Introduction to cache side-channel attacks

Clémentine Maurice, CNRS, IRISA

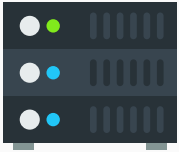
October 9, 2018—Séminaire du DIT, ENS Rennes

Who am I

- **Clémentine Maurice**
- Chargée de Recherche CNRS
- IRISA lab, EMSEC group
- clementine.maurice@irisa.fr
- @BloodyTangerine

Scope

Everyday hardware: servers, workstations, laptops, smartphones...



Side channels

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

Side channels

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

Side channels

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

Side channels

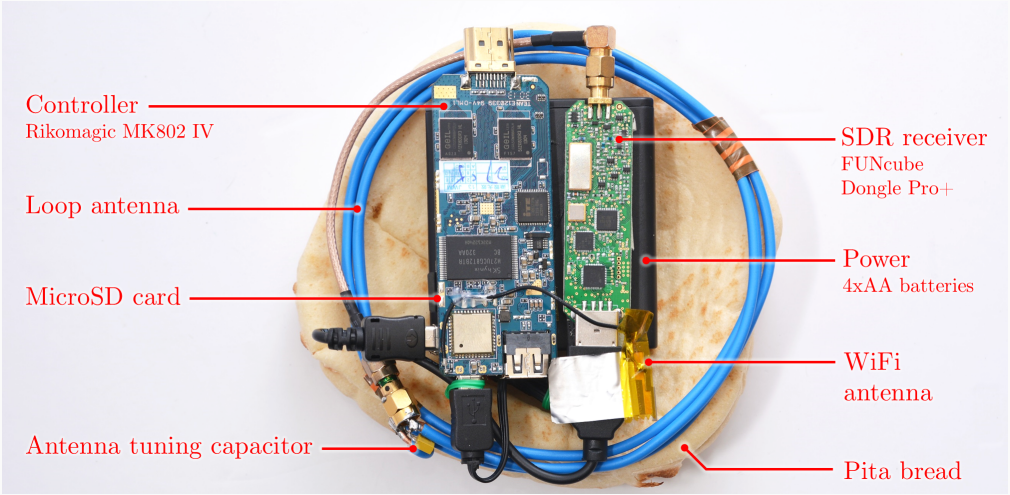
- safe software infrastructure → no bugs, e.g., buffer overflows
- does not mean safe execution
- secrets leak because of the hardware it runs on
- no “bug” in the sense of a mistake → lots of performance optimizations

→ 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

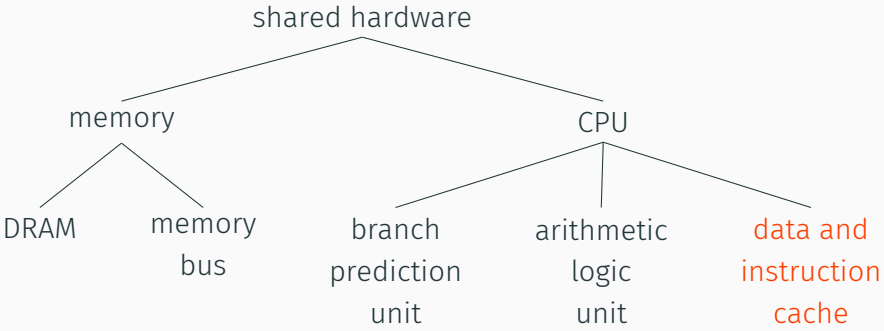
Sources of leakage

- 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, no physical access to the device

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 $\rightarrow$ 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 (may. 2018): 4844 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 (may. 2018): 4844 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 on cache attacks
- Reverse-engineering
- Practical attacks
- Countermeasures and open challenges
- Conclusion

Background on cache attacks

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?

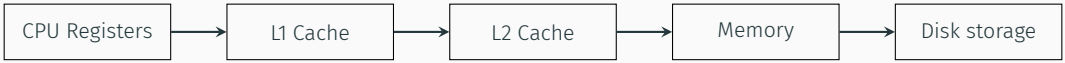
- lots of exceptions for `mov`
- but accessing data loads it to the cache

`mov`—What could go wrong?

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

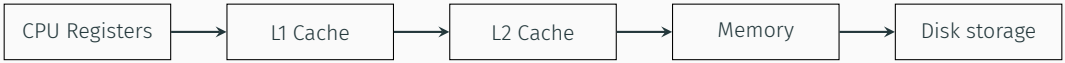
→ side effects on computations!

Memory hierarchy



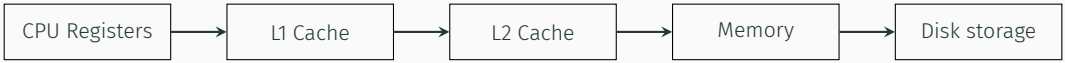
- data can reside in

Memory hierarchy



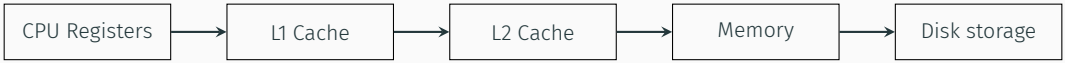
- data can reside in
  - CPU registers

Memory hierarchy



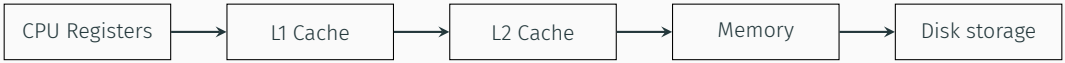
- data can reside in
  - CPU registers
  - different levels of the CPU cache

Memory hierarchy



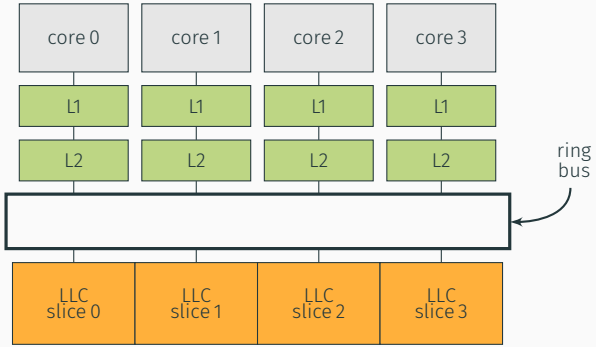
- data can reside in
  - CPU registers
  - different levels of the CPU cache
  - main memory

