

Evolution of microarchitectural attacks

Clémentine Maurice, CNRS, IRISA

December 11, 2018–WOS 8

Attacks on micro-architecture

- hardware usually modeled as an abstract layer behaving correctly

Attacks on micro-architecture

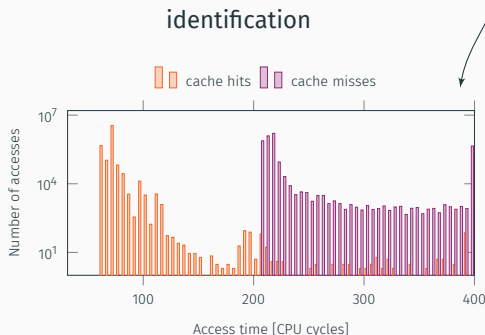
- **hardware** usually modeled as an abstract layer behaving correctly, but possible attacks

Attacks on micro-architecture

- 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

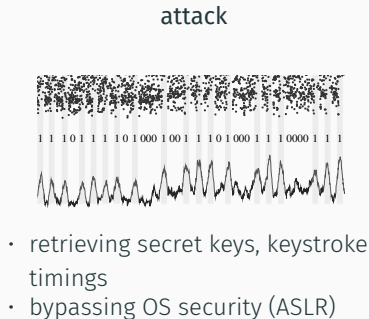
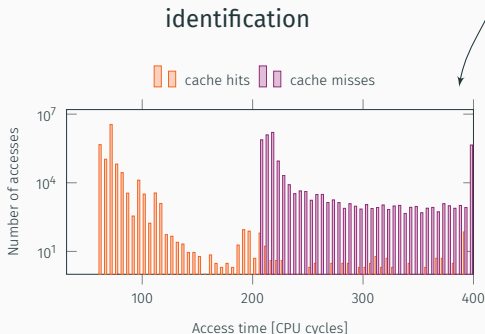
Attacks on micro-architecture

- **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

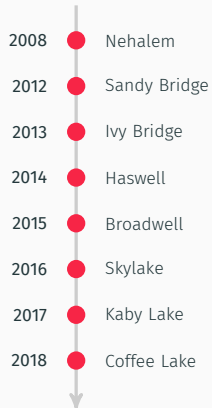


Attacks on micro-architecture

- **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

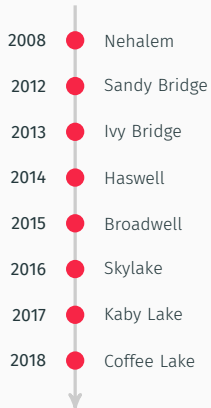


From small optimizations...



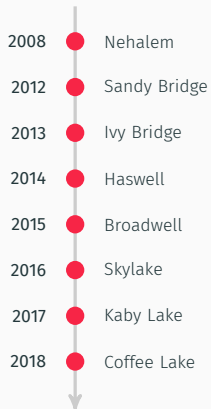
- new microarchitectures yearly

From small optimizations...



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

From small optimizations...



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

... To microarchitectural side-channel attacks

- microarchitectural side channels come from these optimizations

... To microarchitectural side-channel attacks

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

... To microarchitectural side-channel attacks

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

... To microarchitectural side-channel attacks

- 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

... To microarchitectural side-channel attacks

- 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 Diffie-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.

Hyper-threading: Same-core attacks

- threads sharing one core **share resources**: L1, L2 cache, branch predictor

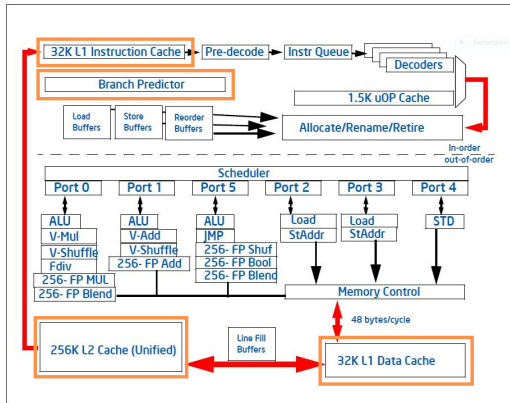


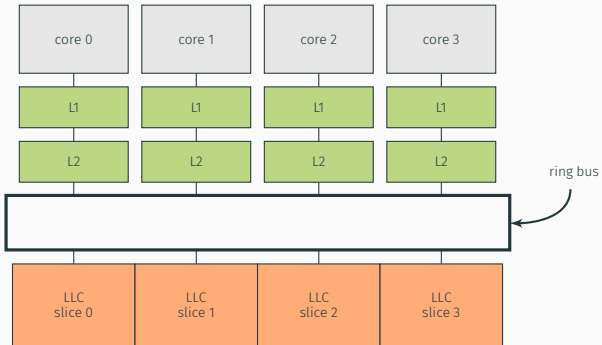
Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

Possible side channels using
components shared by a core?

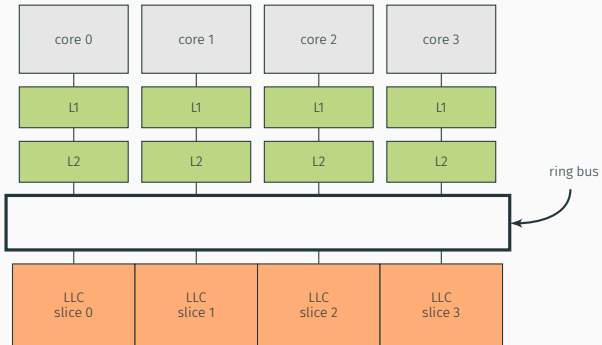
Possible side channels using
components shared by a core?

