Évolution des attaques sur la micro-architecture

Clémentine Maurice, CNRS, IRISA

23 Janvier 2019–Journée "Nouvelles Avancées en Sécurité des Systèmes d'Information", INSA de Toulouse/LAAS-CNRS

- hardware usually modeled as an abstract layer behaving correctly
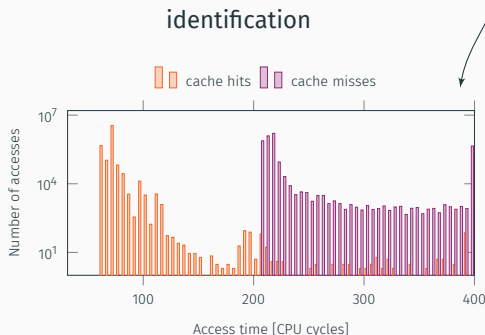
- hardware usually modeled as an abstract layer behaving correctly, but possible attacks

- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing hardware errors
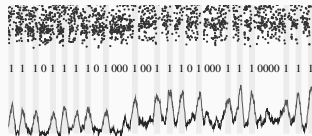 - side channels: observing side effects of hardware on computations

- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing hardware errors
  - side channels: observing side effects of hardware on computations
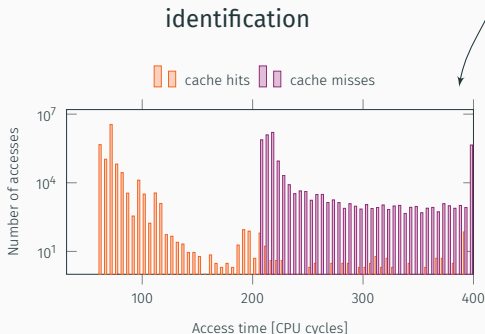
identification

- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing hardware errors
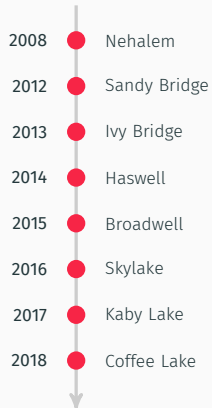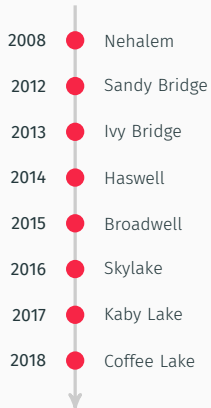 - side channels: observing side effects of hardware on computations

identification

attack

$\rightarrow$

- retrieving secret keys, keystroke timings
- bypassing OS security (ASLR)

2

From small optimizations...

Year	Microarchitecture
2008	Nehalem
2012	Sandy Bridge
2013	Ivy Bridge
2014	Haswell
2015	Broadwell
2016	Skylake
2017	Kaby Lake
2018	Coffee Lake

- new microarchitectures yearly

From small optimizations...

Year	Microarchitecture
2008	Nehalem
2012	Sandy Bridge
2013	Ivy Bridge
2014	Haswell
2015	Broadwell
2016	Skylake
2017	Kaby Lake
2018	Coffee Lake

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

From small optimizations...

2008	Nehalem
2012	Sandy Bridge
2013	Ivy Bridge
2014	Haswell
2015	Broadwell
2016	Skylake
2017	Kaby Lake
2018	Coffee Lake
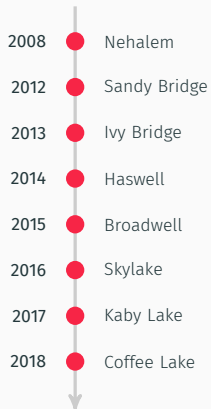
- new microarchitectures yearly
- performance improvement $\approx 5\%$
- very small optimizations: caches, branch prediction...

