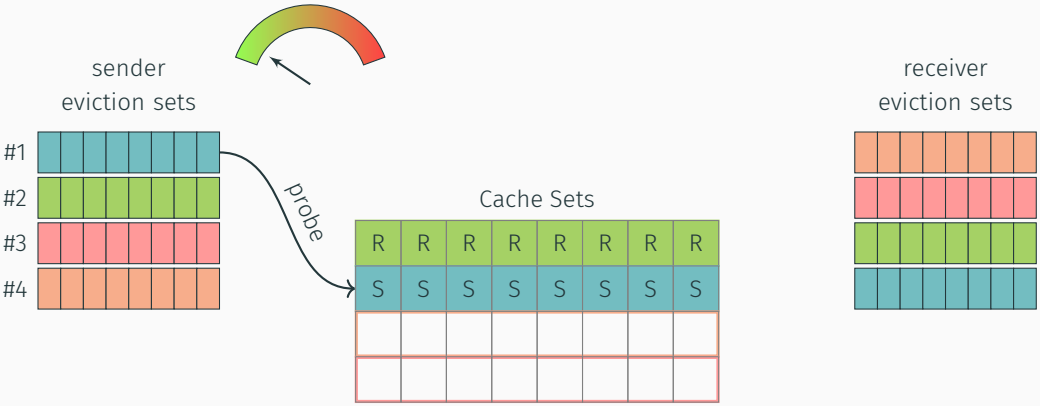
Jamming agreement



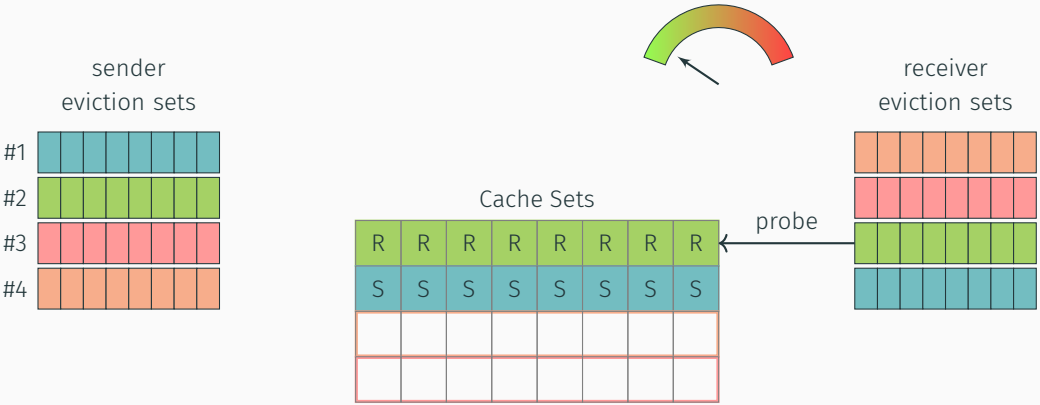
Jamming agreement



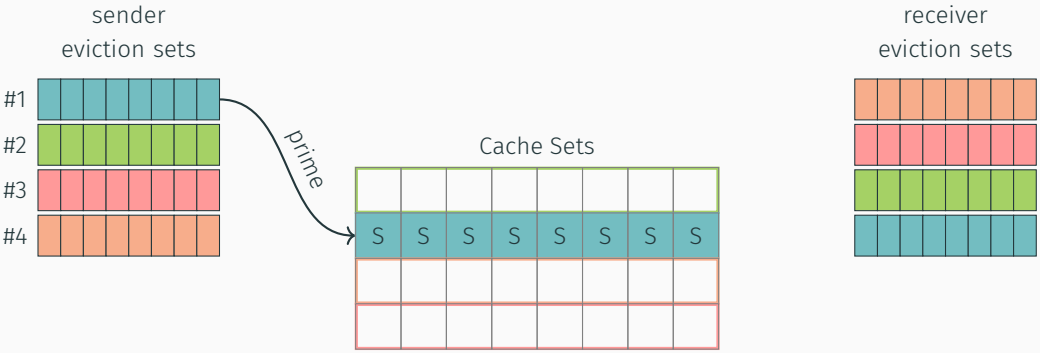
Jamming agreement



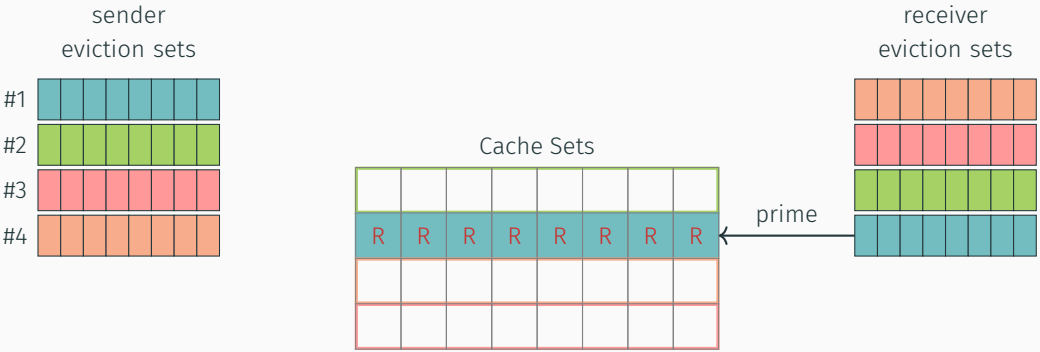
Jamming agreement



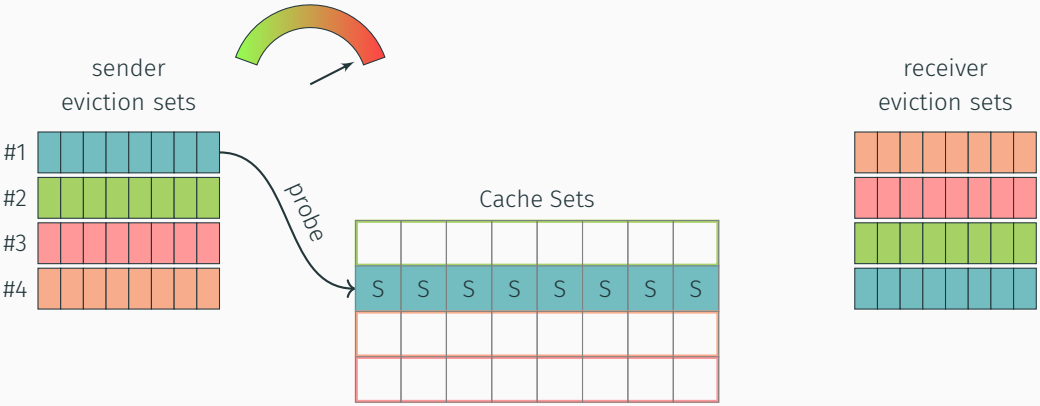
Jamming agreement



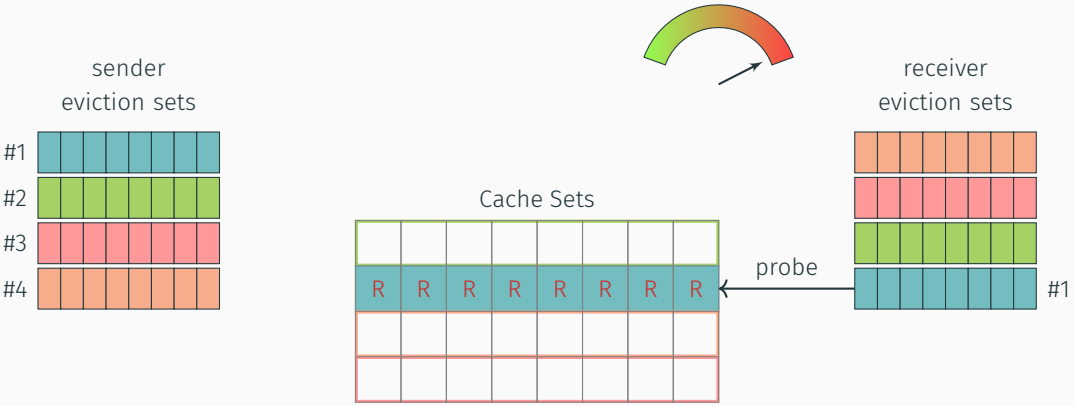
Jamming agreement



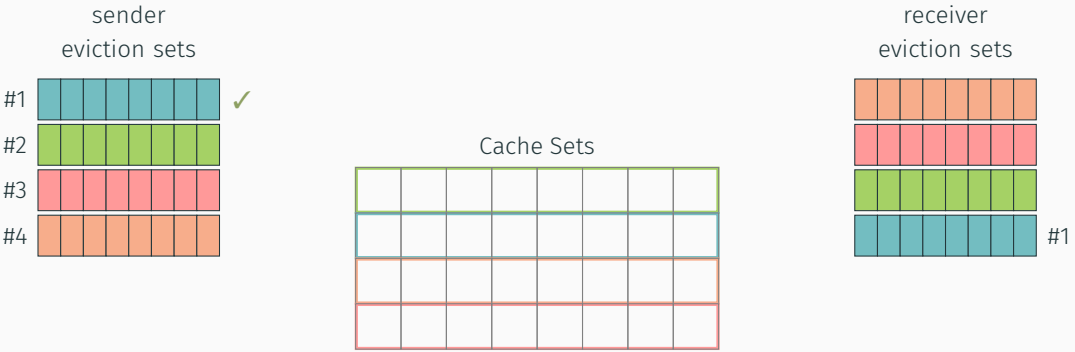
Jamming agreement



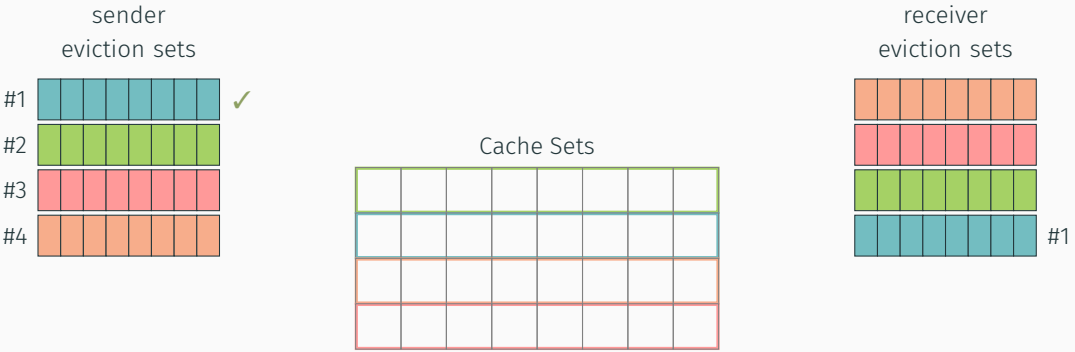
Jamming agreement



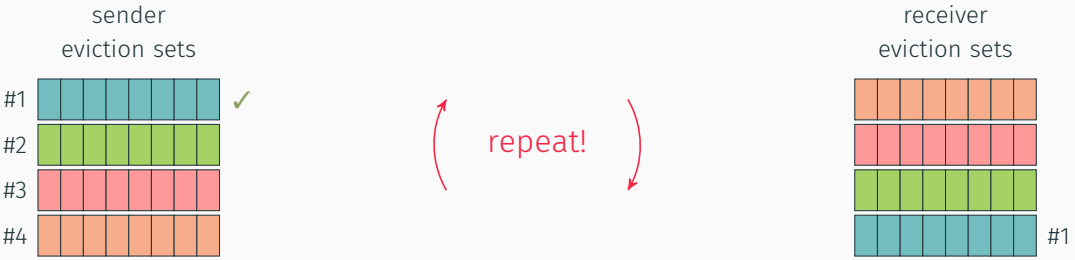
Jamming agreement



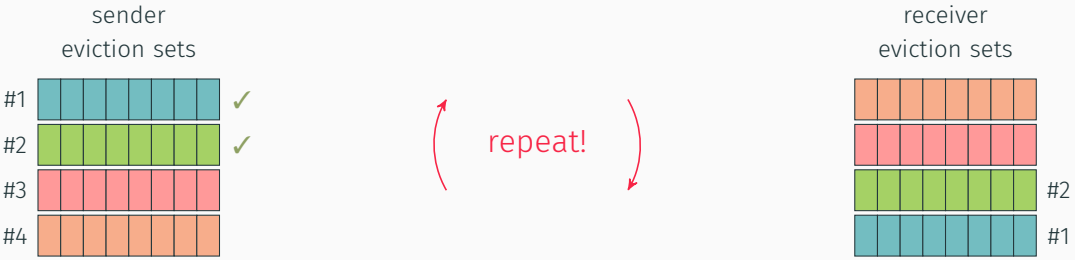
Jamming agreement



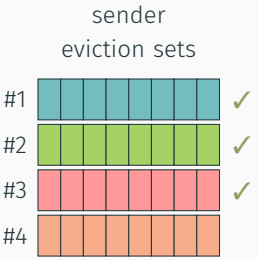
Jamming agreement

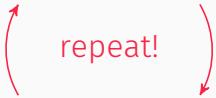


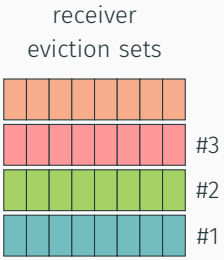
Jamming agreement



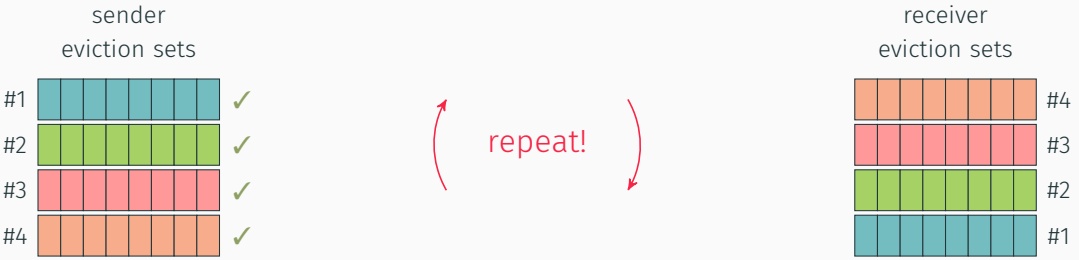
Jamming agreement



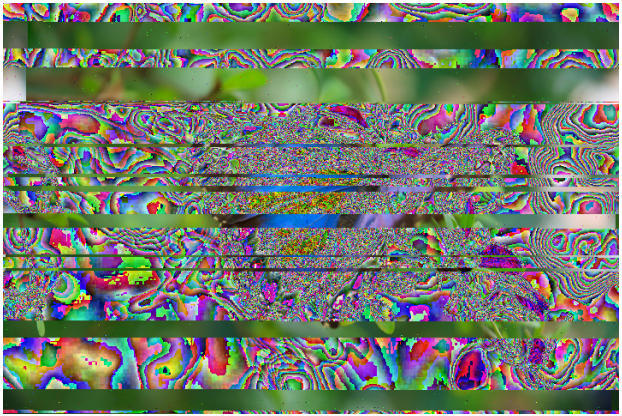




Jamming agreement



Sending the first image



Handling synchronization errors

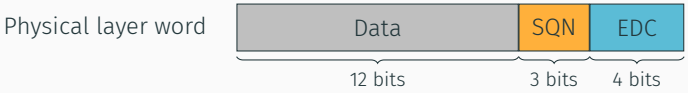
Handling synchronization errors

- deletion errors: request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)