Stop sharing a core!

Caches on Intel CPUs

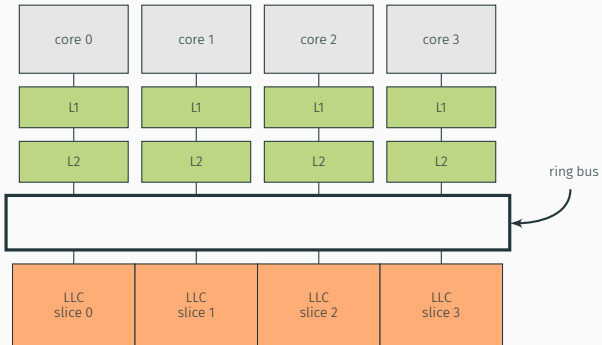


Caches on Intel CPUs



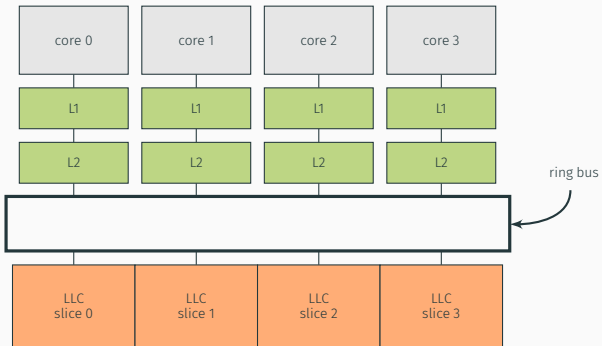
- L1 and L2 are private

Caches on Intel CPUs



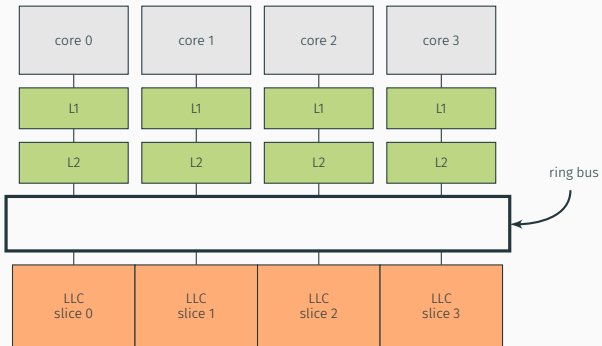
- L1 and L2 are private
- last-level cache

Caches on Intel CPUs



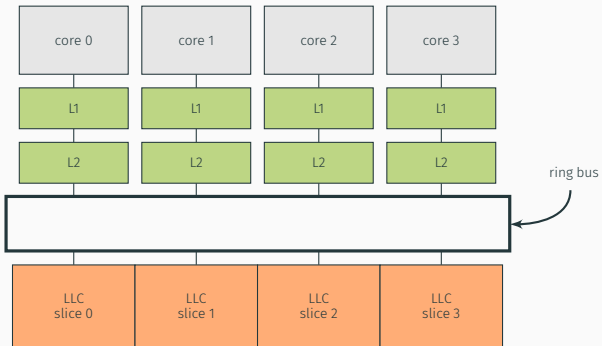
- L1 and L2 are private
- last-level cache
 - divided in **slices**

Caches on Intel CPUs



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

Caches on Intel CPUs



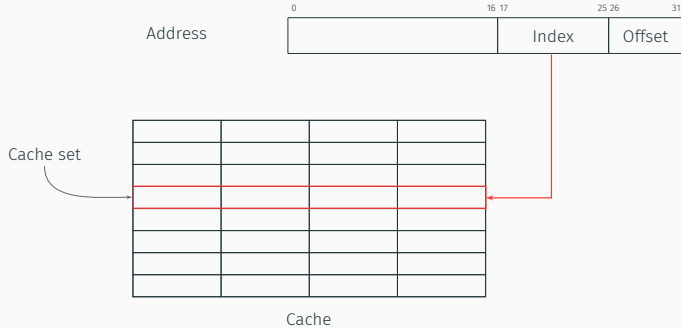
- L1 and L2 are private
- last-level cache
 - divided in **slices**
 - **shared** across cores
 - **inclusive**