- microarchitectural side channels come from these optimizations

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
- attacker infers information from a victim process via hardware usage

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
- attacker infers information from a victim process via hardware usage
- pure-software attacks by unprivileged processes

- microarchitectural side channels come from these optimizations
- several processes are sharing microarchitectural components
- attacker infers information from a victim process via hardware usage
- pure-software attacks by unprivileged processes
- sequences of benign-looking actions → hard to detect

Historical recap of past attacks

Historical recap of past attacks

Recent advances

Historical recap of past attacks

Recent advances

Future and challenges

Historical Recap

From theoretical to practical cache attacks

- first theoretical attack in 1996 by Kocher
- first practical attack on RSA in 2005 by Percival, on AES in 2006 by Osvik et al.
- renewed interest for the field in 2014 after Flush+Reload by Yarom and Falkner

P. C. Kocher. "Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and Other Systems". In: *Crypto'96*. 1996.

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

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

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

- threads sharing one core share resources: L1, L2 cache, branch predictor
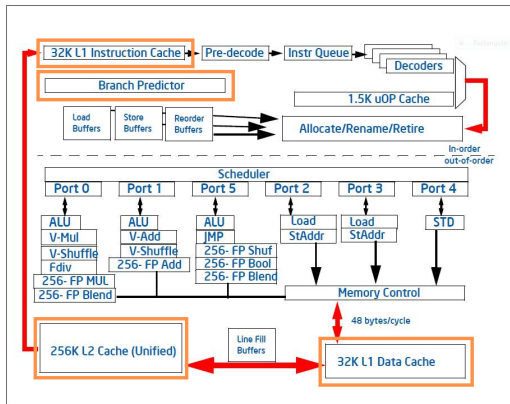
Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

Possible side channels using

components shared by a core?

Possible side channels using

components shared by a core?

Stop sharing a core!

Caches on Intel CPUs
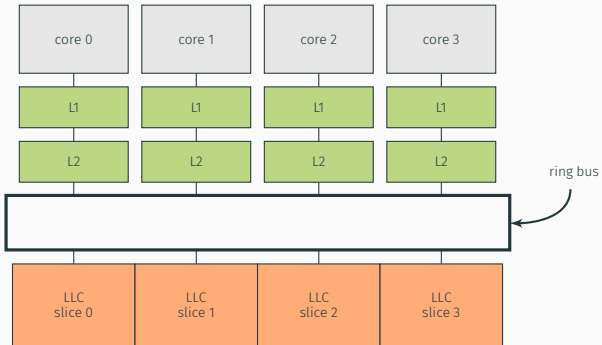
Caches on Intel CPUs
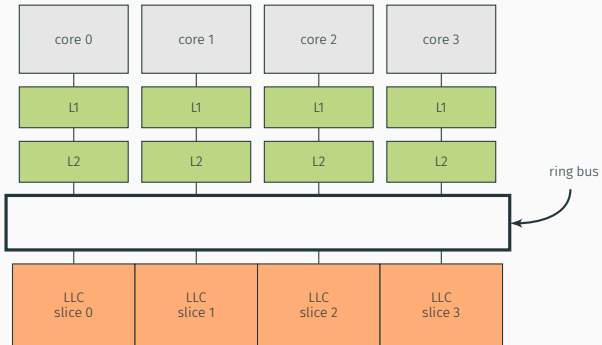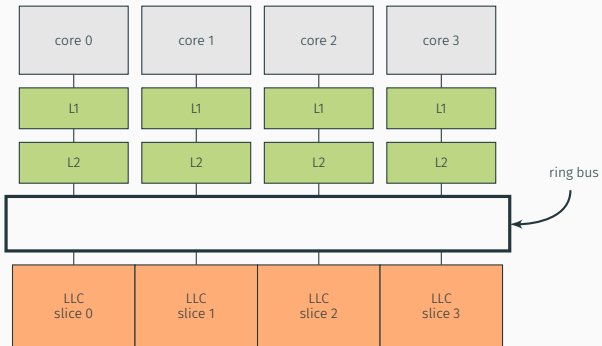