Memory hierarchy



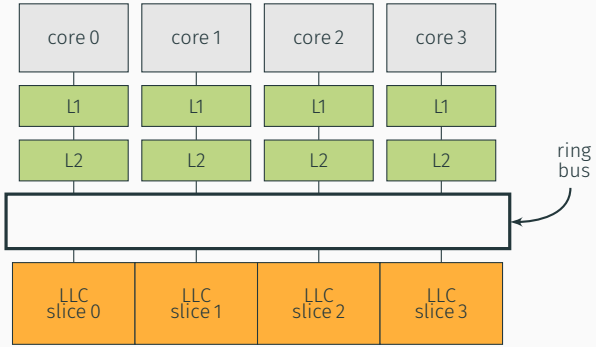
- data can reside in
  - CPU registers
  - different levels of the CPU cache
  - main memory
  - disk storage

Caches on Intel CPUs



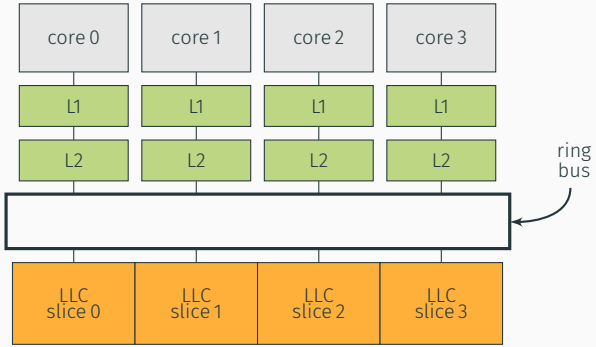
- L1 and L2 are private

Caches on Intel CPUs



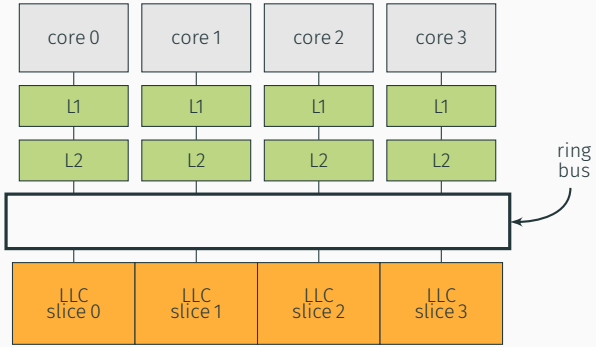
- L1 and L2 are private
- last-level cache

Caches on Intel CPUs



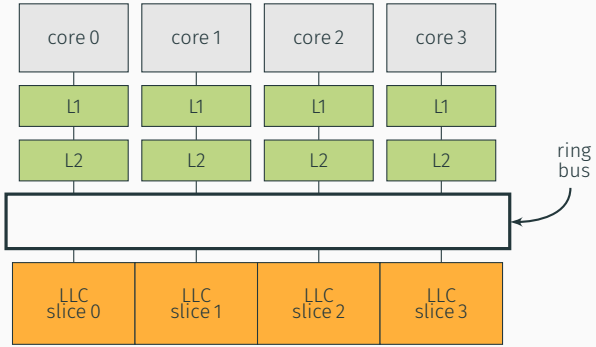
- L1 and L2 are private
- last-level cache
  - divided in slices

Caches on Intel CPUs



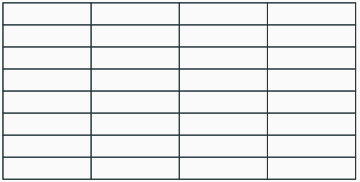
- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores

Caches on Intel CPUs



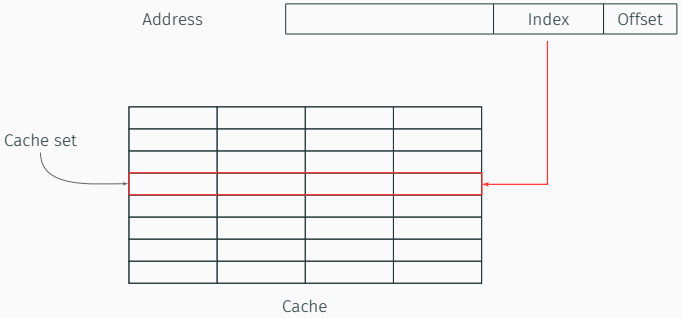
- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive

Set-associative caches



Cache

Set-associative caches



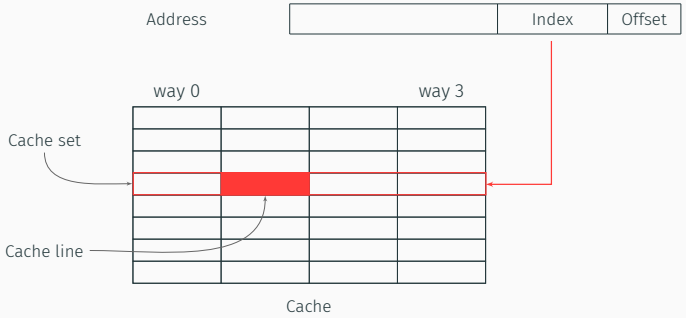
Data loaded in a specific set depending on its address