Set-associative caches



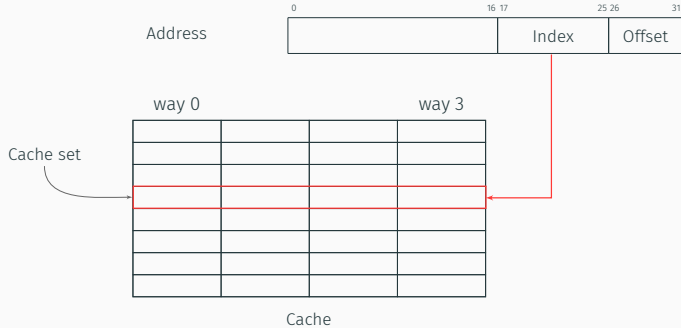
Cache

Set-associative caches



Data loaded in a specific **set** depending on its address

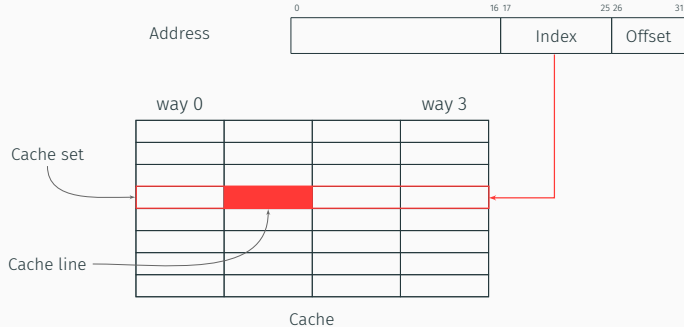
Set-associative caches



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

- caches improve performance

Cache attacks

- caches improve performance
- SRAM is expensive → small caches

Cache attacks

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses

Cache attacks

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

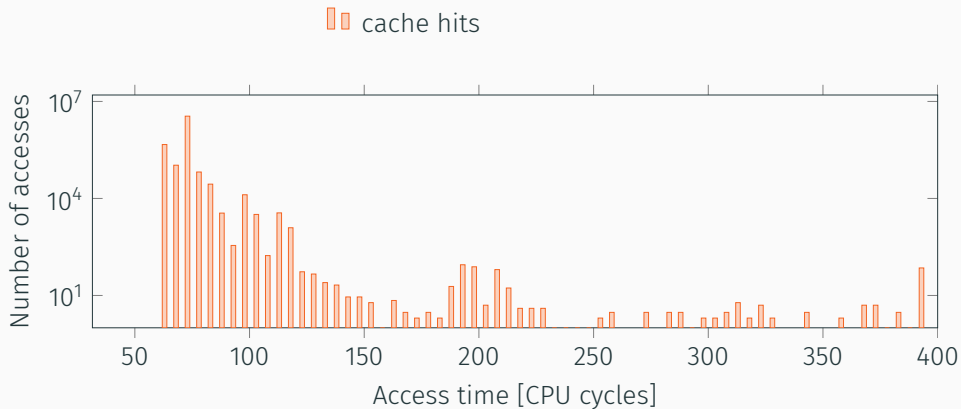
Cache attacks

- 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**

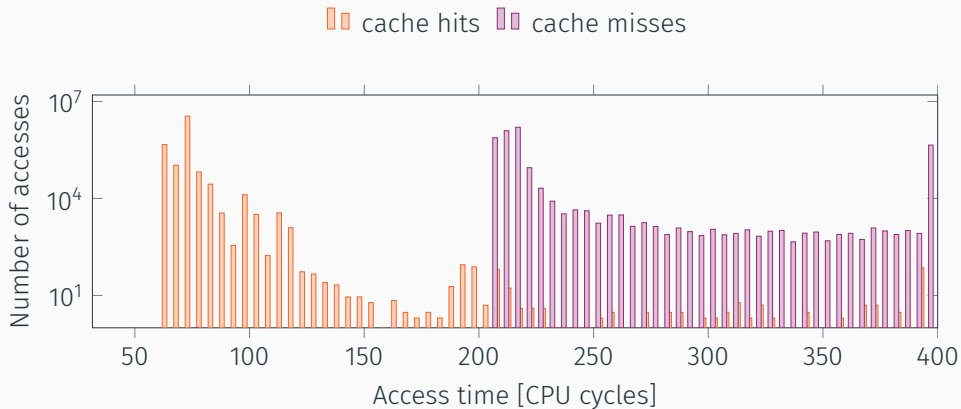
Cache attacks

- 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**
- **cache attacks** leverage this timing difference

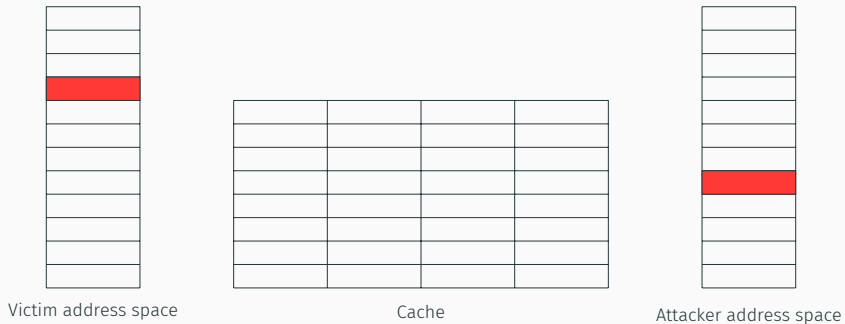
Timing differences



Timing differences

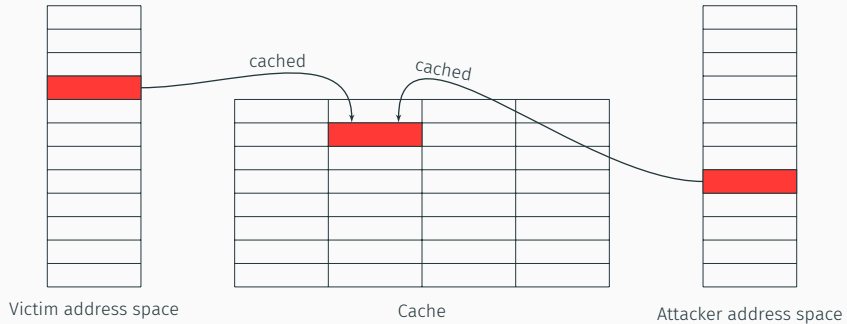


Cache attacks: Flush+Reload



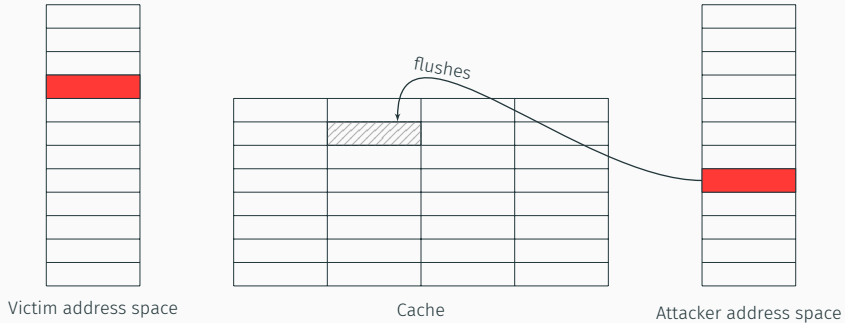
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attacks: Flush+Reload