- L1 and L2 are private

Caches on Intel CPUs
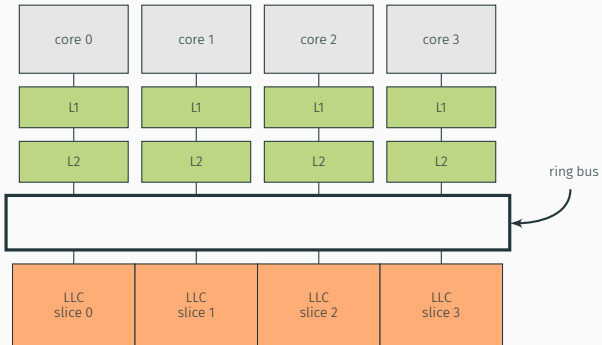
- L1 and L2 are private
- last-level cache
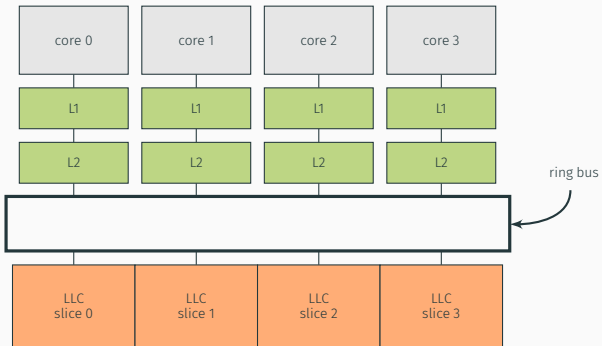
- L1 and L2 are private
- last-level cache
  - divided in slices

Caches on Intel CPUs

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

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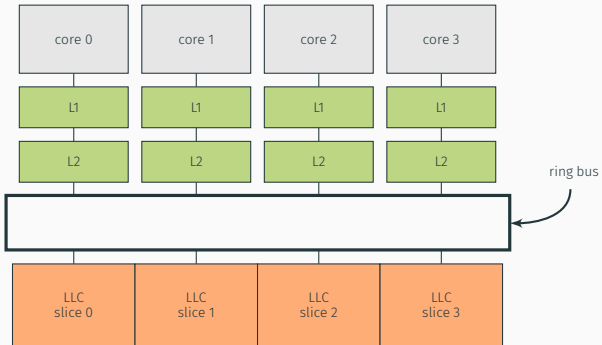
- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive

Set-associative caches

Address

0	16 17	25 26	31
	Index	Offset	

Cache

Set-associative caches
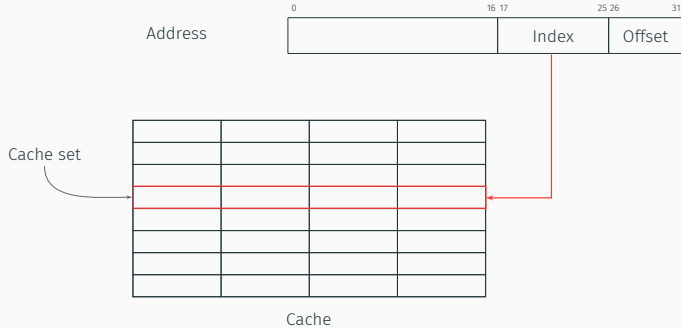
Data loaded in a specific set depending on its address

Set-associative caches
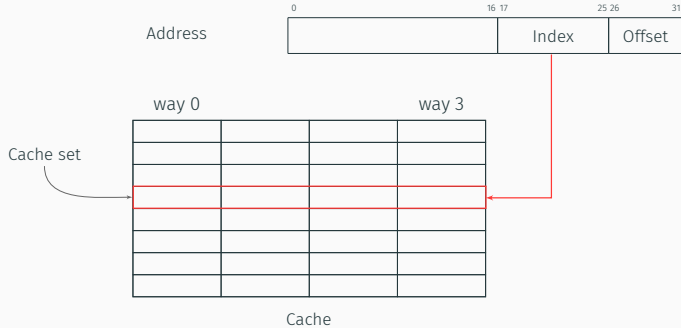
Data loaded in a specific set depending on its address

Several ways per set

Set-associative caches
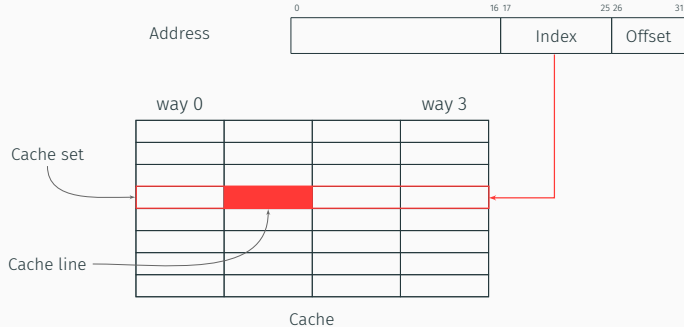
Data loaded in a specific set depending on its address