Set-associative caches

Data loaded in a specific set depending on its address

Several ways per set

Set-associative caches

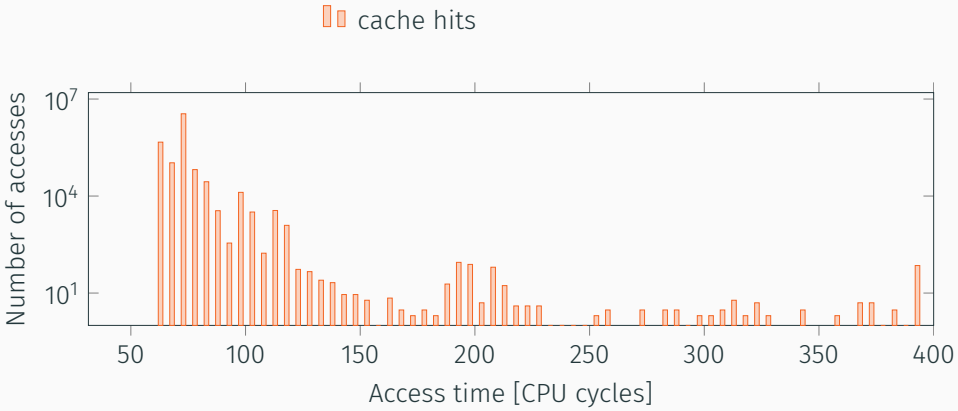


Data loaded in a specific set depending on its address

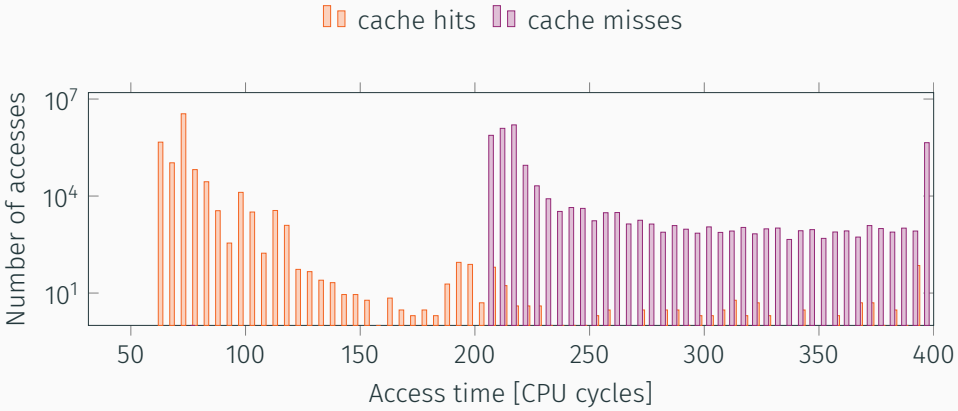
Several ways per set

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

Timing differences



Timing differences



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.

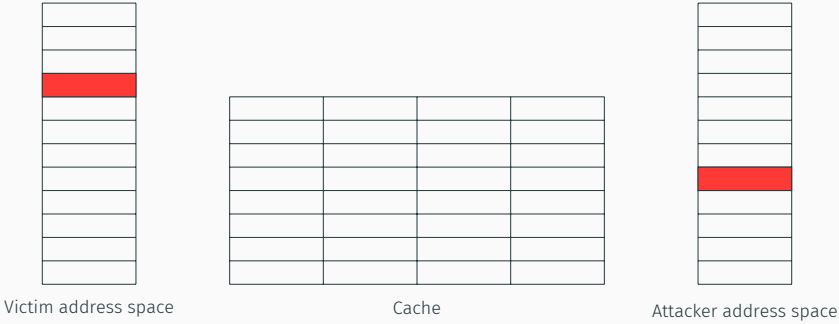
Y. Yarom et al. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014.

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.

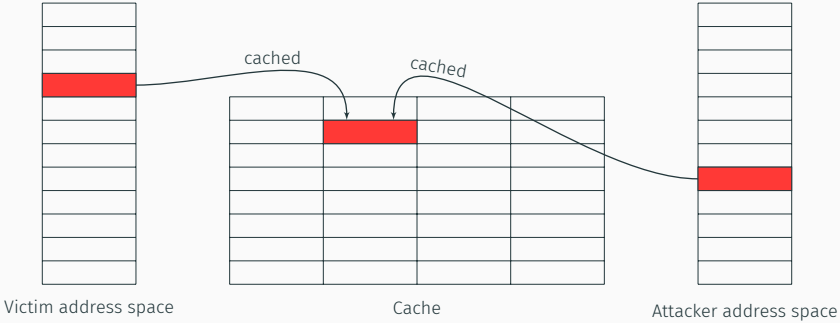
F. Liu et al. “Last-Level Cache Side-Channel Attacks are Practical”. In: *S&P’15*. 2015.

Cache attacks: Flush+Reload



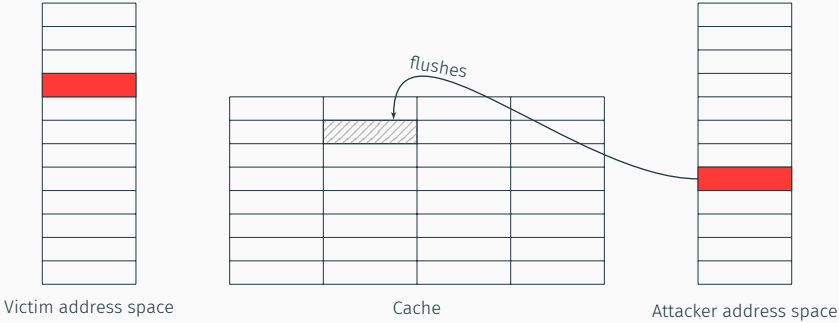
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attacks: Flush+Reload



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

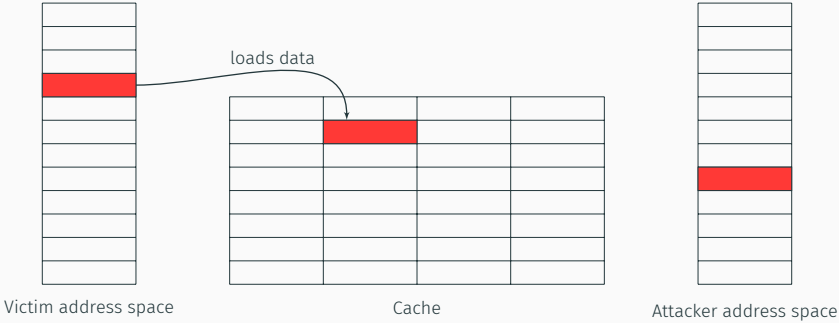
Cache attacks: Flush+Reload



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

Step 2: Attacker flushes the shared cache line

Cache attacks: Flush+Reload

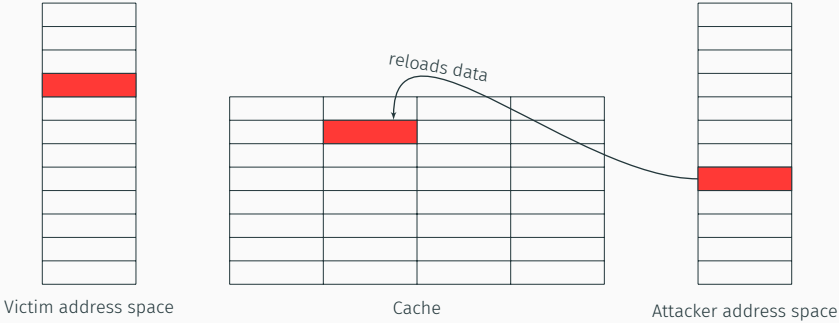


Step 1: Attacker maps shared library (shared memory, in cache)