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

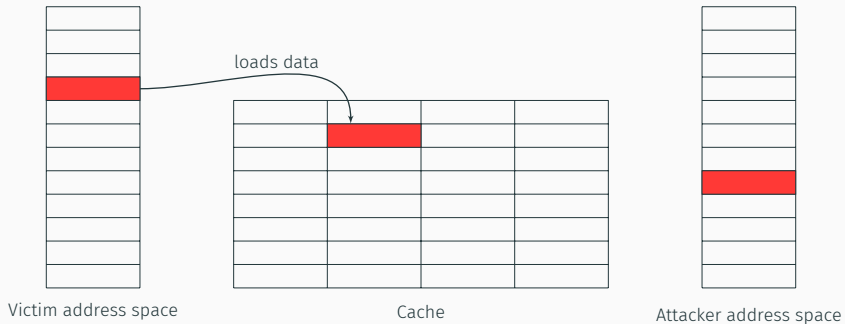
Cache attacks: Flush+Reload



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

Step 2: Attacker **flushes** the shared cache line

Cache attacks: Flush+Reload

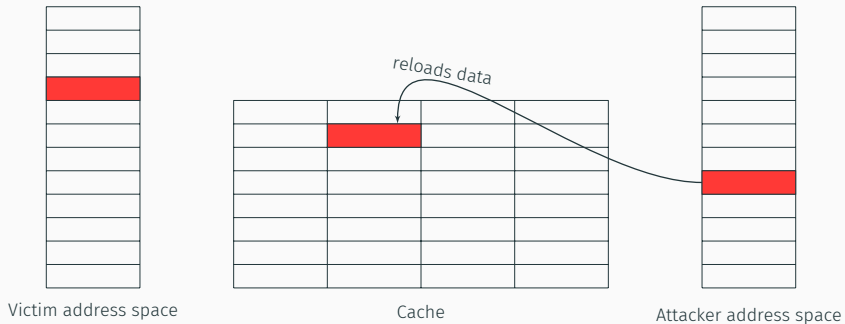


Step 1: Attacker maps shared library (shared memory, in cache)

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

Flush+Reload: Applications

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

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

B. Gülmözöglu, 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

Flush+Reload: Applications

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

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

B. Gülmezoglu, M. S. Inci, T. Eisenbarth, and B. Sunar. “A Faster and More Realistic Flush+Reload Attack on AES”. In: *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

Flush+Reload: Pros and cons

- **fine granularity**: 1 cache line (64 Bytes)

Flush+Reload: Pros and cons

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

Flush+Reload: Pros and cons

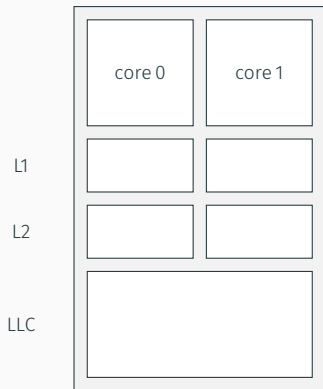
- **fine granularity**: 1 cache line (64 Bytes)
 - but requires shared memory
- **memory deduplication** between VMs

Possible side channels using
memory deduplication?

Possible side channels using
memory deduplication?

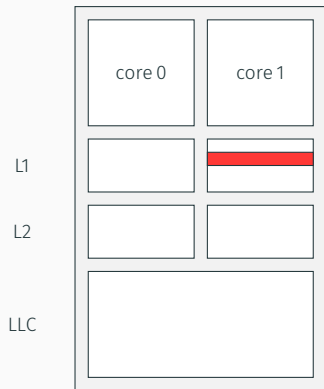
Disable memory deduplication!

Inclusive property



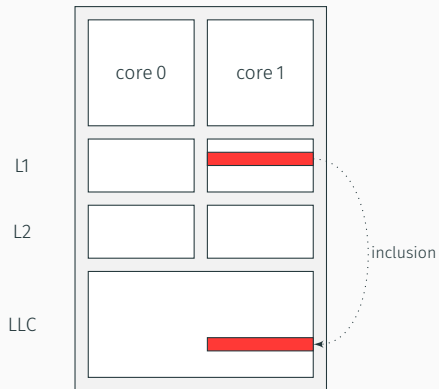
- **inclusive** LLC: superset of L1 and L2

Inclusive property



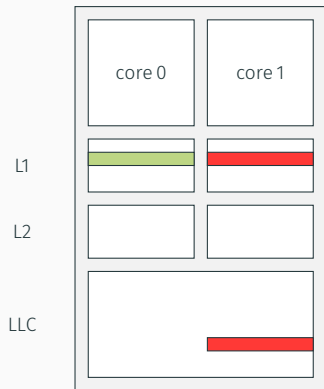
- **inclusive** LLC: superset of L1 and L2

Inclusive property



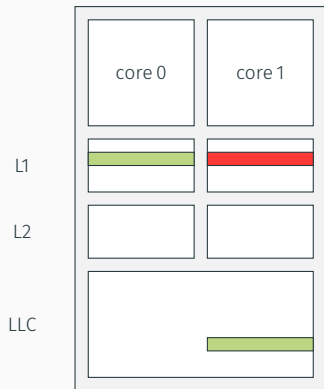
- **inclusive** LLC: superset of L1 and L2