Several ways per set

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

Cache attacks

- caches improve performance

Cache attacks

- caches improve performance
- SRAM is expensive $\rightarrow$ small caches

Cache attacks

- caches improve performance
- SRAM is expensive $\rightarrow$ small caches
- different timings for memory accesses

Cache attacks

- caches improve performance
- SRAM is expensive $\rightarrow$ small caches
- different timings for memory accesses
    1. data is cached $\rightarrow$ cache hit $\rightarrow$ fast

Cache attacks

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
  1. data is cached → cache hit → fast
  2. data is not cached → cache miss → slow

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
  1. data is cached → cache hit → fast
  2. data is not cached → cache miss → slow
- cache attacks leverage this timing difference

Timing differences

Victim address space   Cache   Attacker address space

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

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

Victim address space   Cache   Attacker address space

flushes

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

Step 2: Attacker flushes the shared cache line

loads data

Victim address space   Cache   Attacker address space

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

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

reloads data

Victim address space

Cache

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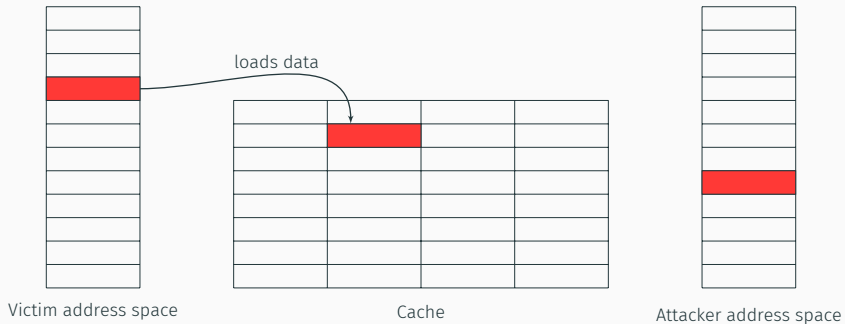
Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Step 4: Attacker reloads the data

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

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

B. Gülmezoglu, M. S. Inci, T. Eisenbarth, and B. Sunar. "A Faster and More Realistic Flush+Reload Attack on AES". In: *Constructive Side-Channel Analysis and Secure Design (COSADE)*. 2015

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

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

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

- Cache Template Attacks: automatically finds information leakage
  - → side channel on keystrokes and AES T-tables implementation

---

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

B. Gülmezoglu, M. S. Inci, T. Eisenbarth, and B. Sunar. "A Faster and More Realistic Flush+Reload Attack on AES". In: *Constructive Side-Channel Analysis and Secure Design (COSADE)*. 2015

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

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

- fine granularity: 1 cache line (64 Bytes)

- fine granularity: 1 cache line (64 Bytes)
- but requires shared memory

- fine granularity: 1 cache line (64 Bytes)
- but requires shared memory
→ memory deduplication between VMs

Possible side channels using

memory deduplication?

Possible side channels using

memory deduplication?

Disable memory deduplication!

Inclusive property

- inclusive LLC: superset of L1 and L2

- inclusive LLC: superset of L1 and L2

- inclusive LLC: superset of L1 and L2

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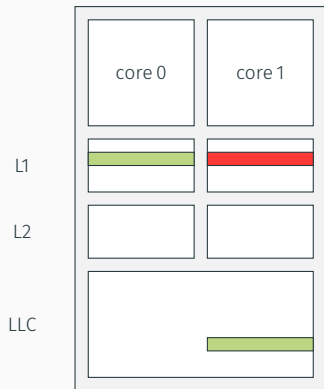
- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
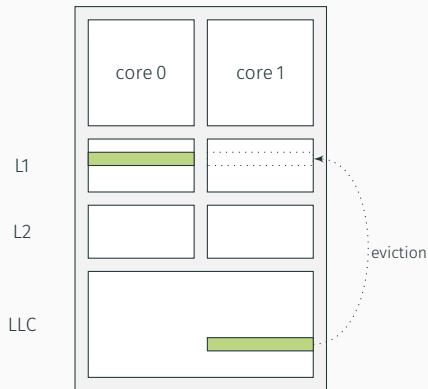
- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
- a core can evict lines in the private L1 of another core