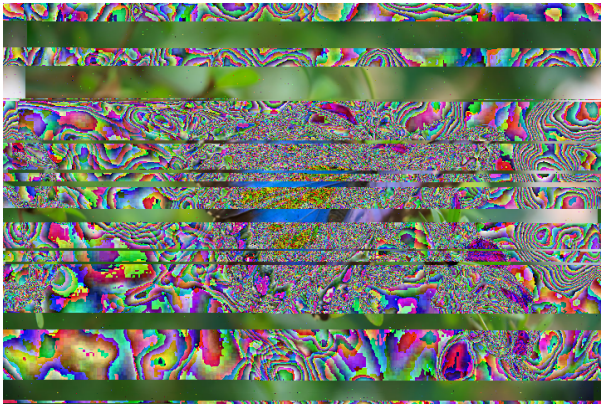
Handling synchronization errors

- deletion errors: request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)
- '0'-insertion errors: error detection code → Berger codes
  - appending the number of '0's in the word to itself

  → property: a word cannot consist solely of '0's



Synchronization (before)



Synchronization (after)

Synchronization (after)

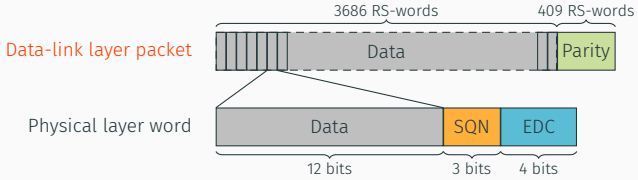
Synchronization (after)

Data-link layer: Error correction

- Reed-Solomon codes to correct the remaining errors

Data-link layer: Error correction

- Reed-Solomon codes to correct the remaining errors
- RS word size = physical layer word size = 12 bits
- packet size = $2^{12} - 1 = 4095$ RS words
- 10% error-correcting code: 409 parity and 3686 data RS words



Error correction (after)

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`

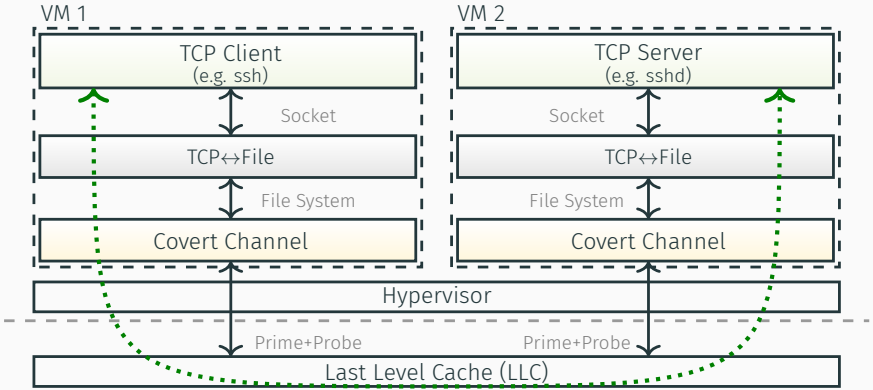
Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`
Amazon EC2	45.25 KBps	0.00%	–

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`
Amazon EC2	45.25 KBps	0.00%	–
Amazon EC2	45.09 KBps	0.00%	web server serving files on sender VM
Amazon EC2	42.96 KBps	0.00%	`stress -m 2` on sender VM
Amazon EC2	42.26 KBps	0.00%	`stress -m 1` on receiver VM
Amazon EC2	37.42 KBps	0.00%	web server on all 3 VMs, `stress -m 4` on 3rd VM, `stress -m 1` on sender and receiver VMs
Amazon EC2	34.27 KBps	0.00%	`stress -m 8` on third VM

Building an SSH connection



SSH evaluation

Between two instances on Amazon EC2

Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓
Web server on third VM	✓
Web server on SSH server VM	✓
Web server on all VMs	✓
`stress -m 1` on server side	unstable

SSH evaluation

Between two instances on Amazon EC2

Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓
Web server on third VM	✓
Web server on SSH server VM	✓
Web server on all VMs	✓
`stress -m 1` on server side	unstable

Telnet also works with occasional corrupted bytes with `stress -m 1`

Cache covert channels: Take-away

- cache covert channels are practical

Cache covert channels: Take-away

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise

Cache covert channels: Take-away

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection

Cache covert channels: Take-away

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection
- we extended Amazon's product portfolio :)

Cache covert channels: Take-away

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection
- we extended Amazon's product portfolio :)

Cache covert channels: Take-away

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection
- we extended Amazon's product portfolio :)

#2. Side-Channel Attack on Crypto

Side-channel attack on AES T-Tables implementation

- AES T-Tables: fast software implementation

D. A. Osvik et al. “Cache Attacks and Countermeasures: the Case of AES”. In: *CT-RSA 2006*. 2006.

Side-channel attack on AES T-Tables implementation

- AES T-Tables: fast software implementation
- uses precomputed look-up tables

D. A. Osvik et al. “Cache Attacks and Countermeasures: the Case of AES”. In: *CT-RSA 2006*. 2006.

Side-channel attack on AES T-Tables implementation

- AES T-Tables: fast software implementation
- uses precomputed look-up tables
- one-round known-plaintext attack by Osvik et al.
  - p plaintext and k secret key
  - intermediate state $x^{(r)} = (x_0^{(r)}, \ldots, x_{15}^{(r)})$ at each round r
  - first round, accessed table indices are

$$x_i^{(0)} = p_i \oplus k_i \qquad \text{for all } i = 0, \ldots, 15$$

D. A. Osvik et al. "Cache Attacks and Countermeasures: the Case of AES". In: *CT-RSA 2006*. 2006.

Side-channel attack on AES T-Tables implementation

- AES T-Tables: fast software implementation
- uses precomputed look-up tables
- one-round known-plaintext attack by Osvik et al.
  - p plaintext and k secret key
  - intermediate state $x^{(r)} = (x_0^{(r)}, \ldots, x_{15}^{(r)})$ at each round r
  - first round, accessed table indices are

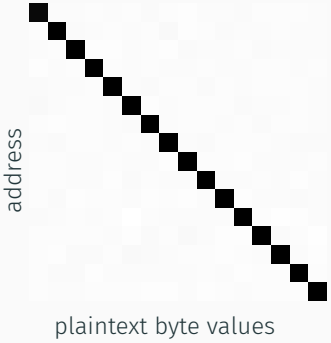
$$x_i^{(0)} = p_i \oplus k_i \qquad \text{for all } i = 0, \ldots, 15$$

$\rightarrow$ recovering accessed table indices $\Rightarrow$ recovering the key

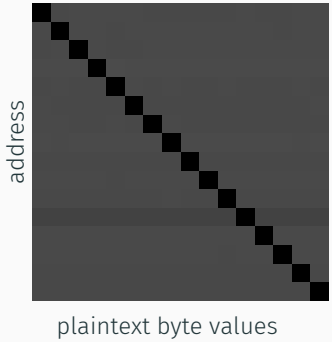
D. A. Osvik et al. "Cache Attacks and Countermeasures: the Case of AES". In: *CT-RSA 2006*. 2006.

Side-channel attack on AES T-Tables implementation

- monitoring which T-Table entry is accessed ($k_0 = \texttt{0x00}$)



Flush+Reload



Prime+Probe

Side-channel attack on AES T-Tables implementation

- it's an old attack...

M. Lipp et al. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016.

Side-channel attack on AES T-Tables implementation

- it's an old attack...
- everything should be fixed by now...

M. Lipp et al. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016.

Side-channel attack on AES T-Tables implementation

- it's an old attack...
- everything should be fixed by now...
- Bouncy Castle on Android $\rightarrow$ default implementation uses T-Tables

M. Lipp et al. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016.

More attacks

- Flush+Reload
  - RSA: 96.7% of secret key bits in a single signature (Yarom et al.)
  - AES: full key recovery in 30000 dec. (a few seconds) (Guelmezoglu et al.)
- Prime+Probe
  - El Gamal (sliding window): full key recovery in 12 min. (Liu et al.)

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

B. Gülmezoglu et al. “A Faster and More Realistic Flush+Reload Attack on AES”. In: *COSADE'15*. 2015

F. Liu et al. “Last-Level Cache Side-Channel Attacks are Practical”. In: *S&P'15*. 2015.

Side-channel attack on crypto: Take-away

- you should never write crypto with secret-dependent memory accesses

Side-channel attack on crypto: Take-away

- you should never write crypto with secret-dependent memory accesses
- most recent implementations are now protected against these attacks
- e.g., really hard to use AES T-Tables by mistake on OpenSSL

Side-channel attack on crypto: Take-away

- you should never write crypto with secret-dependent memory accesses
- most recent implementations are now protected against these attacks
- e.g., really hard to use AES T-Tables by mistake on OpenSSL
- big issue is adoption and legacy code still running

#3. Side-Channel Attack on Keystroke Timings

Cache Template Attacks

- locating event-dependent memory access $\rightarrow$ Cache Template Attacks

D. Gruss et al. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015.

Cache Template Attacks

- locating event-dependent memory access → Cache Template Attacks
- learning phase

D. Gruss et al. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015.

Cache Template Attacks

- locating event-dependent memory access → Cache Template Attacks
- learning phase
    1. shared library or executable is mapped

D. Gruss et al. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015.

Cache Template Attacks

- locating event-dependent memory access → Cache Template Attacks
- learning phase
  1. shared library or executable is mapped
  2. trigger an event while Flush+Reload one address

D. Gruss et al. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015.

Cache Template Attacks

- locating event-dependent memory access → Cache Template Attacks
- learning phase
  1. shared library or executable is mapped
  2. trigger an event while Flush+Reload one address
     → cache hit: address used by the library/executable

D. Gruss et al. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015.

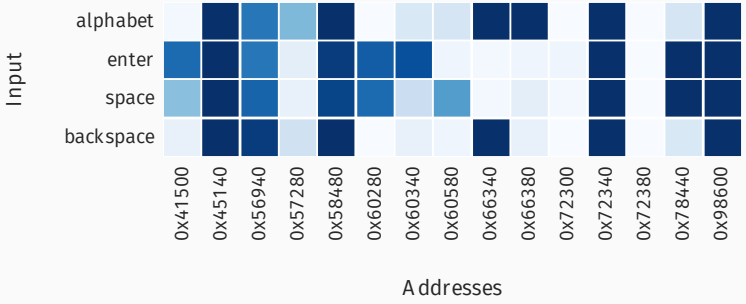
Cache Template Attacks

- locating event-dependent memory access → Cache Template Attacks
- learning phase
  1. shared library or executable is mapped
  2. trigger an event while Flush+Reload one address
     → cache hit: address used by the library/executable
  3. repeat step 2 for every address

D. Gruss et al. "Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches". In: *USENIX Security Symposium*. 2015.

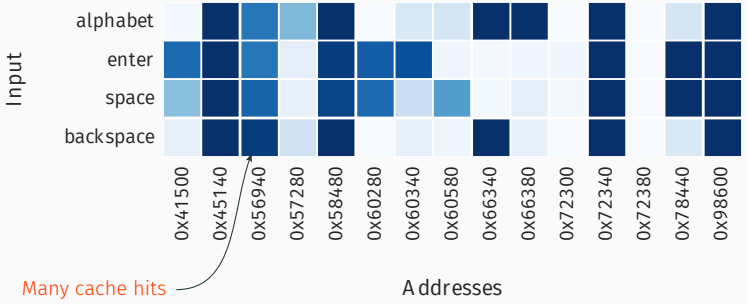
Which addresses to monitor?

- cache template matrix
  = how many cache hits for each pair (event, address)?