Inclusive property



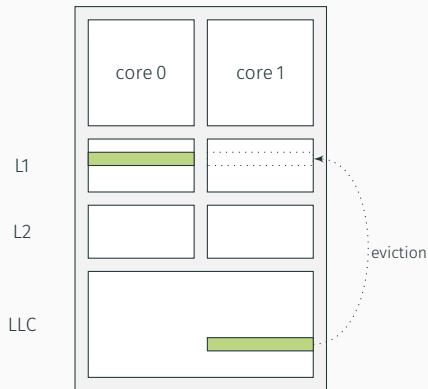
- **inclusive** LLC: superset of L1 and L2

Inclusive property



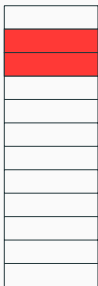
- **inclusive** LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2

Inclusive property

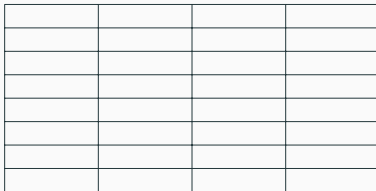


- **inclusive** LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
- a core can **evict lines** in the private L1 of another core

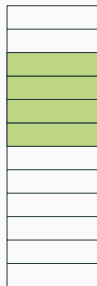
Cache attacks: Prime+Probe



Victim address space

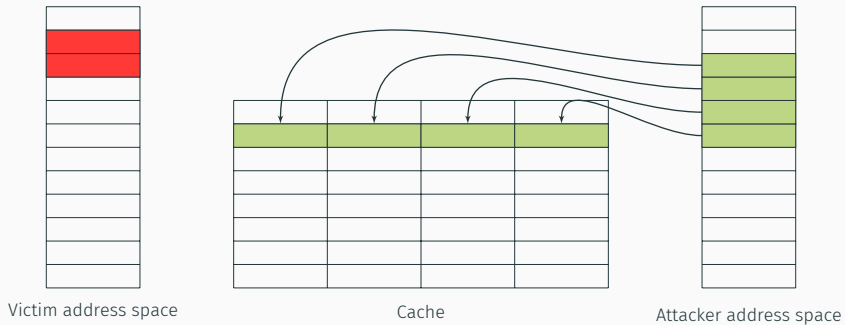


Cache



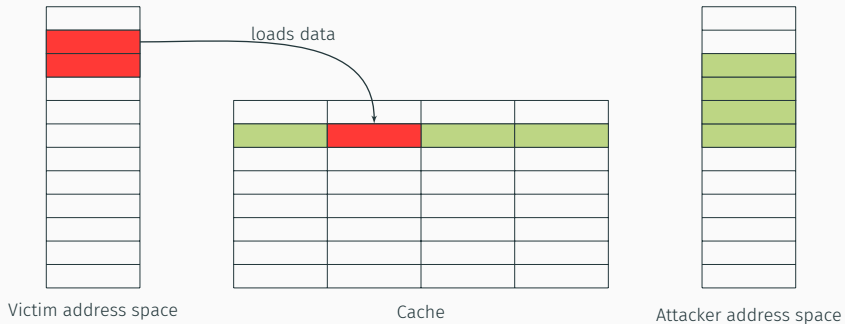
Attacker address space

Cache attacks: Prime+Probe



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

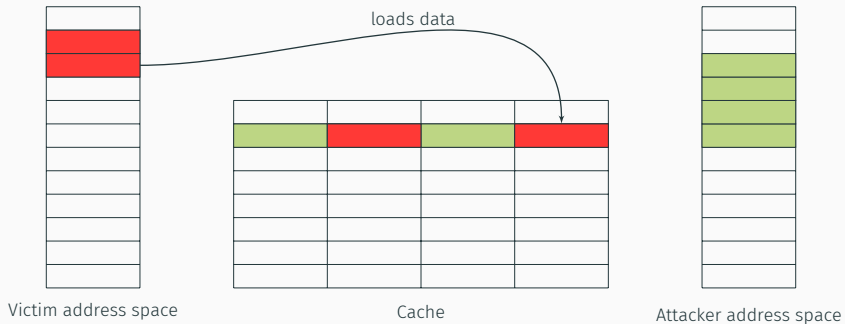
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

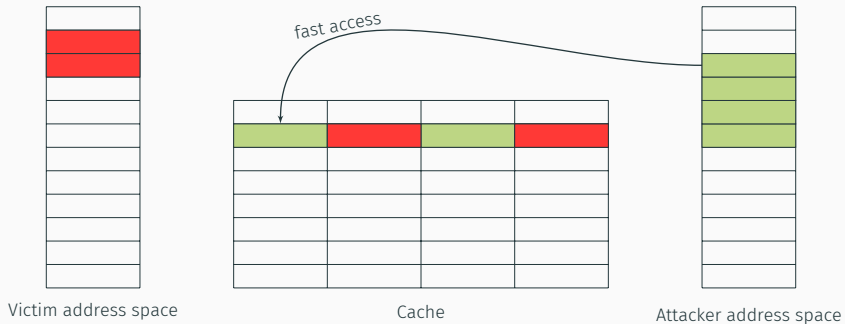
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