Victim address space

Cache

Attacker address space

Victim address space   Cache   Attacker address space

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

loads data

Victim address space

Cache

Attacker address space

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

Step 2: Victim evicts cache lines while running

loads data

Victim address space

Cache

Attacker address space

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

Step 2: Victim evicts cache lines while running

Victim address space   Cache   Attacker address space

fast access

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

Victim address space   Cache   Attacker address space

slow access

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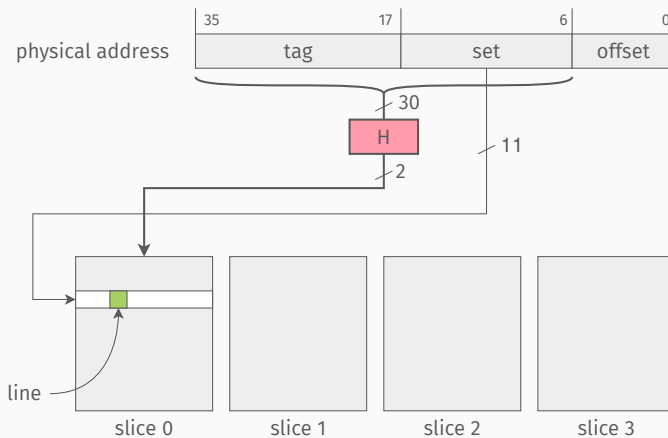
- last-level cache $\rightarrow$ as many slices as cores
- **undocumented** hash function that maps a physical address to a slice
- designed for performance

For 2^k slices:

physical address
30 bits

$\longrightarrow$ | H | $\longrightarrow$

slice $(o_0, \ldots, o_{k-1})$
k bits

Undocumented function → impossible to target a set
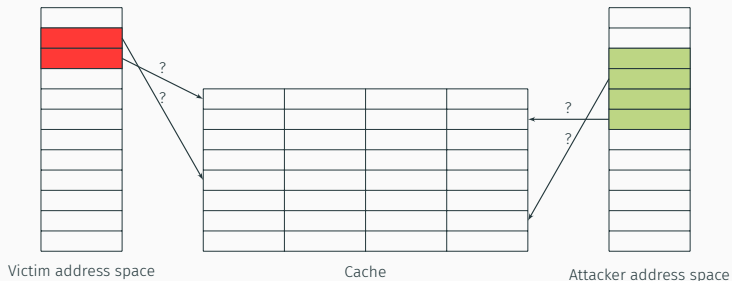
Victim address space   Cache   Attacker address space

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

Undocumented function → impossible to target a set

Victim address space   Cache   Attacker address space
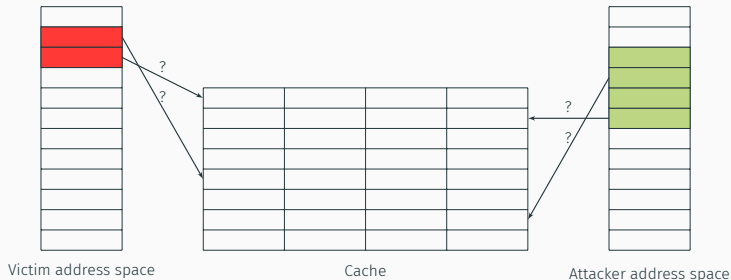
→ We reverse-engineered the function!

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

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

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

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

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

Possible side channels using
components shared by a CPU?

Possible side channels using

components shared by a CPU?

Stop sharing a CPU!?

Recent Advances

Building practical attacks

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

- 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 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"
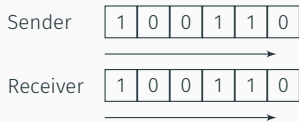
(a) Transmission without errors

Why can't we just use error correcting codes?
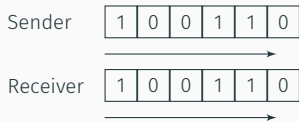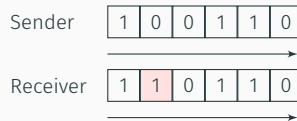
(a) Transmission without errors