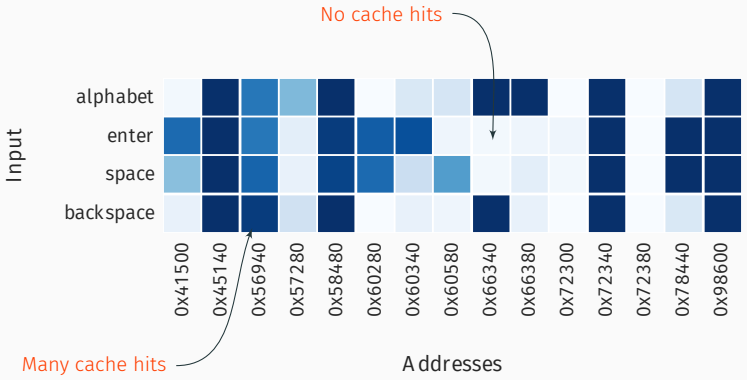
Which addresses to monitor?

- cache template matrix
  = how many cache hits for each pair (event, address)?



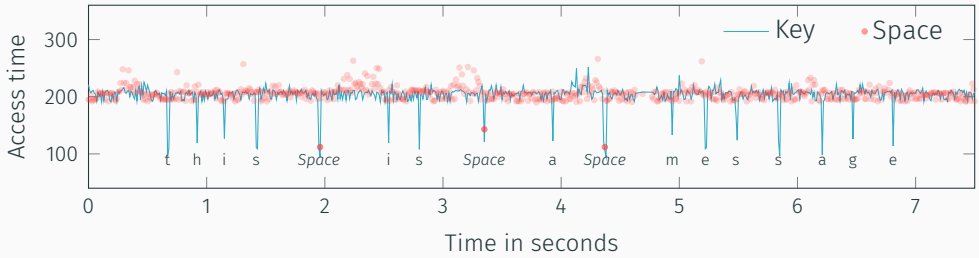
Which addresses to monitor?

- cache template matrix
  = how many cache hits for each pair (event, address)?



Spying on keystrokes

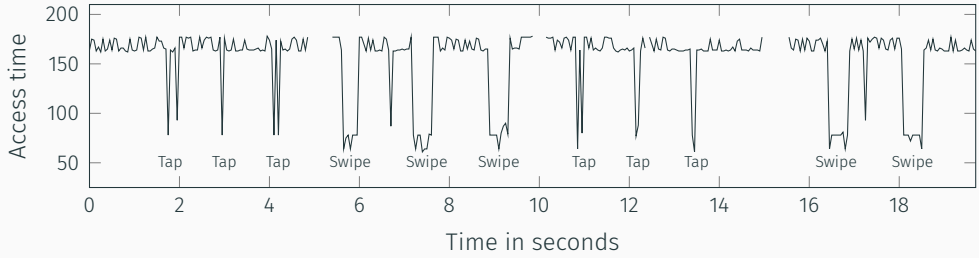
- Flush+Reload and Evict+Reload: fine-grained attacks → spy on keystrokes
- high-resolution timers → precise inter-keystroke timing
- next step: infer typed words with Hidden Markov Models



M. Lipp et al. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX Security Symposium*. 2016

Differentiating events

- distinguishing between different types of events by monitoring access time



M. Lipp et al. “ARMageddon: Cache Attacks on Mobile Devices”. In: *USENIX Security Symposium*. 2016.

Side-channel attack on keystrokes: Take-away

- obtaining precise timings is easy

Side-channel attack on keystrokes: Take-away

- obtaining precise timings is easy
- further computations are needed to derive typed words

Side-channel attack on keystrokes: Take-away

- obtaining precise timings is easy
- further computations are needed to derive typed words
- but the attack also allows distinguishing key groups

Side-channel attack on keystrokes: Take-away

- obtaining precise timings is easy
- further computations are needed to derive typed words
- but the attack also allows distinguishing key groups

→ reduces search space for, e.g., password retrieval

#4. Side-Channel Attack on KASLR

`prefetch` instructions

`prefetch` fetches the line of data from memory containing the specified byte

6 `prefetch` instructions:

- `prefetcht0`: suggests CPU to load data into L1
- `prefetcht1`: suggests CPU to load data into L2
- `prefetcht2`: suggests CPU to load data into L3
- `prefetchnta`: suggests CPU to load data for non-temporal access
- `prefetchw`: suggests CPU to load data with intention to write
- `prefetchwt1`: suggests CPU to load vector data with intention to write

`prefetch` according to Intel

NOTE

Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. 2014

`prefetch` according to Intel

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache.

Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. 2014

`prefetch` according to Intel

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context.

Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. 2014

`prefetch` according to Intel

NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty.

Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. 2014

`prefetch` according to Intel

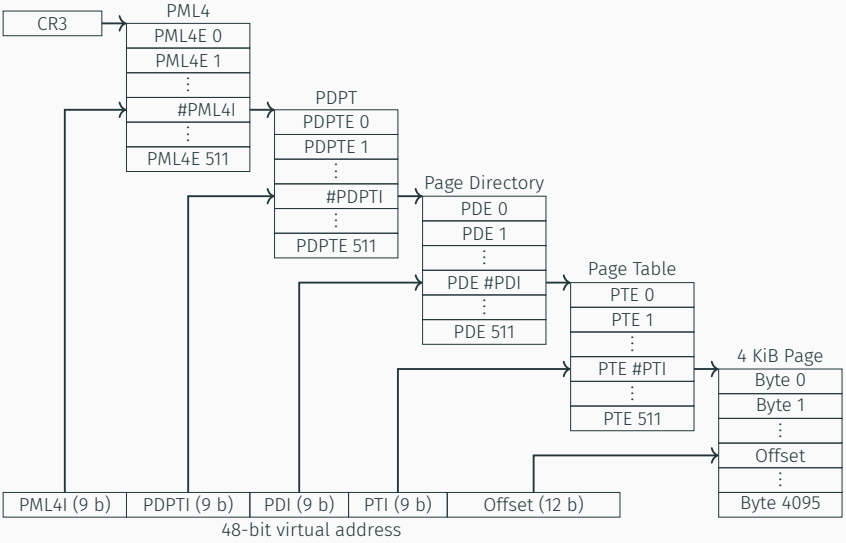
NOTE

Using the PREFETCH instruction is recommended only if data does not fit in cache. Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty. For example specifying a NULL pointer (0L) as address for a prefetch can cause long delays.

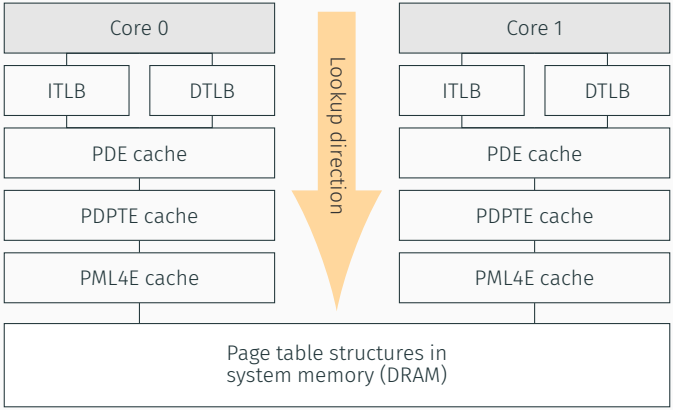
Intel Corporation. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. 2014

A little bit more background
before continuing...

Address translation

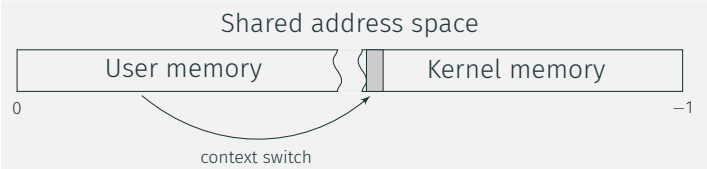


Address translation caches

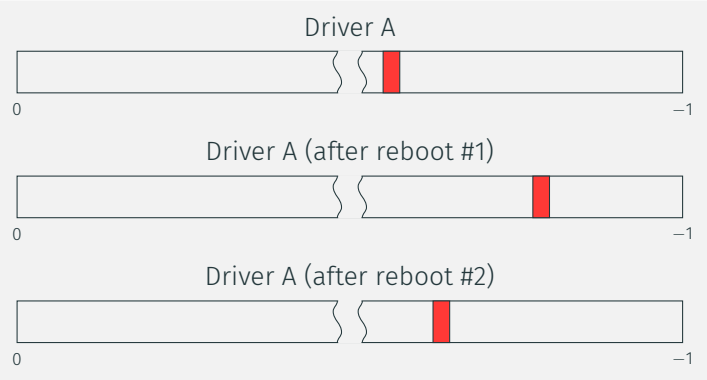


Kernel is mapped in every process

Today's operating systems:

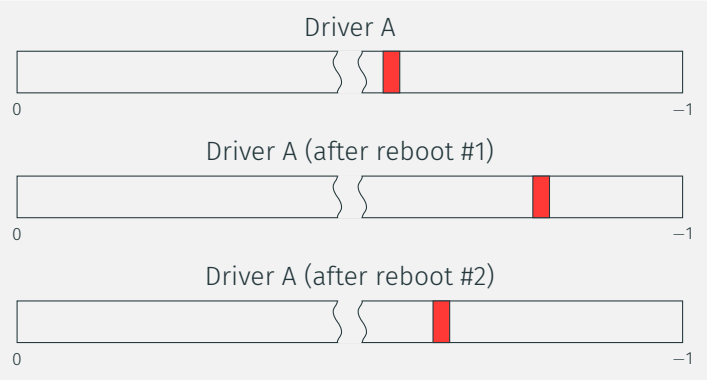


Kernel Address Space Layout Randomization (KASLR)



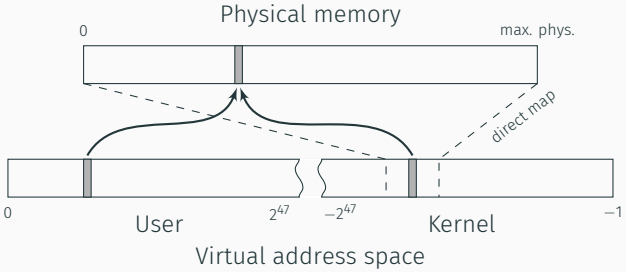
- same driver, different offset at each boot

Kernel Address Space Layout Randomization (KASLR)



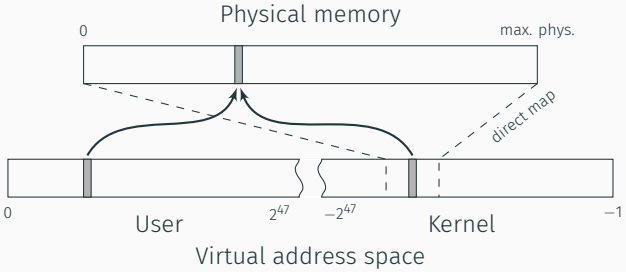
- same driver, different offset at each boot
- leaking kernel/driver addresses defeats KASLR

Kernel direct-physical map



- OS X, Linux, BSD, Xen PVM (Amazon EC2)

Kernel direct-physical map



- OS X, Linux, BSD, Xen PVM (Amazon EC2)
- not Windows

Let's go back to `prefetch`!

`prefetch`: Unusual instructions (1)

- tells the CPU “I might need that later”

D. Gruss et al. “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: *CCS'16*. 2016

`prefetch`: Unusual instructions (1)

- tells the CPU “I might need that later”
- hint—may be ignored by the CPU

D. Gruss et al. “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: *CCS'16*. 2016

`prefetch`: Unusual instructions (1)

- tells the CPU “I might need that later”
- hint—may be ignored by the CPU
- generates no faults

D. Gruss et al. “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: *CCS'16*. 2016

`prefetch`: Unusual instructions (1)

- tells the CPU “I might need that later”
- hint—may be ignored by the CPU
- generates no faults

Property #1: do not check privileges

D. Gruss et al. “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: *CCS'16*. 2016

`prefetch`: Unusual instructions (2)

- operand is a virtual address

D. Gruss et al. “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: *CCS'16*. 2016

`prefetch`: Unusual instructions (2)

- operand is a virtual address
- but it needs to translate the virtual address to a physical address

D. Gruss et al. “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: *CCS'16*. 2016

`prefetch`: Unusual instructions (2)

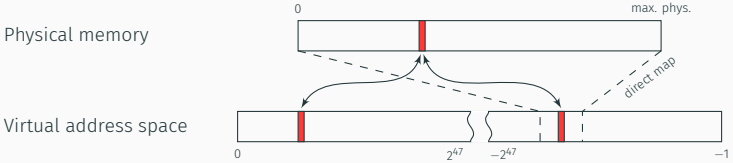
- operand is a virtual address
- but it needs to translate the virtual address to a physical address