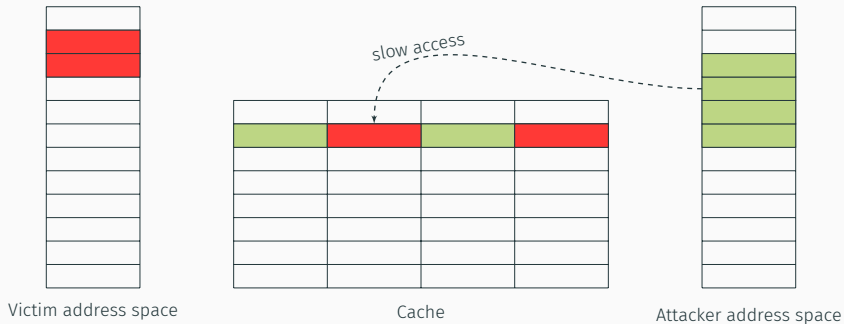


Step 1: Attacker **primes**, i.e., fills, the cache (no shared memory)

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

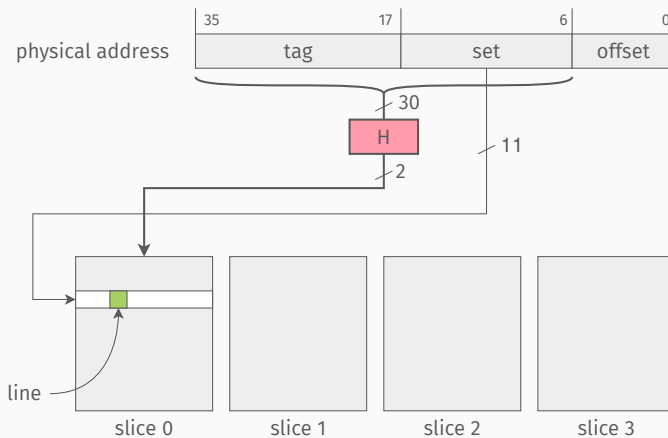
Step 3: Attacker **probes** data to determine if set has been accessed

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?

Last-level cache addressing



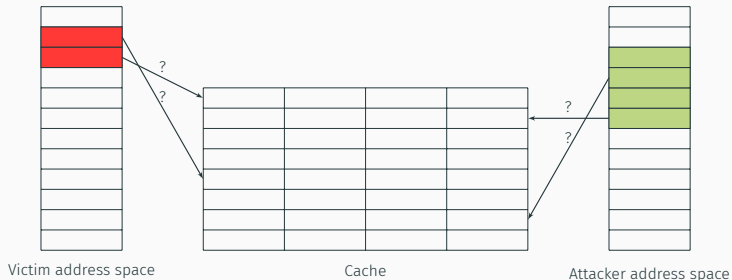
Last-level cache addressing

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



Prime+Probe on recent processors?

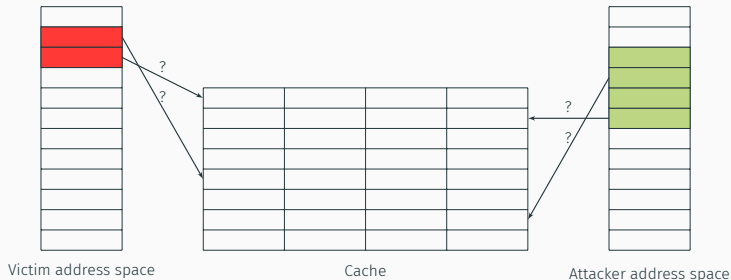
Undocumented function → impossible to **target a set**



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

Prime+Probe on recent processors?

Undocumented function → impossible to **target a set**



→ 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

Prime+Probe: Applications

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

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

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

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

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

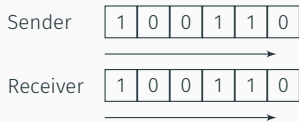
Covert channels in the cloud

- 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

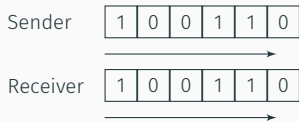
- covert channel: two processes communicating with each other
 - not allowed to do so, e.g., across VMs
- literature: stops working with noise on the machine
- solution? “Just use error-correcting codes”

Why can't we just use error correcting codes?

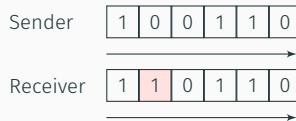


(a) Transmission without errors

Why can't we just use error correcting codes?

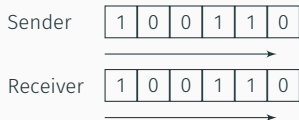


(a) Transmission without errors

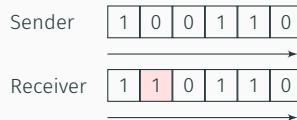


(b) Noise: **substitution** error

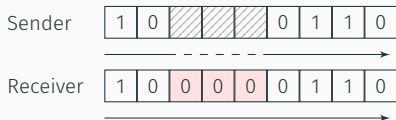
Why can't we just use error correcting codes?



(a) Transmission without errors

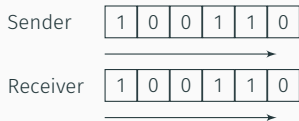


(b) Noise: **substitution** error

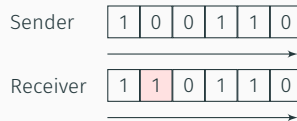


(c) Sender descheduled: **insertions**

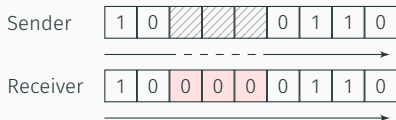
Why can't we just use error correcting codes?