(b) Noise: substitution error
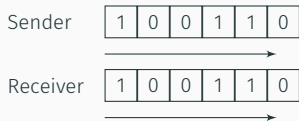
(a) Transmission without errors
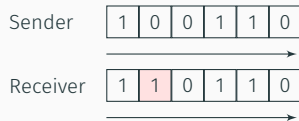
(b) Noise: substitution error
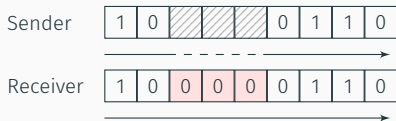
(c) Sender descheduled: insertions

Why can't we just use error correcting codes?
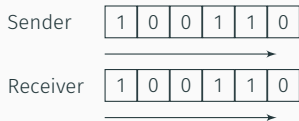
(a) Transmission without errors
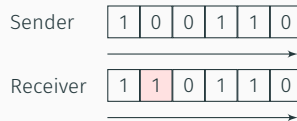
(b) Noise: substitution error
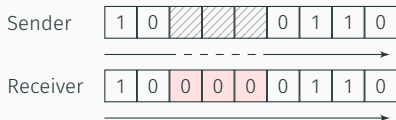
(c) Sender descheduled: insertions
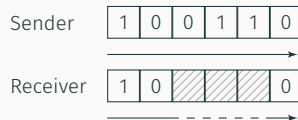
(d) Receiver descheduled: deletions

- 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, 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*. to appear. 2017

- 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

- 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 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

- 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

- 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

physical address

cache tag

xxxx

cache set index

cache line offset

2MB page offset

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

- 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
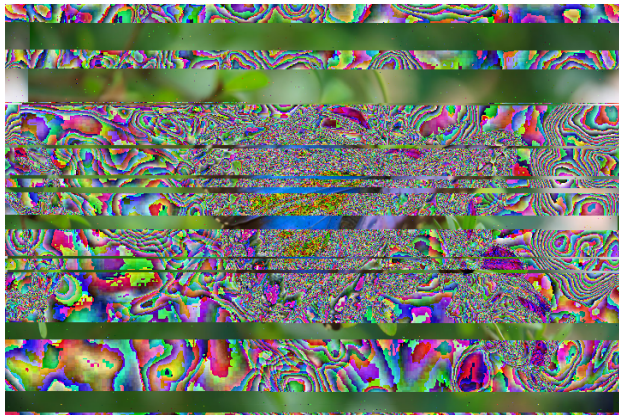
Physical layer word

Data

12 bits

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

Physical layer word

Data	SQN
12 bits	3 bits

- 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
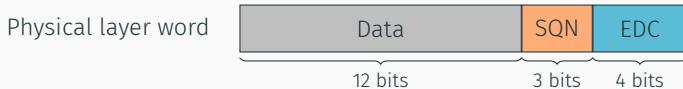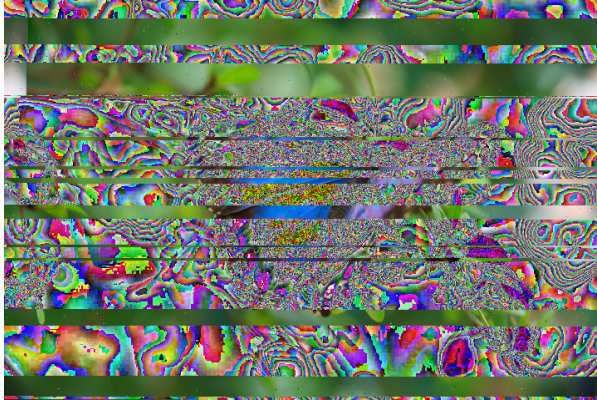
Physical layer word

Data	SQN	EDC
12 bits	3 bits	4 bits

- Reed-Solomon codes to correct the remaining errors

- 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
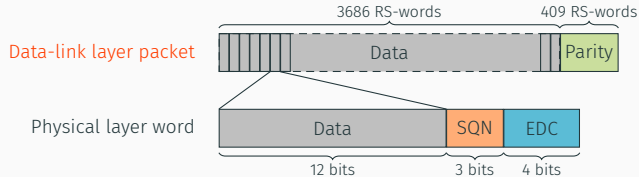
Evaluation