Property #2: execution time varies

D. Gruss et al. "Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR". In: *CCS'16*. 2016

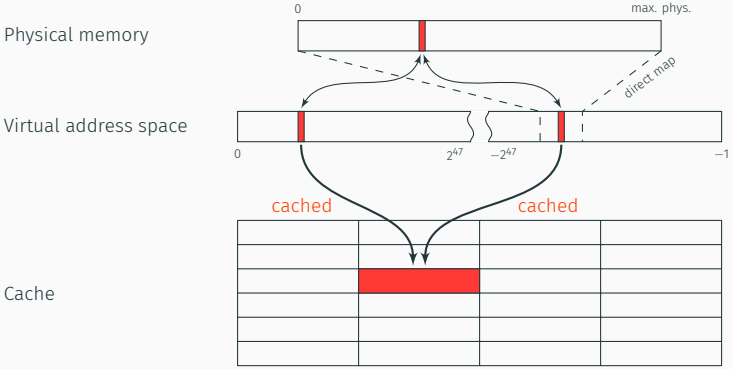
Address-translation oracle

Exploiting property #1 + kernel direct-physical map



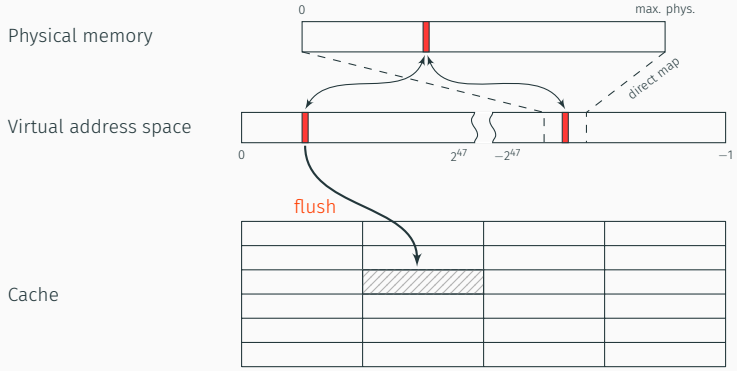
Address-translation oracle

Exploiting property #1 + kernel direct-physical map



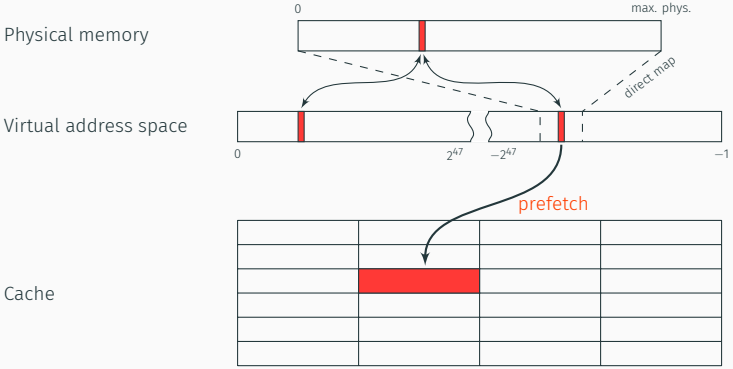
Address-translation oracle

Exploiting property #1 + kernel direct-physical map



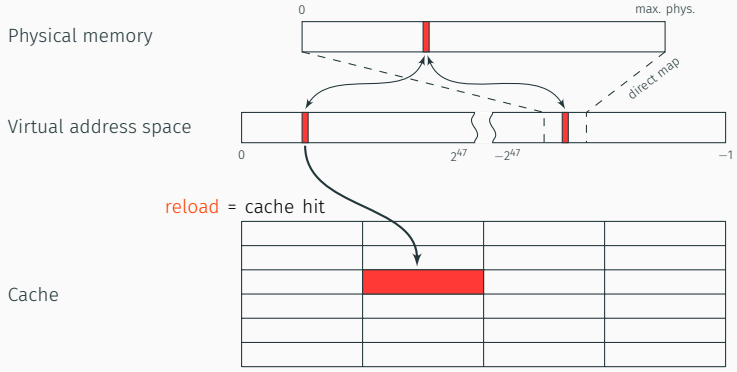
Address-translation oracle

Exploiting property #1 + kernel direct-physical map



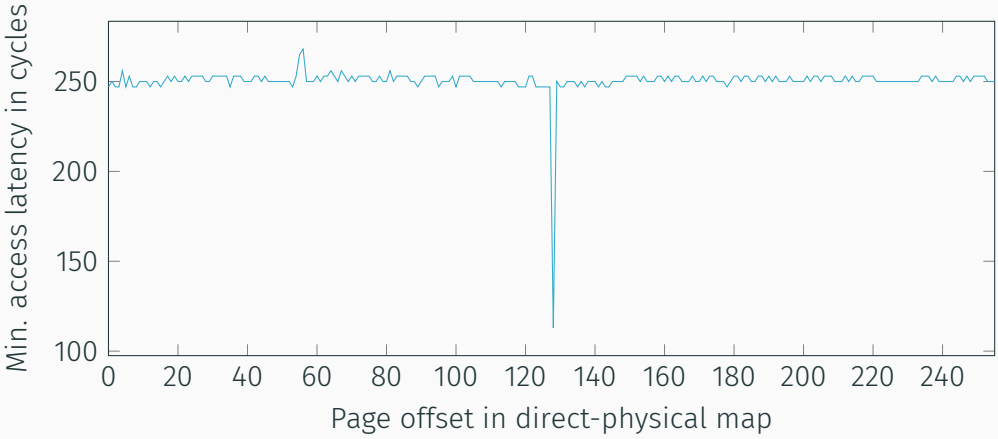
Address-translation oracle

Exploiting property #1 + kernel direct-physical map



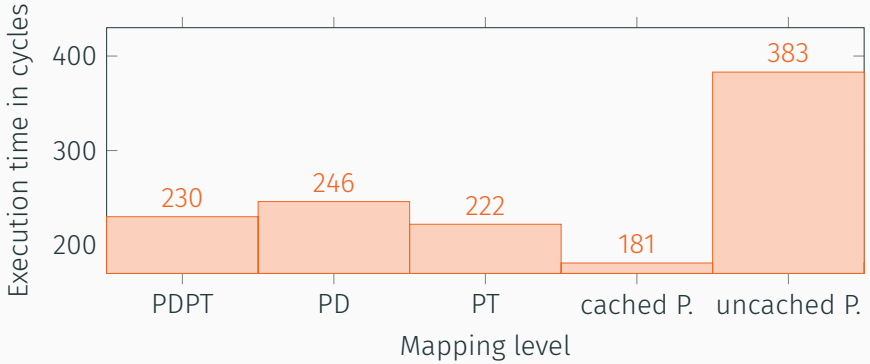
- cache hit → physical address in kernel mapping is the correct translation

Address-translation oracle



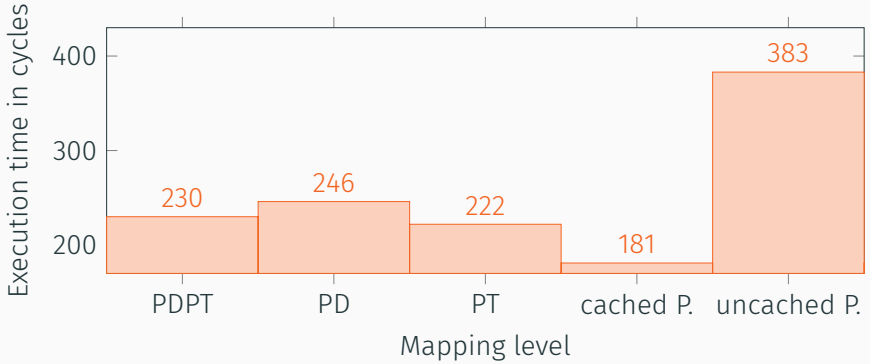
Translation-level oracle

Exploiting property #2



Translation-level oracle

Exploiting property #2



- timing depends on where the translation stops

Prefetch side-channel attacks

Using the two oracles

Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)

Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)
- Rowhammer attacks on privileged addresses

Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)
- Rowhammer attacks on privileged addresses
- recovering translation levels of a process ($\rightarrow$ `/proc/pid/pagemap`)
  $\rightarrow$ now privileged $\rightarrow$ bypasses ASLR

Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)
- Rowhammer attacks on privileged addresses
- recovering translation levels of a process (→ `/proc/pid/pagemap`)
  → now privileged → bypasses ASLR
- translating virtual addresses to physical addresses (→ `/proc/pid/pagemap`)
  → now privileged → re-enables ret2dir exploits

Prefetch side-channel attacks

Using the two oracles

- variants of cache attacks (e.g., Flush+Prefetch)
- Rowhammer attacks on privileged addresses
- recovering translation levels of a process (→ `/proc/pid/pagemap`)
  → now privileged → bypasses ASLR
- translating virtual addresses to physical addresses (→ `/proc/pid/pagemap`)
  → now privileged → re-enables ret2dir exploits
- locating kernel drivers
  → bypasses KASLR

Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

1. evict translation caches: `Sleep()` / access large memory buffer

Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

1. evict translation caches: `Sleep()` / access large memory buffer
2. perform syscall to driver

Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

1. evict translation caches: `Sleep()` / access large memory buffer
2. perform syscall to driver
3. time `prefetch(page address)`

Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

1. evict translation caches: `Sleep()` / access large memory buffer
2. perform syscall to driver
3. time `prefetch(page address)`

→ fastest average access time is a driver page

Defeating KASLR by locating kernel driver (1)

For all mapped pages, found with the translation-level oracle

1. evict translation caches: `Sleep()` / access large memory buffer
2. perform syscall to driver
3. time `prefetch(page address)`

$\rightarrow$ fastest average access time is a driver page

Full attack on Windows 10 in $<$ 12 seconds

Defeating KASLR by locating kernel driver (2)

Prefetch side-channel attacks: Take-away

- `prefetch` does not allow you to access memory

D. Gruss et al. “KASLR is Dead: Long Live KASLR”. In: *ESSoS'17*. to appear. 2017.

Prefetch side-channel attacks: Take-away

- `prefetch` does not allow you to access memory
- doesn't mean that it is a good idea to not perform any check

D. Gruss et al. "KASLR is Dead: Long Live KASLR". In: *ESSoS'17*. to appear. 2017.

Prefetch side-channel attacks: Take-away

- `prefetch` does not allow you to access memory
- doesn't mean that it is a good idea to not perform any check
- other microarchitectural attacks targeted KASLR

D. Gruss et al. "KASLR is Dead: Long Live KASLR". In: *ESSoS'17*. to appear. 2017.

Prefetch side-channel attacks: Take-away

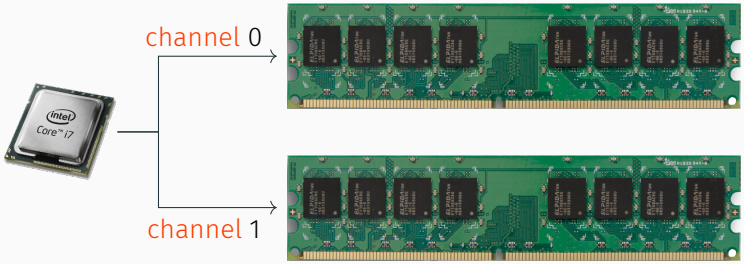
- `prefetch` does not allow you to access memory
- doesn't mean that it is a good idea to not perform any check
- other microarchitectural attacks targeted KASLR
- still not completely dead

D. Gruss et al. "KASLR is Dead: Long Live KASLR". In: *ESSoS'17*. to appear. 2017.

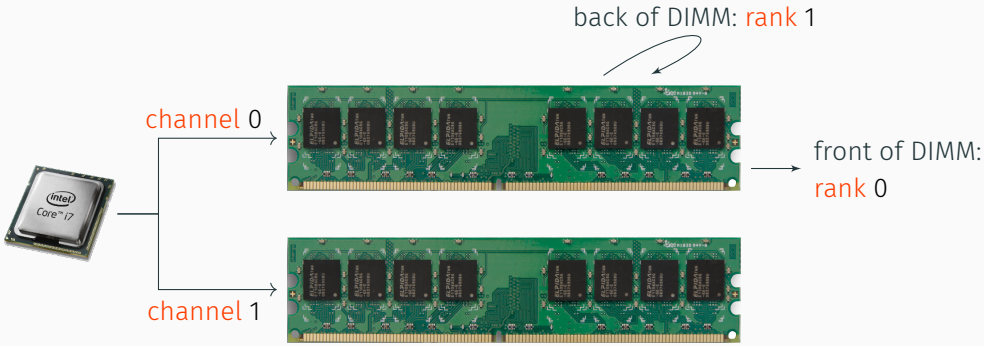
#5. Fault Attacks on DRAM in JavaScript

DRAM organization

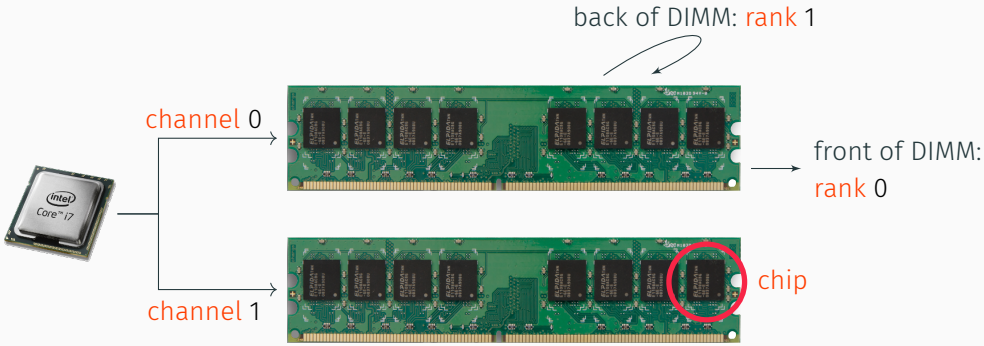
DRAM organization



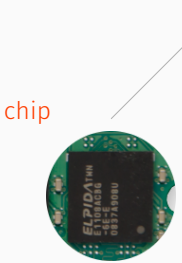
DRAM organization

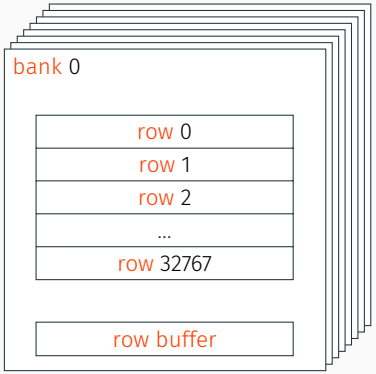


DRAM organization



DRAM organization





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

DRAM refresh

- cells leak $\rightarrow$ repetitive refresh necessary
- refresh $\approx$ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee data integrity

DRAM refresh

- cells leak → repetitive refresh necessary
- refresh ≈ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee data integrity