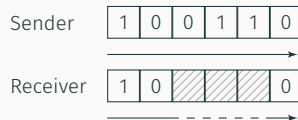
(a) Transmission without errors



(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, 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

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

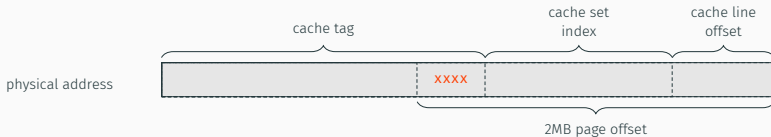
- need a set of addresses in the same cache set and same slice

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

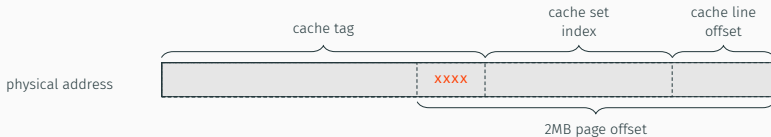
- need a set of addresses in the **same cache set** and **same slice**
- problem: slice number depends on all bits of the physical address



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

Eviction set generation

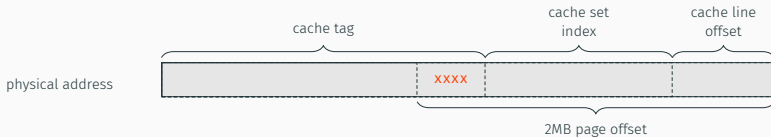
- need a set of addresses in the **same cache set** and **same slice**
- problem: slice number depends on all bits of the physical address



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

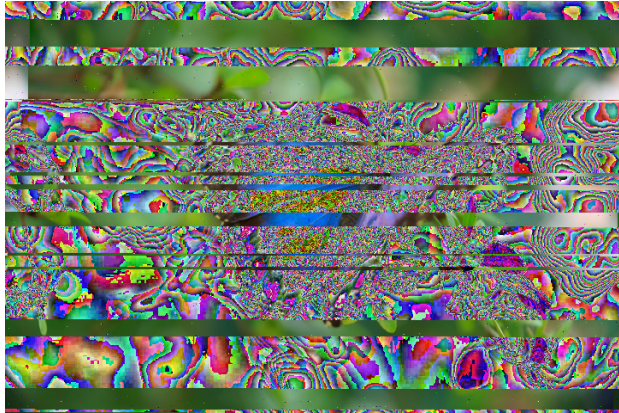
Eviction set generation

- 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



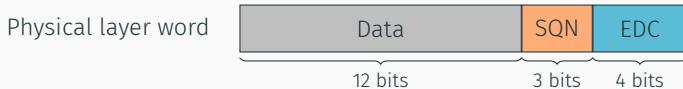
Handling synchronization errors

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

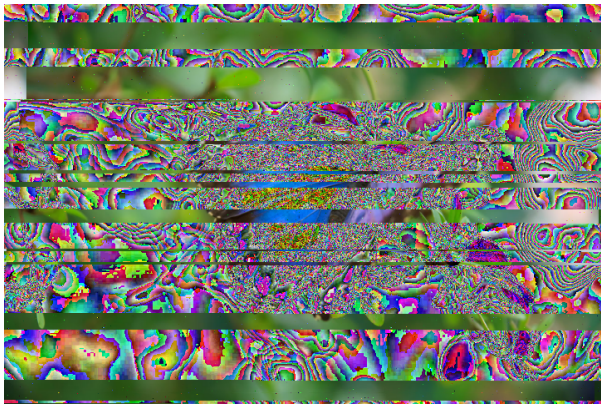


Handling synchronization errors

- 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)

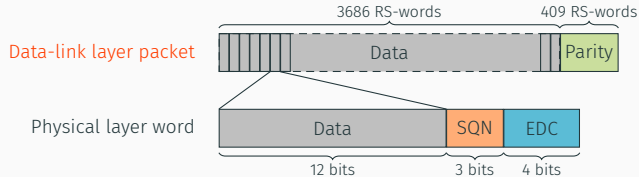


Data-link layer: Error correction

- 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%	<code>stress -m 1</code>

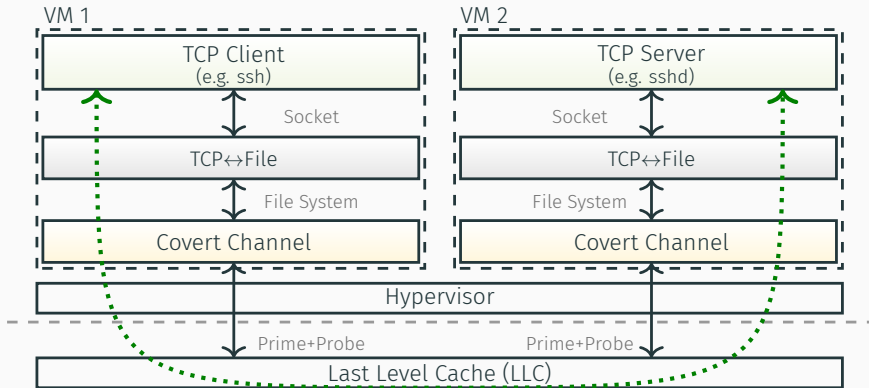
Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	<code>stress -m 1</code>
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**

Increasing the attack surface

It's not just caches: Also the DRAM, GPU, MMU, TLB!