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
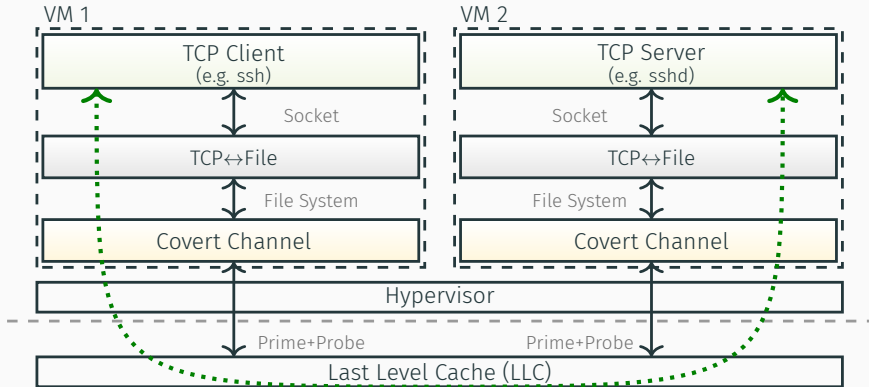
SSH evaluation

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`

Increasing the attack surface

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

DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks

Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz and Stefan Mangard

Graz University of Technology, Austria

Abstract

In cloud computing environments, multiple tenants are often co-located on the same multi-processor system. Thus, preventing the information leakage between tenants is crucial. While the hypervisor enforces software isolation, shared hardware, such as the CPU cache or memory bus, can leak sensitive information. For security reasons, shared memory between tenants is typically disabled. Furthermore, tenants often do not share a physical CPU. In this setting, tenants do not work and only a slow cross-CPU covert channel over the memory bus is known. In contrast, we channel over the memory bus is channel as well as the first side-channel attack working

Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU

Pietro Frigo
Vrije Universiteit
Amsterdam
p.frigo@vu.nl

Cristiano Giuffrida
Vrije Universiteit
Amsterdam
giuffrida@cs.vu.nl

Herbert Bos
Vrije Universiteit
Amsterdam
herbertb@cs.vu.nl

Kaveh Razavi
Vrije Universiteit
Amsterdam
kaveh@cs.vu.nl

Malicious Management Unit: Why Stopping Cache Harder Than You Think

Herbert Bos
Vrije Universiteit
Amsterdam

Cristiano Giuffrida
Vrije Universiteit
Amsterdam

ARMageddon: Cache Attacks on Mobile Devices

Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard
Graz University of Technology, Austria

Fantastic Timers and Where to Find Them:
High-Resolution Microarchitectural
Attacks in JavaScript

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

**The Spy in the Sandbox: Practical Cache Attacks in
JavaScript and their Implications**

Yossef Oren   Vasileios P. Kemerlis   Simha Sethumadhavan   Angelos D. Keromytis
Columbia University
Department of Computer Science
{yos, vpk, simha, angelos}@cs.columbia.edu

Practical Keystroke
Sandboxed Jav.

Moritz Lipp, Daniel Gruss, Michael Schwarz,
Clémentine Maurice, and Stefan Manga
Graz University of Technology, Austria

42

Future and Challenges

Challenges and questions

- lack of documentation on microarchitectural components
- which components are vulnerable to these attacks?
- which software is vulnerable to these attacks?
- how to prevent attacks based on performance optimizations without removing performance?

It's not just code that is executed!

- Meltdown breaks isolation between applications and kernel by exploiting Out-of-Order execution
- Spectre mistrains branch prediction to speculatively execute code that should not be executed
- 3 initial variants in January, as of today 21 variants

C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss. "A Systematic Evaluation of Transient Execution Attacks and Defenses". In: *arXiv preprint arXiv:1811.05441* (2018)

Conclusion

- first paper by Kocher in 1996: 22 years of research in this area
- domain still in expansion: increasing number of papers published since 2015
- adopted countermeasures mainly target cryptographic implementations
- still a lot more to discover on this iceberg :)
- quick fixes don't work
- still a lot more work needed to find satisfying countermeasures

Thank you!

Contact

✉ clementine.maurice@irisa.fr

🐦 @BloodyTangerine

Évolution des attaques sur la micro-architecture

Clémentine Maurice, CNRS, IRISA

23 Janvier 2019–Journée "Nouvelles Avancées en Sécurité des Systèmes d'Information", INSA de Toulouse/LAAS-CNRS