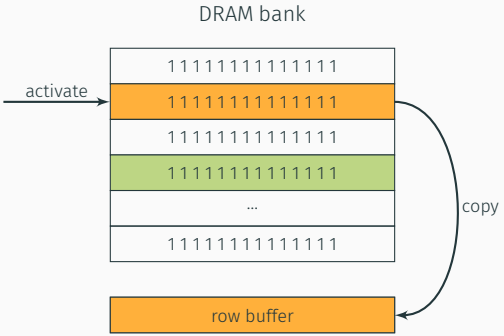
- cells leak faster upon proximate accesses → Rowhammer

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

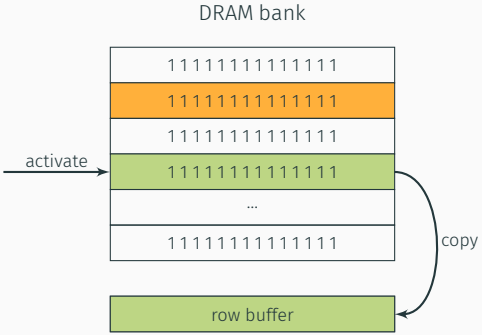
Rowhammer

"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



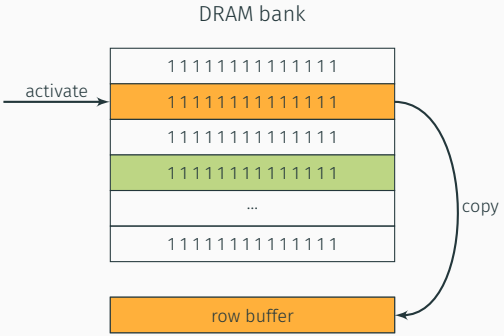
Rowhammer

"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



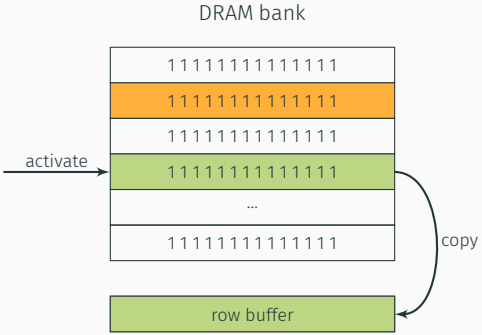
Rowhammer

"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



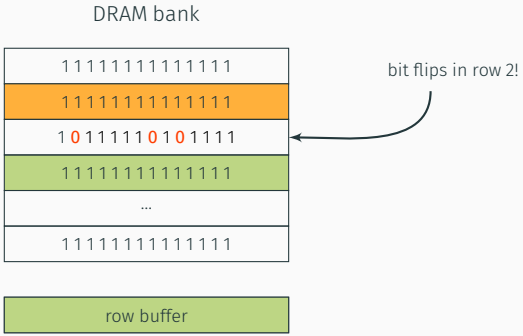
Rowhammer

"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

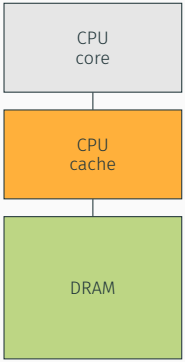


Rowhammer

"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



Impact of the CPU cache

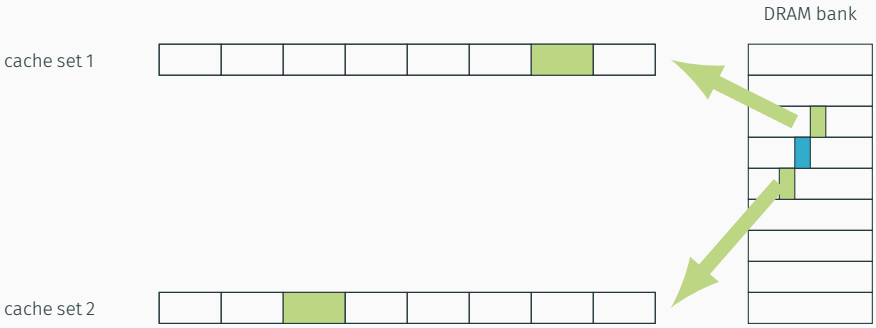


- only non-cached accesses reach DRAM
- original attacks use `clflush` instruction

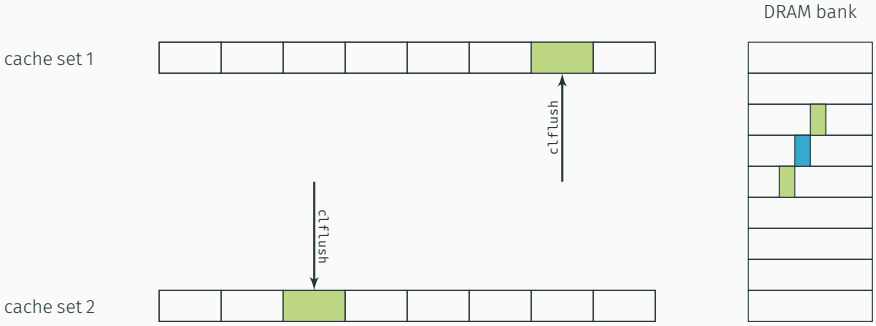
→ flush line from cache

→ next access will be served from DRAM

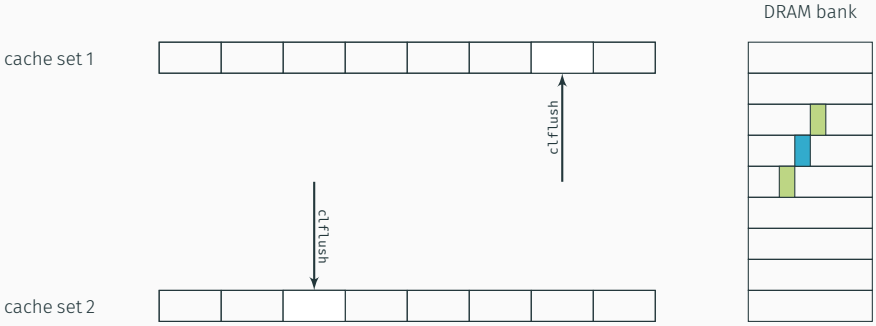
Rowhammer (with `clflush`)



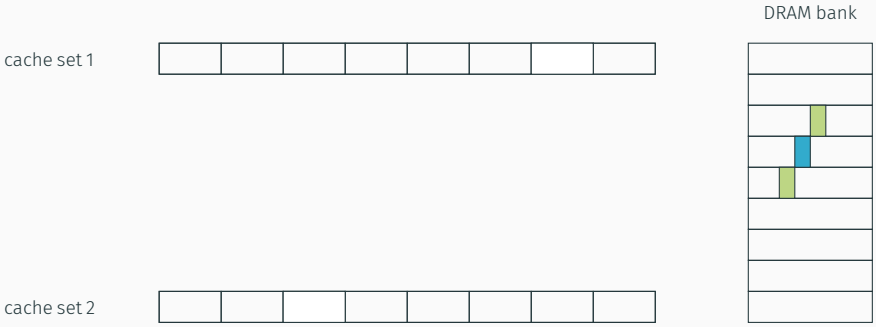
Rowhammer (with `clflush`)



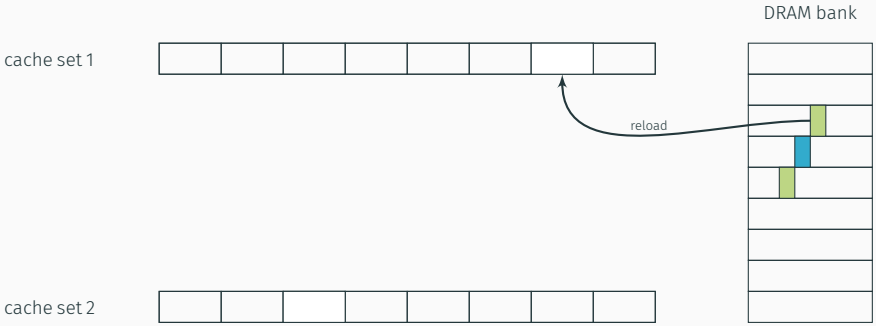
Rowhammer (with `clflush`)



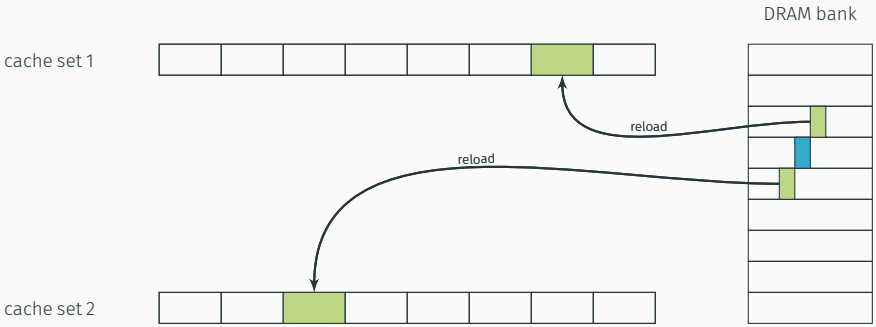
Rowhammer (with `clflush`)



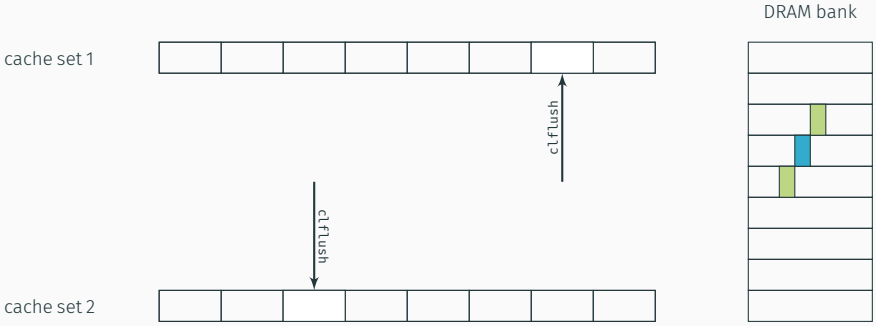
Rowhammer (with `clflush`)



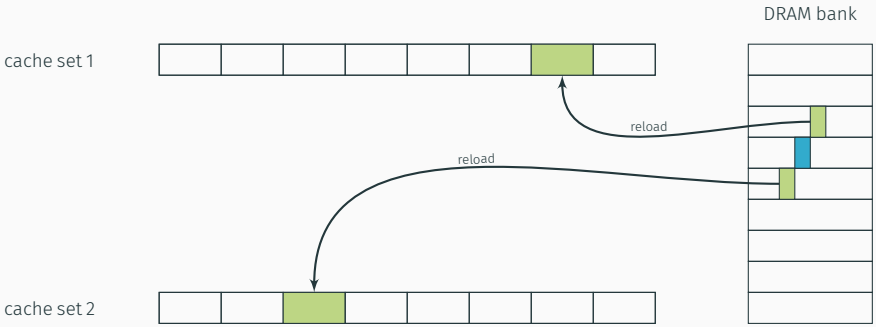
Rowhammer (with `clflush`)



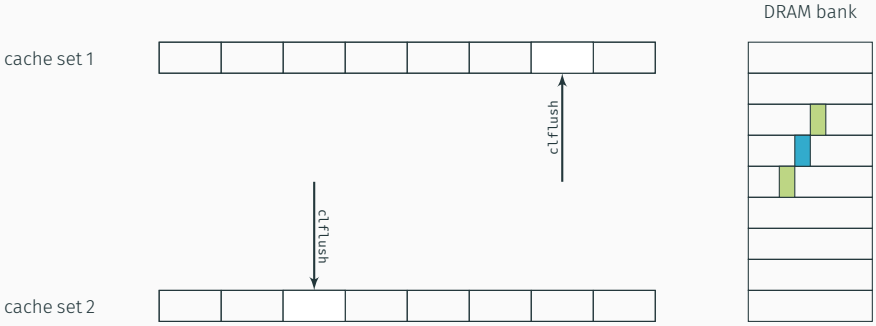
Rowhammer (with `clflush`)



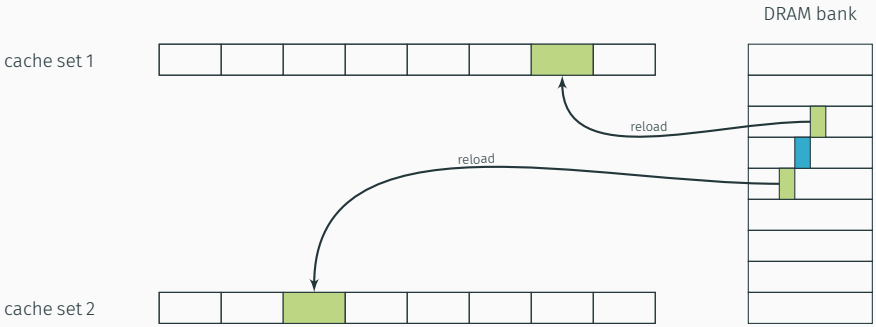
Rowhammer (with `clflush`)



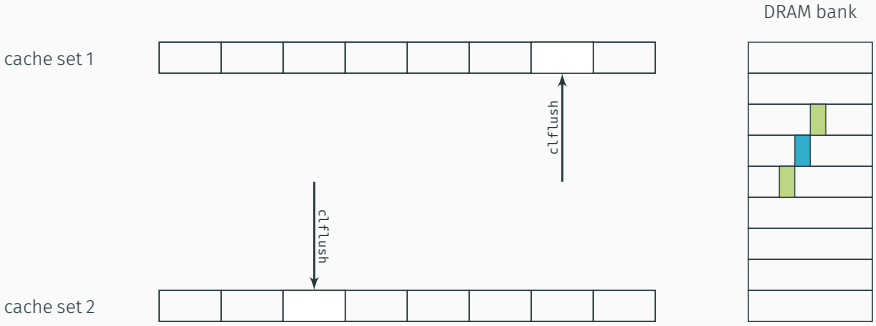
Rowhammer (with `clflush`)