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)

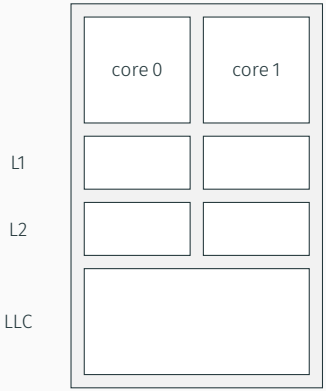
Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Step 4: Attacker reloads the data

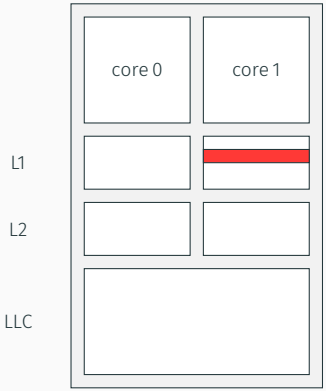
What if there is no shared memory?

Inclusive property



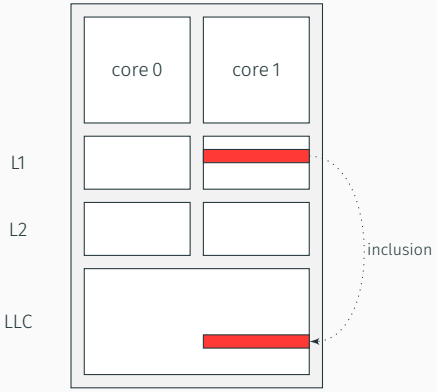
- inclusive LLC: superset of L1 and L2

Inclusive property



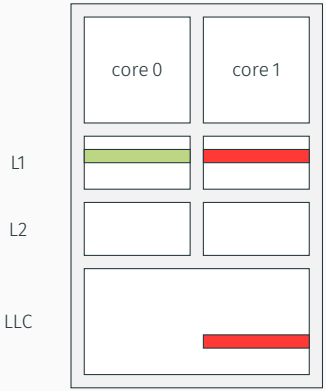
- inclusive LLC: superset of L1 and L2

Inclusive property



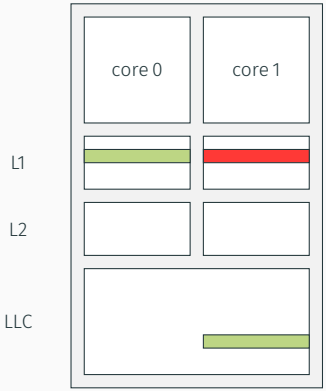
- inclusive LLC: superset of L1 and L2

Inclusive property



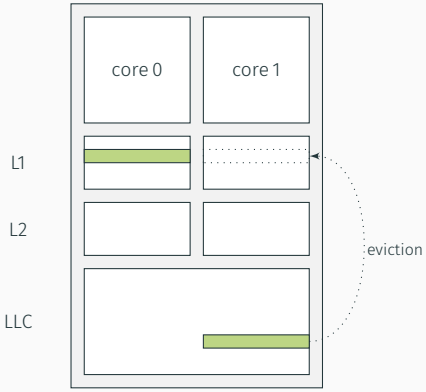
- inclusive LLC: superset of L1 and L2

Inclusive property



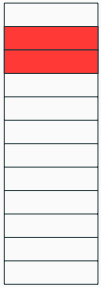
- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2

Inclusive property

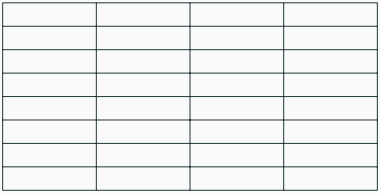


- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
- a core can evict lines in the private L1 of another core