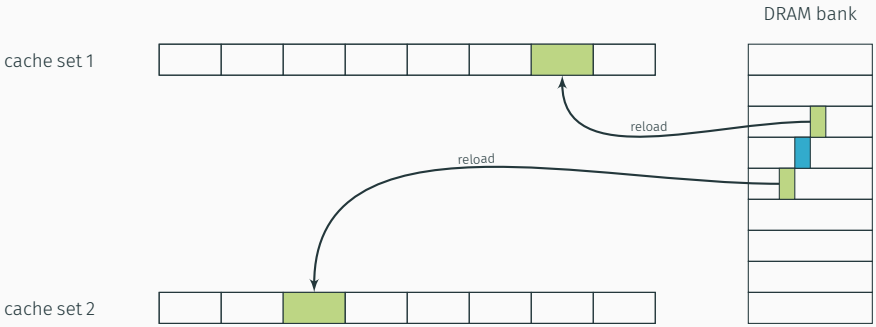
Rowhammer (with `clflush`)



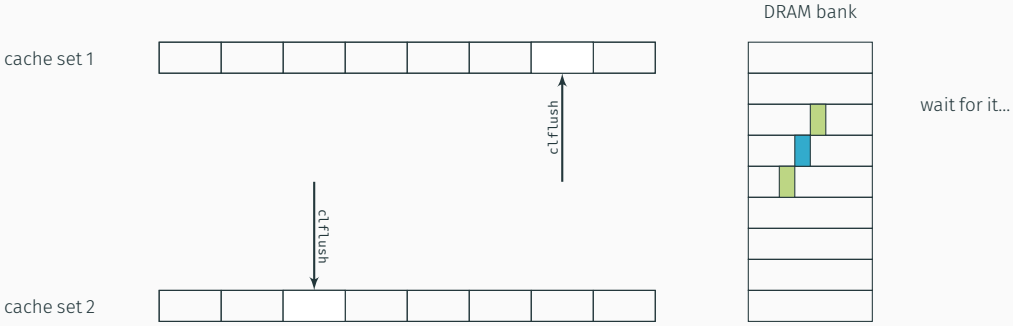
Rowhammer (with `clflush`)



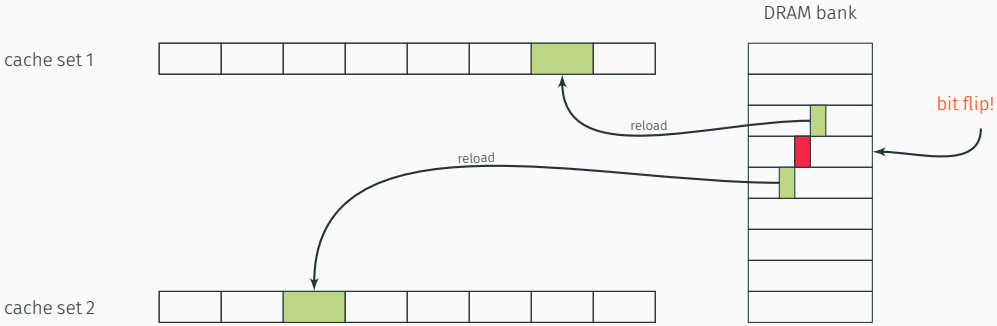
Rowhammer (with `clflush`)



Rowhammer (with `clflush`)



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
  $\rightarrow$ no `clflush` in JavaScript

D. Gruss et al. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA'16*. 2016.

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!

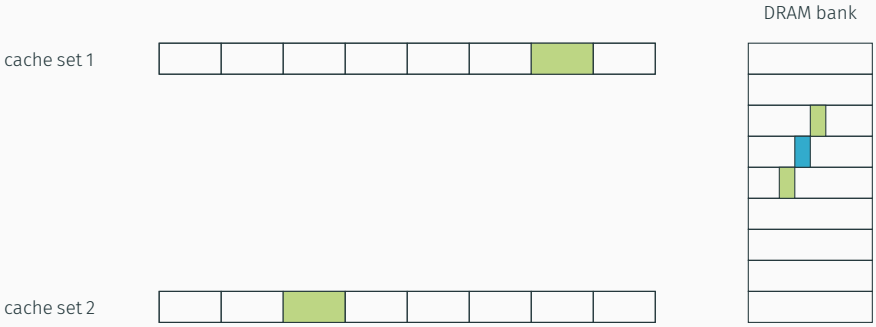
D. Gruss et al. “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: *DIMVA'16*. 2016.

Rowhammer without `clflush`?

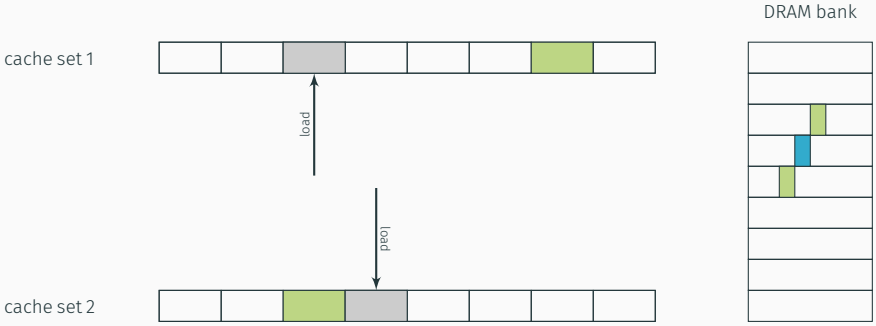
- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript
- our approach: use regular memory accesses for eviction
  → techniques from cache attacks!
  → Rowhammer, Prime+Probe style!

D. Gruss et al. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA'16*. 2016.

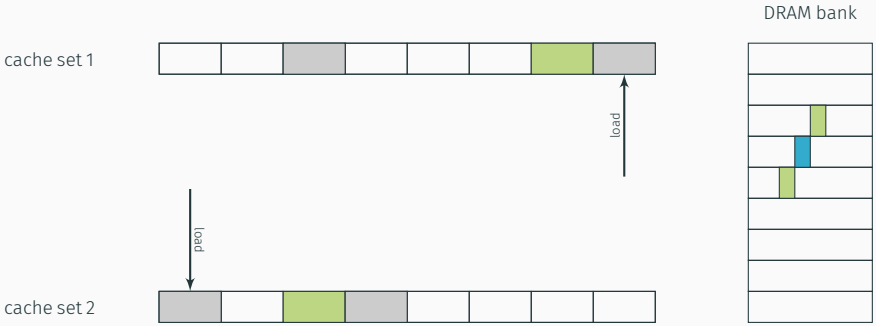
Rowhammer without `clflush`



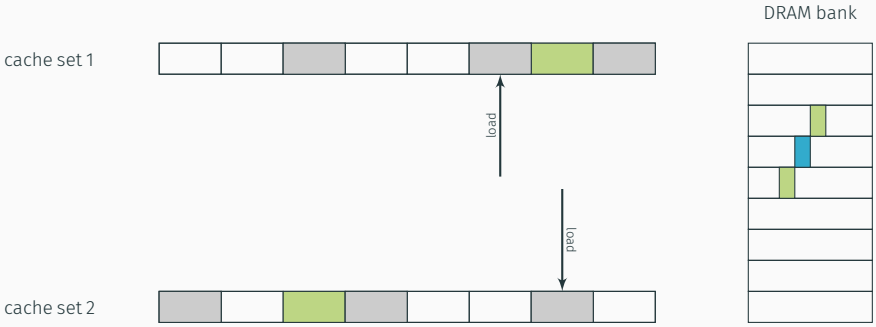
Rowhammer without `clflush`



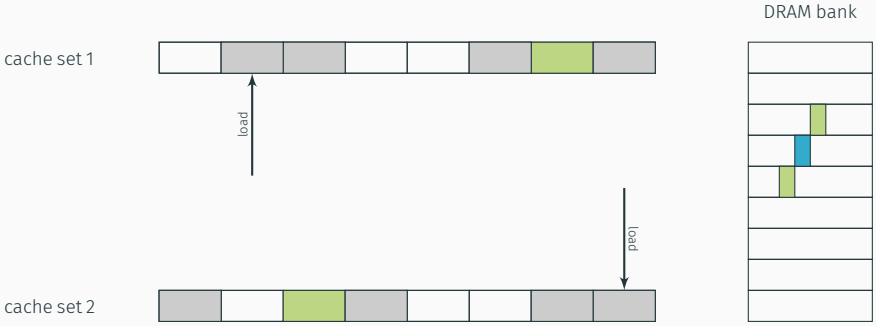
Rowhammer without `clflush`



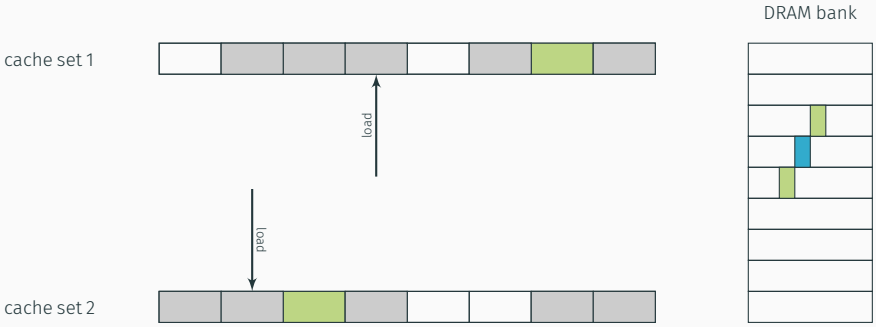
Rowhammer without `clflush`



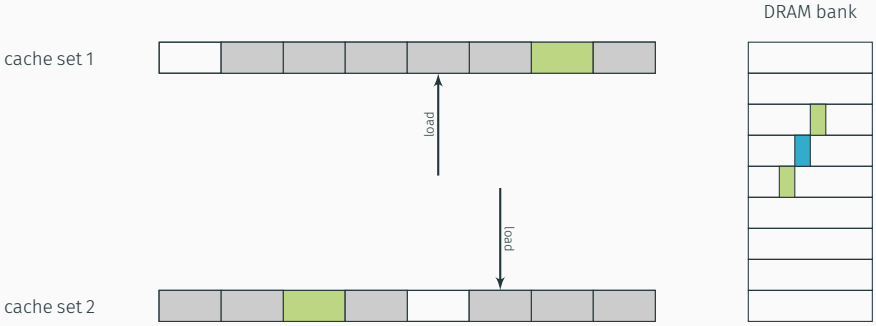
Rowhammer without `clflush`



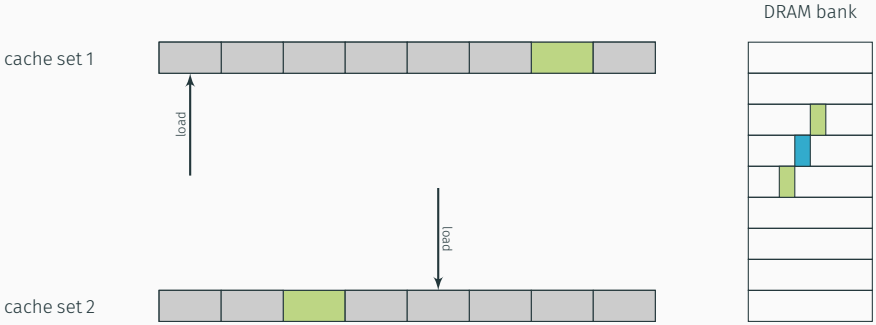
Rowhammer without `clflush`



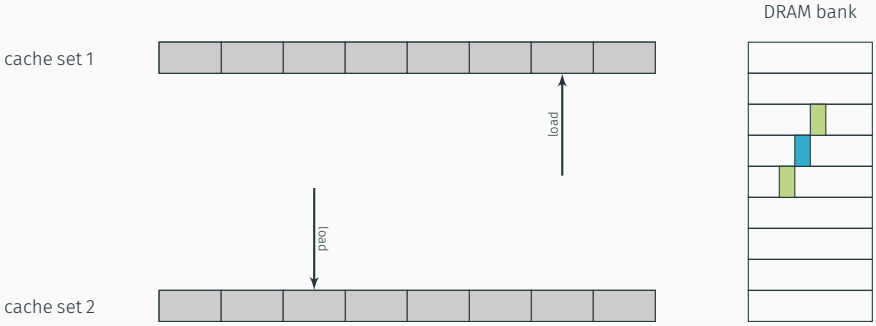
Rowhammer without `clflush`



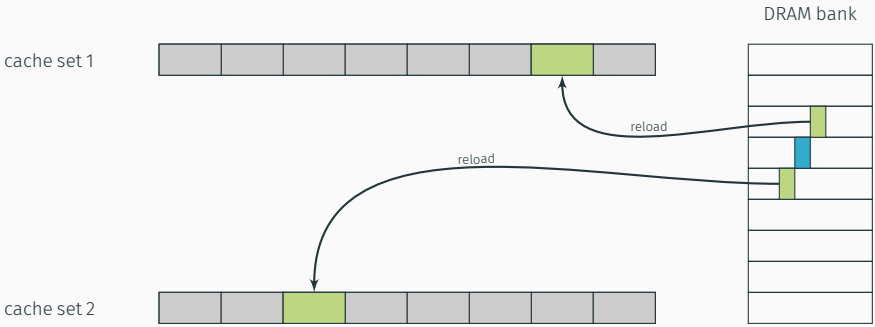
Rowhammer without `clflush`



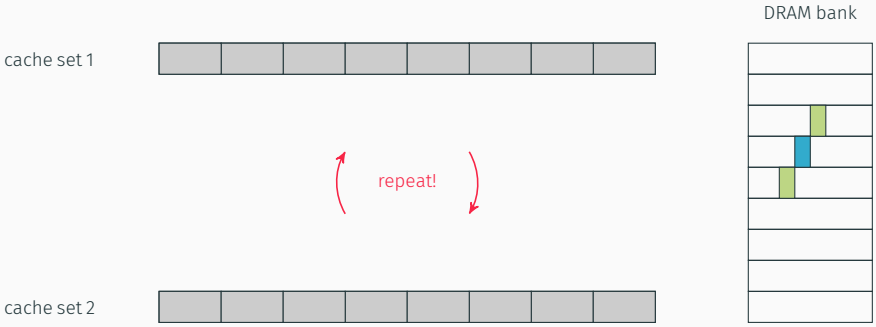
Rowhammer without `clflush`



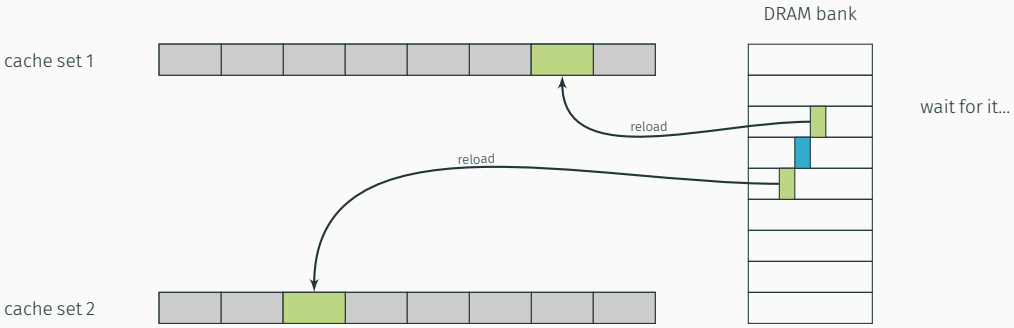
Rowhammer without `clflush`



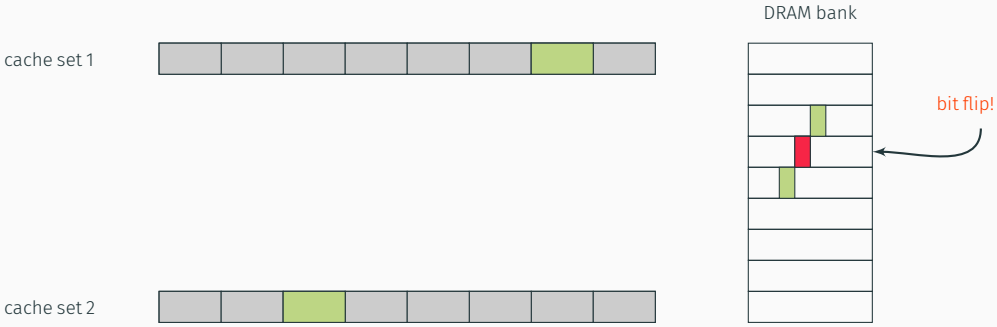
Rowhammer without `clflush`



Rowhammer without `clflush`

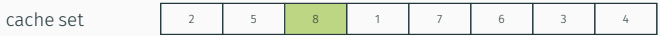


Rowhammer without `clflush`



Replacement policy since Ivy Bridge

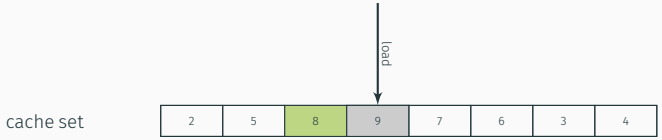
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

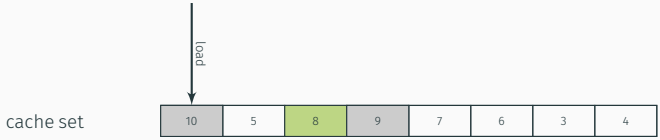
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

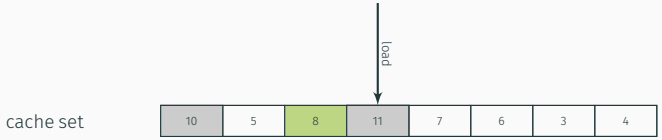
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

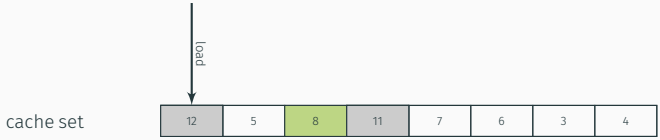
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

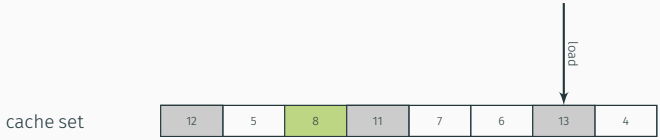
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

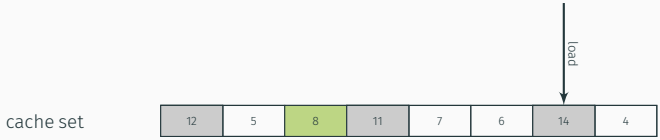
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

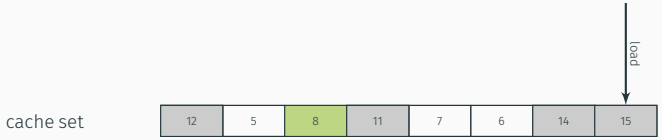
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

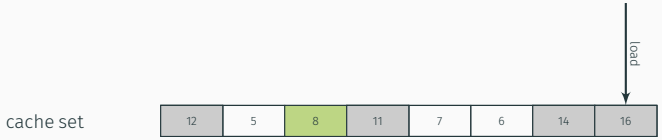
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

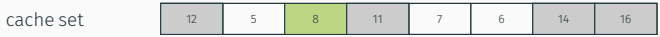
n accesses for an *n*-way cache



- no LRU replacement

Replacement policy since Ivy Bridge

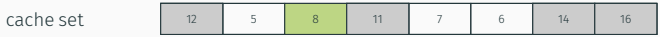
n accesses for an *n*-way cache



- no LRU replacement
- only 75% success rate on Haswell

Replacement policy since Ivy Bridge

n accesses for an *n*-way cache



- no LRU replacement
- only 75% success rate on Haswell
- more accesses $\rightarrow$ higher success rate, but too slow

Requirements for Rowhammer

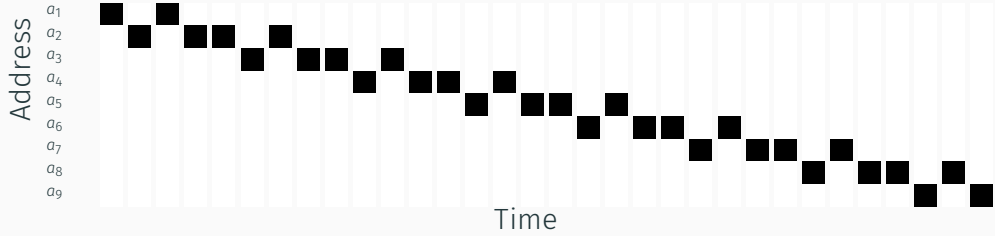
1. uncached memory accesses: need to reach DRAM
2. fast memory accesses: race against the next row refresh

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

Cache eviction strategies: The beginning



$\rightarrow$ fast and effective on Haswell: eviction rate $>$99.97%

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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

Cache eviction strategies: Evaluation

We evaluated more than 10000 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 ✓

→ more accesses, smaller execution time?

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

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

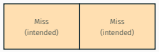
Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

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



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

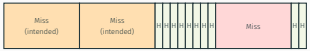
$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)



Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

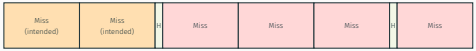
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

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

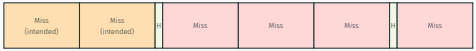


$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

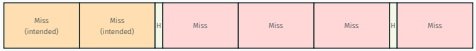
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

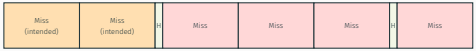
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

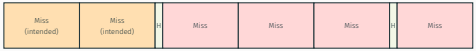
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

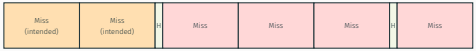
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



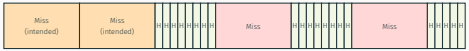
$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

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



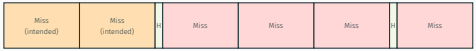
$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)



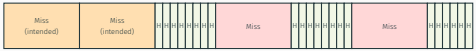
Time in ns

Cache eviction strategies: Illustration

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



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)



Time in ns

Cache eviction strategies: Illustration

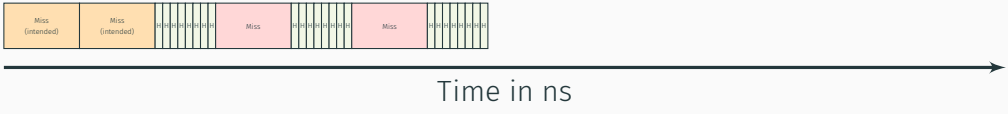
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

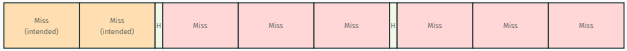
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

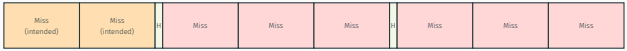
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

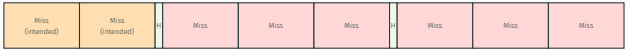
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

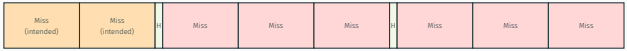
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



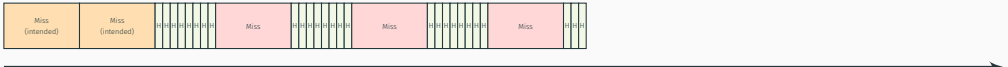
$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

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

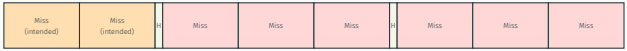


$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

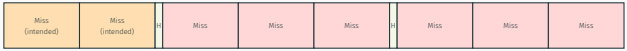
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

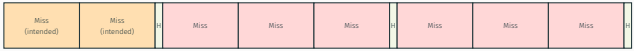
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

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

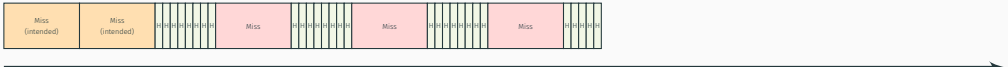


$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

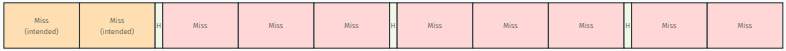
$\mathscr{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

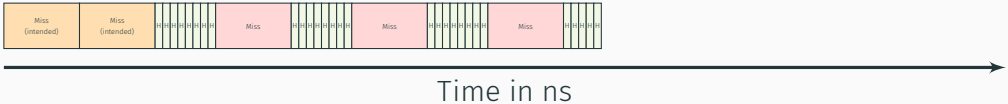


Cache eviction strategies: Illustration

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



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

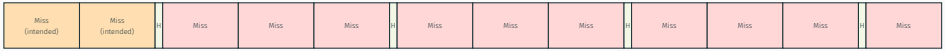
Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

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



$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

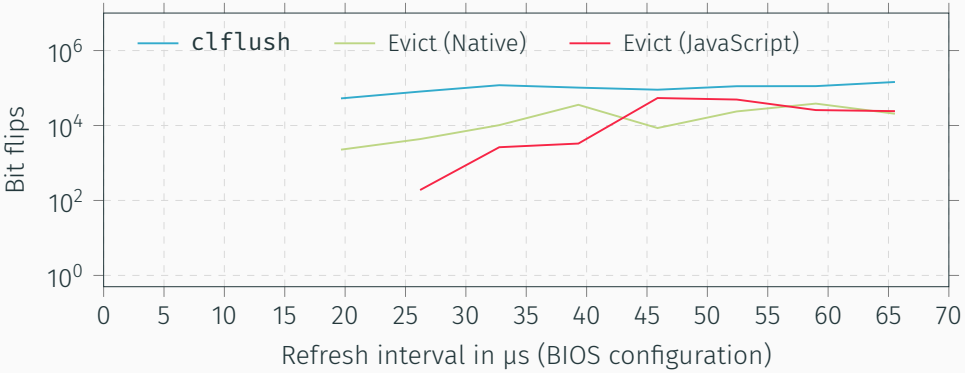
Cache eviction strategies: Illustration

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

$\mathscr{P}$-2-1-1-17 (34 accesses, 191ns)

Time in ns

Evaluation on Haswell



Number of bit flips within 15 minutes.

Rowhammer.js: Take-Away

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions

E. Bosman et al. “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector”. In: *S&P’16*. 2016

Rowhammer.js: Take-Away

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions
- lessons learned from side channels to perform a fault attack

E. Bosman et al. “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector”. In: *S&P’16*. 2016

Rowhammer.js: Take-Away

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions
- lessons learned from side channels to perform a fault attack
- first remote fault attack, from a browser

E. Bosman et al. “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector”. In: *S&P’16*. 2016

Rowhammer.js: Take-Away

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions
- lessons learned from side channels to perform a fault attack
- first remote fault attack, from a browser
- if you think a fault is not exploitable, think again

E. Bosman et al. “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector”. In: *S&P’16*. 2016

Countermeasures

Countermeasures

- different levels: hardware, system, application
- different goals
  - remove interferences
  - add noise to interferences
  - make it impossible to measure interferences

Hardware level: Fixing the instruction set?