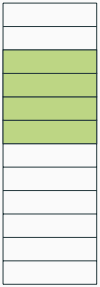
Cache attacks: Prime+Probe



Victim address space

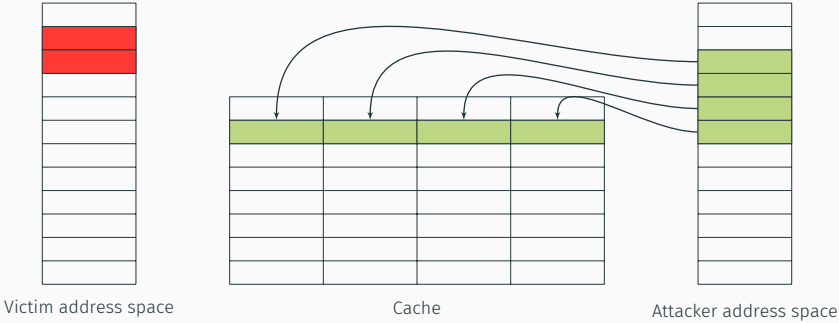


Cache



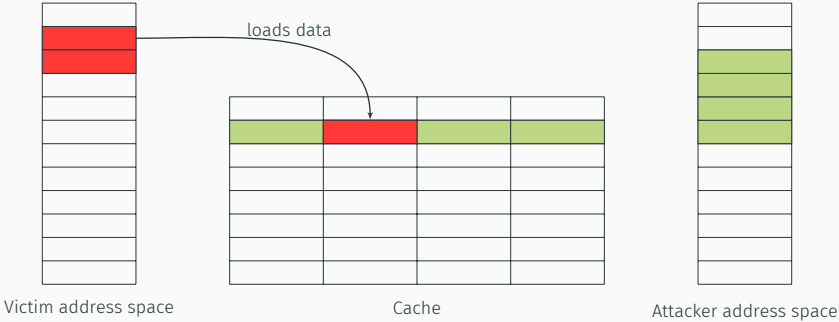
Attacker address space

Cache attacks: Prime+Probe



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

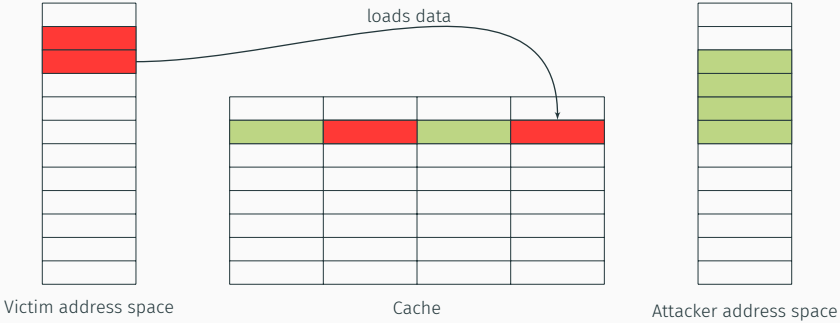
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

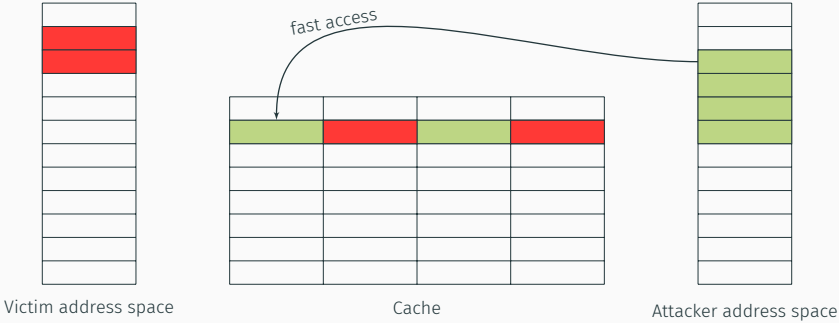
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

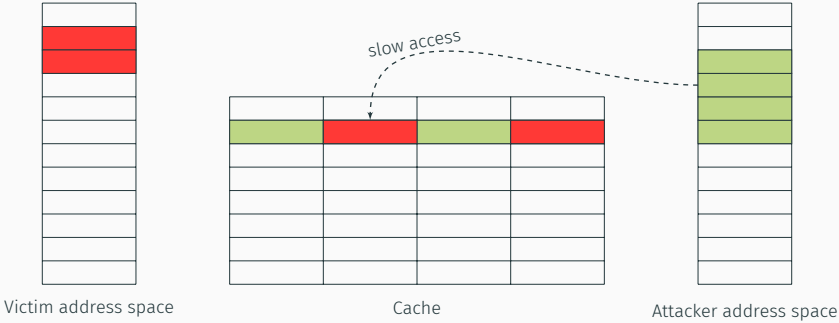


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

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

Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

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

Challenges with Prime+Probe

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

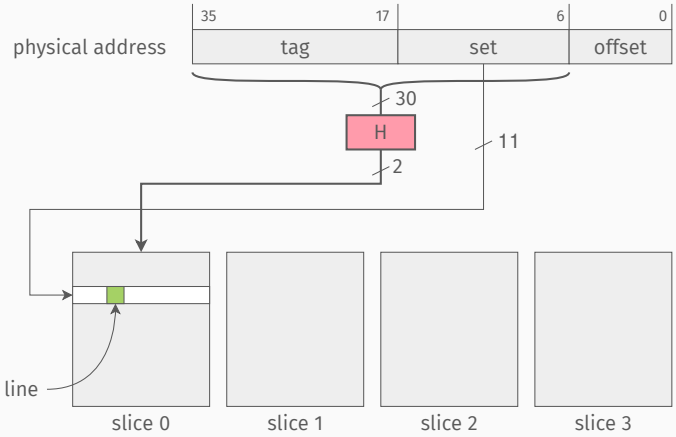
1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?

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



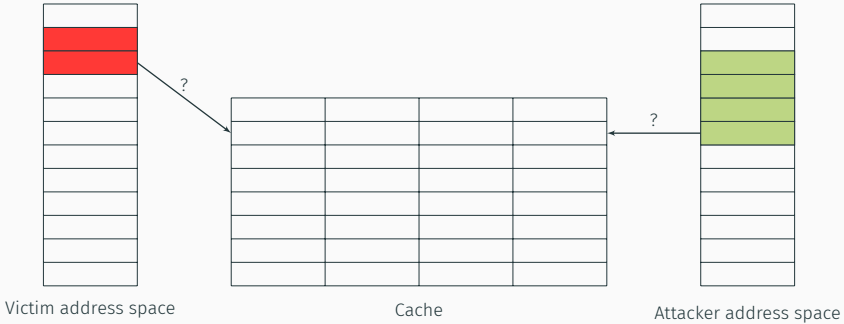
Last-level cache addressing

- 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 processors?

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



Reverse-engineering 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)

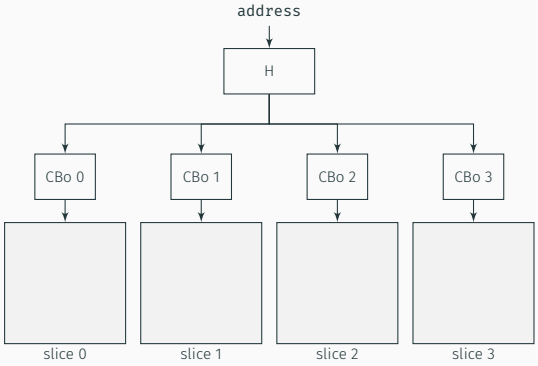
How to map addresses to slices?

- 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

Mapping with performance counters

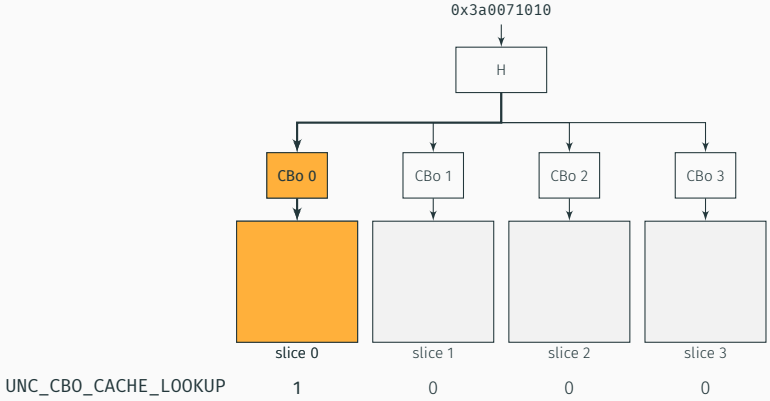
- event UNC_CBO_CACHE_LOOKUP counts accesses to a slice



UNC_CBO_CACHE_LOOKUP	0	0	0	0

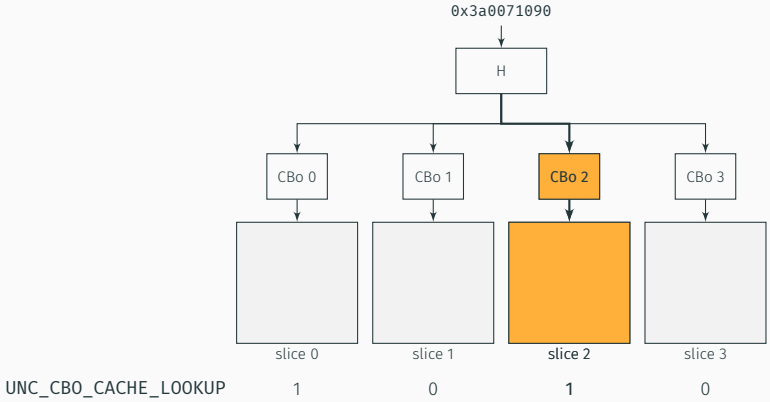
Mapping with performance counters

- event UNC_CBO_CACHE_LOOKUP counts accesses to a slice



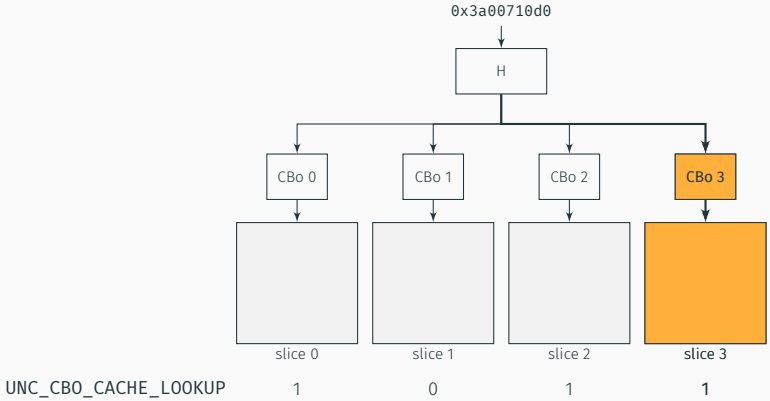
Mapping with performance counters

- 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

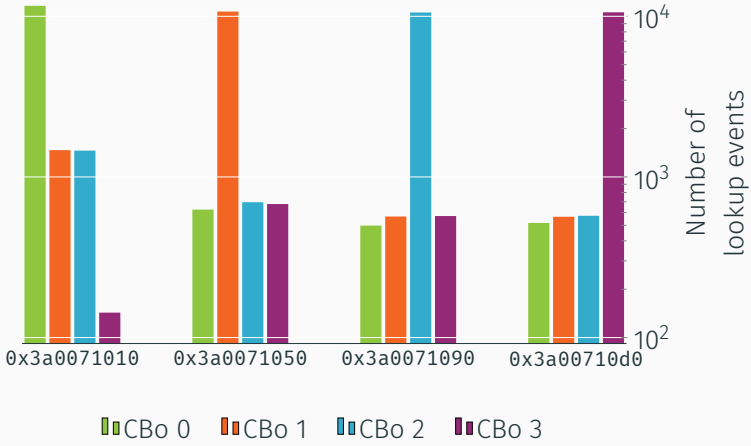
Mapping a physical address to a slice

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



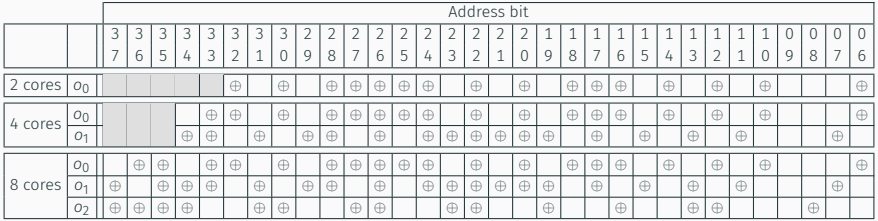
Inferring the function

Two cases:

1. 2^n number of cores: linear function $\rightarrow$ XORs of address bits
   - solve the linear equation
   - or brute force (not that long)
2. the remainder: non-linear function

Last-level cache linear functions

3 functions, depending on the number of cores



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

Practical applications

Covert channels in the cloud

- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs

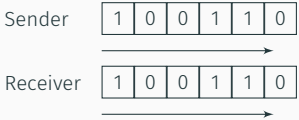
Covert channels in the cloud

- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- literature: stops working with noise on the machine

Covert channels in the cloud

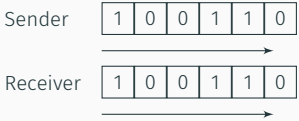
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- literature: stops working with noise on the machine
- solution? “Just use error-correcting codes”

Why can’t we just use error correcting codes?

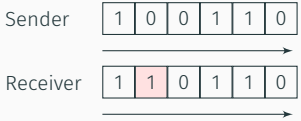


(a) Transmission without errors

Why can’t we just use error correcting codes?