- `clflush`
  - unprivileged line eviction

Hardware level: Fixing the instruction set?

- `clflush`
  - unprivileged line eviction → make it privileged

Hardware level: Fixing the instruction set?

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information

Hardware level: Fixing the instruction set?

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time

Hardware level: Fixing the instruction set?

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time
- `rdtsc`
  - unprivileged fine-grained timing

Hardware level: Fixing the instruction set?

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time
- `rdtsc`
  - unprivileged fine-grained timing → make it privileged

Hardware level: Fixing the instruction set?

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time
- `rdtsc`
  - unprivileged fine-grained timing → make it privileged

→ require changes to the architecture

Hardware level: Fixing the instruction set?

- `clflush`
  - unprivileged line eviction → make it privileged
  - leaks timing information → make it constant-time
- `rdtsc`
  - unprivileged fine-grained timing → make it privileged

→ require changes to the architecture

→ attacks still possible (e.g., Prime+Probe)

Hardware level: Stop sharing hardware?

- stop sharing cache

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU
  - 2016: first attack across processors

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU
  - 2016: first attack across processors → what next?

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU
  - 2016: first attack across processors → what next?
- not an option for cost reasons in the cloud

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Stop sharing hardware?

- stop sharing cache → attacks are getting better
  - first attacks on L1 → same core → stop sharing core
  - current attacks on LLC → across cores → stop sharing CPU
  - 2016: first attack across processors → what next?
- not an option for cost reasons in the cloud
- what about JavaScript attacks?

G. Irazoqui et al. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache

Z. Wang et al. “New cache designs for thwarting software cache-based side channel attacks”. In: *ACM SIGARCH Computer Architecture News* 35.2 (June 2007), p. 494

J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: *HPCA'09*. 2009.

A. Fuchs et al. “Disruptive Prefetching: Impact on Side-Channel Attacks and Cache Designs”. In: *SYSTOR'15*. 2015

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  $\rightarrow$ expensive, not always high performance

Z. Wang et al. “New cache designs for thwarting software cache-based side channel attacks”. In: *ACM SIGARCH Computer Architecture News* 35.2 (June 2007), p. 494

J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: *HPCA'09*. 2009.

A. Fuchs et al. “Disruptive Prefetching: Impact on Side-Channel Attacks and Cache Designs”. In: *SYSTOR'15*. 2015

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  $\rightarrow$ expensive, not always high performance
- disruptive prefetching: random hardware prefetches

Z. Wang et al. “New cache designs for thwarting software cache-based side channel attacks”. In: *ACM SIGARCH Computer Architecture News* 35.2 (June 2007), p. 494

J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: *HPCA'09*. 2009.

A. Fuchs et al. “Disruptive Prefetching: Impact on Side-Channel Attacks and Cache Designs”. In: *SYSTOR'15*. 2015

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  → expensive, not always high performance
- disruptive prefetching: random hardware prefetches
  → adding noise makes attacks harder, not impossible

Z. Wang et al. “New cache designs for thwarting software cache-based side channel attacks”. In: *ACM SIGARCH Computer Architecture News* 35.2 (June 2007), p. 494

J. Kong et al. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: *HPCA'09*. 2009.

A. Fuchs et al. “Disruptive Prefetching: Impact on Side-Channel Attacks and Cache Designs”. In: *SYSTOR'15*. 2015

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  - → expensive, not always high performance
- disruptive prefetching: random hardware prefetches
  - → adding noise makes attacks harder, not impossible

→ trade-off security/performance/cost

Z. Wang et al. "New cache designs for thwarting software cache-based side channel attacks". In: *ACM SIGARCH Computer Architecture News* 35.2 (June 2007), p. 494

J. Kong et al. "Hardware-software integrated approaches to defend against software cache-based side channel attacks". In: *HPCA'09*. 2009.

A. Fuchs et al. "Disruptive Prefetching: Impact on Side-Channel Attacks and Cache Designs". In: *SYSTOR'15*. 2015

Hardware level: Changes in microarchitecture

- secure cache designs: Random-Permutation Cache, Partition-Locked Cache
  - → expensive, not always high performance
- disruptive prefetching: random hardware prefetches
  - → adding noise makes attacks harder, not impossible

→ trade-off security/performance/cost

→ performance and cost win, no implementation by manufacturers

---

Z. Wang et al. "New cache designs for thwarting software cache-based side channel attacks". In: *ACM SIGARCH Computer Architecture News* 35.2 (June 2007), p. 494

J. Kong et al. "Hardware-software integrated approaches to defend against software cache-based side channel attacks". In: *HPCA'09*. 2009.

A. Fuchs et al. "Disruptive Prefetching: Impact on Side-Channel Attacks and Cache Designs". In: *SYSTOR'15*. 2015

System level: Prevention

- L1 cache cleansing

Y. Zhang et al. “Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: *CCS’13*. 2013

H. Raj et al. “Resource Management for Isolation Enhanced Cloud Services”. In: *CCSW’09*. 2009

B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: *CCSW’11*. 2011

M. Schwarz et al. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC’17*. 2017.

System level: Prevention

- L1 cache cleansing
  - $\rightarrow$ if applied to LLC $\rightarrow$ same as no cache, disastrous performance

Y. Zhang et al. “Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: *CCS’13*. 2013

H. Raj et al. “Resource Management for Isolation Enhanced Cloud Services”. In: *CCSW’09*. 2009

B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: *CCSW’11*. 2011

M. Schwarz et al. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC’17*. 2017.

System level: Prevention

- L1 cache cleansing
  → if applied to LLC → same as no cache, disastrous performance
- page coloring → reduces cache sharing

Y. Zhang et al. “Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud”. In: *CCS’13*. 2013

H. Raj et al. “Resource Management for Isolation Enhanced Cloud Services”. In: *CCSW’09*. 2009

B. C. Vattikonda et al. “Eliminating fine grained timers in Xen”. In: *CCSW’11*. 2011

M. Schwarz et al. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC’17*. 2017.

System level: Prevention

- L1 cache cleansing
  - → if applied to LLC → same as no cache, disastrous performance
- page coloring → reduces cache sharing
  - → limited number of colors + bad performance
  - → doesn't prevent Flush+Reload

Y. Zhang et al. "Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud". In: *CCS'13*. 2013

H. Raj et al. "Resource Management for Isolation Enhanced Cloud Services". In: *CCSW'09*. 2009

B. C. Vattikonda et al. "Eliminating fine grained timers in Xen". In: *CCSW'11*. 2011

M. Schwarz et al. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript". In: *FC'17*. 2017.

System level: Prevention

- L1 cache cleansing
  - → if applied to LLC → same as no cache, disastrous performance
- page coloring → reduces cache sharing
  - → limited number of colors + bad performance
  - → doesn't prevent Flush+Reload
- noise in timers or no timer

Y. Zhang et al. "Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud". In: *CCS'13*. 2013

H. Raj et al. "Resource Management for Isolation Enhanced Cloud Services". In: *CCSW'09*. 2009

B. C. Vattikonda et al. "Eliminating fine grained timers in Xen". In: *CCSW'11*. 2011

M. Schwarz et al. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript". In: *FC'17*. 2017.

System level: Prevention

- L1 cache cleansing
  - → if applied to LLC → same as no cache, disastrous performance
- page coloring → reduces cache sharing
  - → limited number of colors + bad performance
  - → doesn't prevent Flush+Reload
- noise in timers or no timer
  - → adding noise makes attacks harder, not impossible
  - → removing timers is not realistic

Y. Zhang et al. "Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud". In: *CCS'13*. 2013

H. Raj et al. "Resource Management for Isolation Enhanced Cloud Services". In: *CCSW'09*. 2009

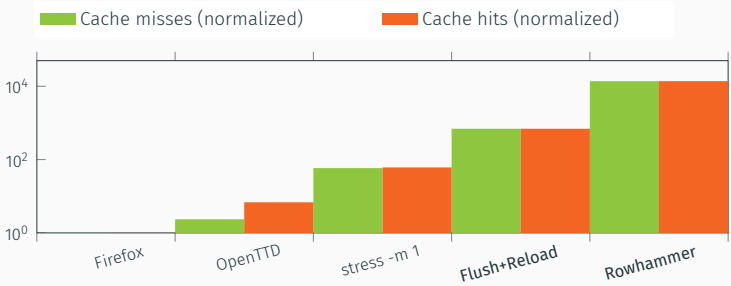
B. C. Vattikonda et al. "Eliminating fine grained timers in Xen". In: *CCSW'11*. 2011

M. Schwarz et al. "Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript". In: *FC'17*. 2017.

System level: Detect on-going attacks

- using performance counters to monitor cache hits and cache misses

→ risk of false positives



N. Herath et al. “These are Not Your Grand Daddys CPU Performance Counters – CPU Hardware Performance Counters for Security”. In: *Black Hat 2015 Briefings*. 2015

D. Gruss et al. “Flush+Flush: A Fast and Stealthy Cache Attack”. In: *DIMVA'16*. 2016

Application level: Detect leakage

- CacheAudit : static analysis of source code
- Cache Template Attacks : dynamic approach

G. Doychev et al. “CacheAudit: A Tool for the Static Analysis of Cache Side Channels”. In: *USENIX Security Symposium*. 2013

D. Gruss et al. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015

Application level: Detect leakage

- CacheAudit : static analysis of source code
- Cache Template Attacks : dynamic approach

→ limited to side-channels → covert channels still possible

→ most effective for critical code

G. Doychev et al. “CacheAudit: A Tool for the Static Analysis of Cache Side Channels”. In: *USENIX Security Symposium*. 2013

D. Gruss et al. “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: *USENIX Security Symposium*. 2015

Application level: Detect leakage

- CacheAudit : static analysis of source code
- Cache Template Attacks : dynamic approach

→ limited to side-channels → covert channels still possible

→ most effective for critical code

→ see Boris' talk this afternoon!

G. Doychev et al. "CacheAudit: A Tool for the Static Analysis of Cache Side Channels". In: *USENIX Security Symposium*. 2013

D. Gruss et al. "Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches". In: *USENIX Security Symposium*. 2015

Application level: Write better code

- square-and-multiply-always algorithm
- bit-sliced AES implementation
- hardware implementations (AES-NI, etc.)

Application level: Write better code

- square-and-multiply-always algorithm
- bit-sliced AES implementation
- hardware implementations (AES-NI, etc.)

→ protecting crypto is possible and necessary!

→ a few CVEs that have been treated: CVE-2005-0109, CVE-2013-4242, CVE-2014-0076, CVE-2016-0702, CVE-2016-2178

Bigger Perspective and Conclusions

`rdseed` and floating point operations

- `rdseed`
  - request a random seed to the hardware random number generator
  - fixed number of precomputed random bits, takes time to regenerate them

  $\rightarrow$ covert channel

D. Evtyushkin et al. “Covert Channels through Random Number Generator: Mechanisms, Capacity Estimation and Mitigations”. In: *CCS’16*. 2016

M. Andrysco et al. “On subnormal floating point and abnormal timing”. In: *S&P’15*. 2015

`rdseed` and floating point operations

- `rdseed`
  - request a random seed to the hardware random number generator
  - fixed number of precomputed random bits, takes time to regenerate them

  → covert channel
- `fadd`, `fmul`
  - floating-point unit
  - floating point operations running time depends on the operands

  → bypassing Firefox's same origin policy via SVG filter timing attack

---

D. Evtyushkin et al. "Covert Channels through Random Number Generator: Mechanisms, Capacity Estimation and Mitigations". In: *CCS'16*. 2016

M. Andrysco et al. "On subnormal floating point and abnormal timing". In: *S&P'15*. 2015

`jmp` and TSX instructions

- `jmp`
  - branch prediction and branch target prediction → branch prediction unit

  → covert channels, side-channel attacks on crypto, bypassing kernel ASLR

O. Acıiçmez et al. “Predicting secret keys via branch prediction”. In: *CT-RSA 2007*. 2007

D. Evtyushkin et al. “Jump over ASLR: Attacking branch predictors to bypass ASLR”. In: *MICRO'16*. 2016

Y. Jang et al. “Breaking kernel address space layout randomization with intel TSX”. In: *CCS'16*. 2016

`jmp` and TSX instructions

- `jmp`
  - branch prediction and branch target prediction → branch prediction unit

  → covert channels, side-channel attacks on crypto, bypassing kernel ASLR
- TSX instructions
  - extension for transactional memory support in hardware

  → bypassing kernel ASLR

O. Acıiçmez et al. “Predicting secret keys via branch prediction”. In: *CT-RSA 2007*. 2007

D. Evtyushkin et al. “Jump over ASLR: Attacking branch predictors to bypass ASLR”. In: *MICRO'16*. 2016

Y. Jang et al. “Breaking kernel address space layout randomization with intel TSX”. In: *CCS'16*. 2016

Conclusion

- more a problem of CPU design than Instruction Set Architecture

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- hard to patch $\rightarrow$ issues linked to performance optimizations

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- hard to patch $\rightarrow$ issues linked to performance optimizations
- quick fixes like removing instructions won't work

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- hard to patch → issues linked to performance optimizations
- quick fixes like removing instructions won't work

→ looking forward to Boris' talk this afternoon!

Questions?

Contact

clementine.maurice@iaik.tugraz.at

@BloodyTangerine

Cache attacks:
Software side-channel and fault attacks

Clémentine Maurice, Graz University of Technology

May 11, 2017—Spring school on Security & Correctness in the IoT 2017