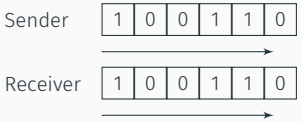


(a) Transmission without errors

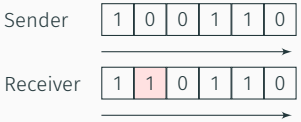


(b) Noise: substitution error

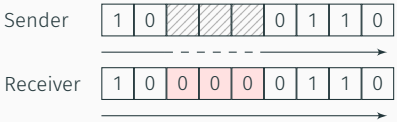
Why can't we just use error correcting codes?



(a) Transmission without errors

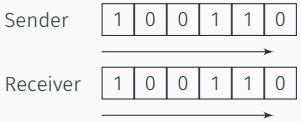


(b) Noise: substitution error

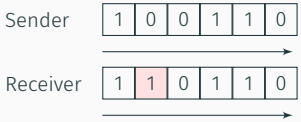


(c) Sender descheduled: insertions

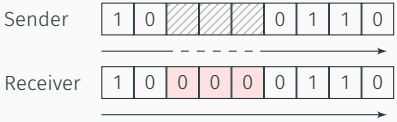
Why can't we just use error correcting codes?



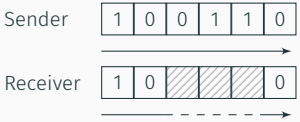
(a) Transmission without errors



(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

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

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
  - $\rightarrow$ lines of the receiver still in cache $\rightarrow$ 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

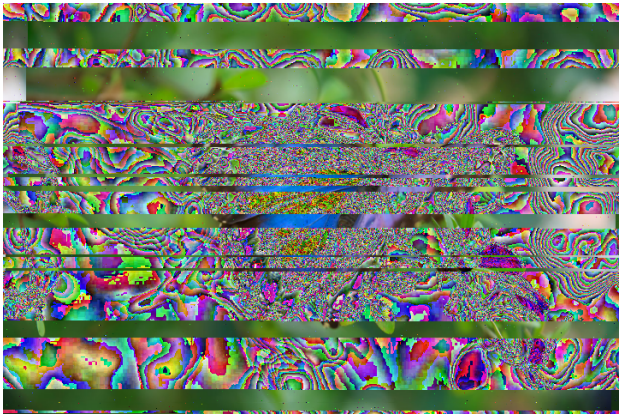
Eviction set generation

- 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

→ we use a jamming agreement

Sending the first image



Handling synchronization errors

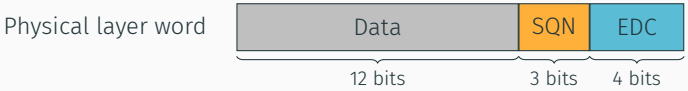
Handling synchronization errors

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

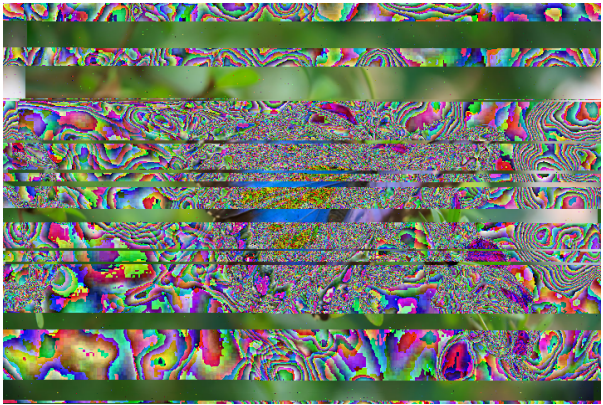
Handling synchronization errors

- 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)

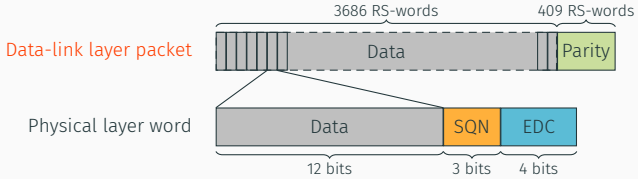
Synchronization (after)

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`

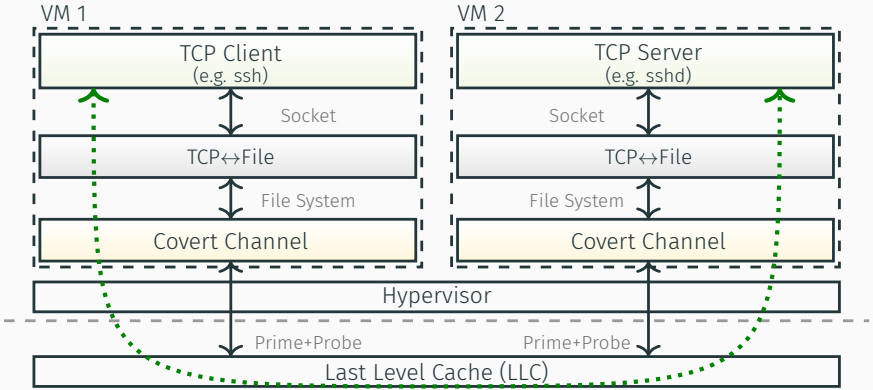
Evaluation

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`

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 $\rightarrow$ 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

→ there are other sources of timing

→ 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
  → 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
  $\rightarrow$ if applied to LLC $\rightarrow$ same as no cache, disastrous performance
- page coloring $\rightarrow$ 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

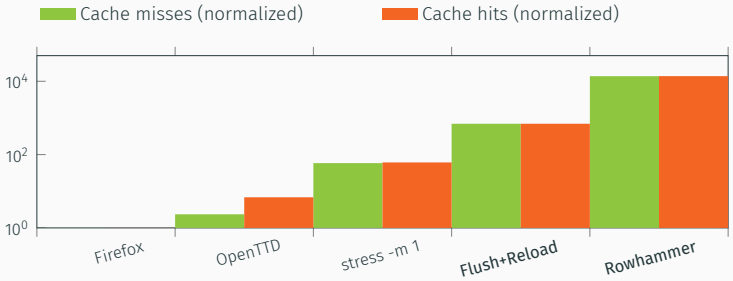
B. C. Vattikonda et al. "Eliminating fine grained timers in Xen". In: *CCSW'11*. 2011

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: Write better code

- no branch or data access depending on a secret
- hardware implementations (AES-NI, etc.)

Application level: Write better code

- no branch or data access depending on a secret
- 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, CVE-2016-7440, CVE-2016-7439, CVE-2016-7438, CVE-2018-0737, ...

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

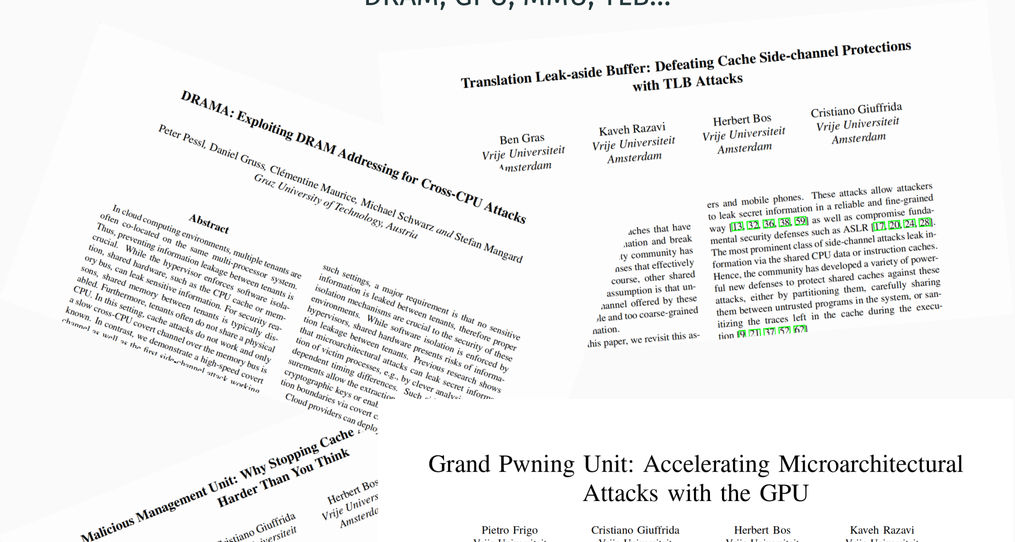
O. Acıiçmez et al. “Predicting secret keys via branch prediction”. In: *CT-RSA 2007*. 2007

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

It’s not just caches!

DRAM, GPU, MMU, TLB...



DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks

Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz *and* Stefan Mangard
Graz University of Technology, Austria

Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks

Ben Gras, *Vrije Universiteit Amsterdam*; Kaveh Razavi, *Vrije Universiteit Amsterdam*; Herbert Bos, *Vrije Universiteit Amsterdam*; Cristiano Giuffrida, *Vrije Universiteit Amsterdam*

Malicious Management Unit: Why Stopping Cache Attacks in Software is Harder Than You Think

Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU

Pietro Frigo, Cristiano Giuffrida, Herbert Bos, Kaveh Razavi

Conclusion

- more a problem of CPU design than Instruction Set Architecture

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- it's also not just the cache

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- it's also not just the cache
- hard to patch $\rightarrow$ issues linked to performance optimizations

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- it's also not just the cache
- hard to patch $\rightarrow$ issues linked to performance optimizations
- we would like to keep the optimizations without the attacks

Conclusion

- more a problem of CPU design than Instruction Set Architecture
- it's also not just the cache
- hard to patch $\rightarrow$ issues linked to performance optimizations
- we would like to keep the optimizations without the attacks
- very interesting and active field of research!

Questions?

Contact

clementine.maurice@irisa.fr

@BloodyTangerine

Introduction to cache side-channel attacks

Clémentine Maurice, CNRS, IRISA

October 9, 2018—Séminaire du DIT, ENS Rennes

References i

O. Acıiçmez, J.-P. Seifert, and c. K. Koç. “Predicting secret keys via branch prediction”. In: *CT-RSA 2007*. 2007.

M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham. “On subnormal floating point and abnormal timing”. In: *S&P’15*. 2015.

G. Doychev, D. Feld, B. Köpf, L. Mauborgne, and J. Reineke. “CacheAudit: A Tool for the Static Analysis of Cache Side Channels”. In: *USENIX Security Symposium*. 2013.

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

D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh. “Jump over ASLR: Attacking branch predictors to bypass ASLR”. In: *MICRO’16*. 2016.

P. Frigo, C. Giuffrida, H. Bos, and K. Razavi. “Grand Pwning unit: accelerating microarchitectural attacks with the GPU”. In: *S&P 2018*. 2018.

References ii

A. Fuchs and R. B. Lee. “Disruptive Prefetching: Impact on Side-Channel Attacks and Cache Designs”. In: *SYSTOR’15*. 2015.

B. Gras, K. Razavi, H. Bos, and C. Giuffrida. “Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks”. In: *USENIX Security Symposium*. 2018.

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

D. Gruss, C. Maurice, K. Wagner, and S. Mangard. “Flush+Flush: A Fast and Stealthy Cache Attack”. In: *DIMVA’16*. 2016.

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

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

G. Irazoqui, T. Eisenbarth, and B. Sunar. “Cross processor cache attacks”. In: *AsiaCCS’16*. 2016.

References iii

Y. Jang, S. Lee, and T. Kim. “Breaking kernel address space layout randomization with intel TSX”. In: *CCS’16*. 2016.

J. Kong, O. Acıiçmez, J.-P. Seifert, and H. Zhou. “Hardware-software integrated approaches to defend against software cache-based side channel attacks”. In: *HPCA’09*. 2009.

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

C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon. “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: *RAID’15*. 2015.

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

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

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

References iv

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

P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security Symposium*. 2016.

H. Raj, R. Nathuji, A. Singh, and P. England. “Resource Management for Isolation Enhanced Cloud Services”. In: *CCSW’09*. 2009.

M. Schwarz, C. Maurice, D. Gruss, and S. Mangard. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC’17*. 2017.

S. van Schaik, K. Razavi, C. Giuffrida, and H. Bos. “Malicious Management Unit: Why Stopping Cache Attacks in Software is Harder Than You Think”. In: *USENIX Security Symposium*. 2018.

B. C. Vattikonda, S. Das, and H. Shacham. “Eliminating fine grained timers in Xen”. In: *CCSW’11*. 2011.

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

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

References v

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