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

Cristiano Giuffrida
Vrije Universiteit
Amsterdam

Herbert Bos
Vrije Universiteit
Amsterdam

Kaveh Razavi
Vrije Universiteit
Amsterdam

Ben Gras
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 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, cache attacks do not work and only a slow cross-CPU covert channel over the memory bus is known. In contrast, we demonstrate a high-speed covert channel as well as the first side-channel attack exploiting

such settings, a major requirement is that no sensitive information is leaked between tenants, therefore proper isolation mechanisms are crucial to the security of these environments. While software isolation is enforced by hypervisors, shared hardware presents risks of information leakage between tenants. Previous research shows that microarchitectural attacks can leak secret information of victim processes, e.g., by clever analysis of dependent timing differences. Such cryptographic keys or encryption boundaries via covert channels. Cloud providers can deploy

caches that have
ation and break
y community has
ises that effectively
course, other shared
assumption is that un-
annel offered by these
ie and too coarse-grained
ation.
his paper, we revisit this as-

ers and mobile phones. These attacks allow attackers to leak secret information in a reliable and fine-grained way [13, 14, 16, 38, 59] as well as compromise fundamental security defenses such as ASLR [17, 20, 24, 28]. The most prominent class of side-channel attacks leak information via the shared CPU data or instruction caches. Hence, the community has developed a variety of powerful new defenses to protect shared caches against these attacks, either by partitioning them, carefully sharing them between untrusted programs in the system, or sanitizing the traces left in the cache during the execution [6, 7, 11, 17, 57, 67].

Malicious Management Unit: Why Stopping Cache Harder Than You Think

Herbert Bos
Vrije Universiteit
Amsterdam

Cristiano Giuffrida
Vrije Universiteit
Amsterdam

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

It's not just native code on x86: Mobile and web too!

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

Abstract. Research showed that microarchitectural attacks like cache attacks can be performed through websites using JavaScript. These timing attacks allow an adversary to spy on users secrets such as their keystrokes, leveraging fine-grained timers. However, the W3C and browser vendors responded to this significant threat by eliminating fine-grained timers from JavaScript. This renders previous high-resolution microarchitectural attacks not applicable. We demonstrate the inefficiency of attacking a wide range of new sources of this mitigation by finding and enumerating methods that exceed the resolution of official timers by 3 to 4 orders of magnitude on all major browsers, an on the browser. Our timing measurements do not only reveal our attacks to their full extent but also allow implementing new and an unsupervised app in a virtual machine channel between aware. Our results emphasize that quick-fix mitigations can create a dangerous false sense of security.

Practical Keystroke Sandboxed JavaScript

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

Abstract

Over the last 10 years, cache attacks on Intel x86 CPUs have increased attention among the scientific community. Powerful techniques to exploit cache side channels have been developed. However, modern smartphones, especially ARM CPUs that have a different instruction set than x86, have not been attacked. We solve key challenges in cross-core cache attacks, such as Evict+Reload, and demonstrate state-of-the-art covert attacks on ARM devices without the need for orders of magnitude more monitor tap and swipe attacks. We even derive the lengths of the attacks. Eventually, we are able to implement a set of primitives that can even

the possibility to use Flush+Reload to automatically exploit cache-based side channels via cache template attacks on Intel platforms. Flush+Reload does not only allow for efficient attacks against cryptographic implementations [8, 26, 56], but also to infer keystroke information and even to build keyloggers on Intel platforms [19]. In contrast to attacks on cryptographic algorithms, which are typically triggered multiple times, these attacks require a significantly higher accuracy as an attacker has only one single chance to observe a user input event.

Although a few publications about cache attacks on AES T-table implementations on mobile devices exist [10, 50–52, 57], the more efficient cross-core attack techniques Prime+Probe, Flush+Reload, Evict+Reload, and Flush+Flush [18] have not been applied on smartphones. In fact, there was reasonable doubt [60] whether these cross-core attacks can be mounted on ARM-based devices at all. In this work, we demonstrate that these attack techniques are applicable on ARM-based devices by solving the following key challenges systematically:

1. Last-level caches are not inclusive on ARM and thus attacks cannot rely on this property. In this work, we

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

It's not just side channels: Fault attacks too!

CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management

Adrian Tang
Columbia University

Simha Sethumadhavan
Columbia University

Salvatore Stolfo
Columbia University

Abstract

The need for power- and energy-efficient computing has resulted in aggressive cooperative hardware-software energy management mechanisms on modern commodity devices. Most systems today, for example, allow software to control the frequency and voltage of the underlying hardware at a very fine granularity to extend battery life. Despite their benefits, these software-exposed energy management mechanisms pose grave security implications that have not been studied before.

In this work, we present the CLKSCREW attack, a new class of fault attacks that exploit the security-obliviousness of energy management mechanisms on these fault attacks become more accessible since they can now be conducted without the need for physical access to the devices or fault injection equipment. We demonstrate CLKSCREW on commodity ARM/Android devices. We show that a malicious kernel driver (1) can extract secret cryptographic keys from Trustzone, and (2) can escalate its privileges by loading self-signed code into Trustzone. As the first work to show the security ramifications of energy management mechanisms, we urge the community to re-examine these security-oblivious designs.

Drammer: De-
on-

Victor van der Veen
Vrije Universiteit Amsterdam
vvdveen@cs.vu.nl

Daniel Gruss
Graz University of Technology
gruss@tugraz.at

Yanick
UC San Diego
yanick@cs.ucsd.edu

Clémentine Maurice
Graz University of Technology
cmaurice@tugraz.at

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

maximize performance. Take as an example, Dynamic Voltage and Frequency Scaling (DVFS) [47], a ubiquitous energy management technique that saves energy by regulating the frequency and voltage of the processor cores according to runtime computing demands. To support DVFS, at the hardware level, vendors have to design the underlying frequency and voltage of the processor to be portable across a wide range of devices while ensuring cost efficiency. At the software level, kernel developers need to track and match program demands to optimize frequency and voltage settings to minimize consumption for those demands. These mechanisms actively and at very fine,

Despite the ubiquity of these mechanisms on commodity systems, their inclusion in the design of these software interoperability mechanisms has not given much time-to-market concerns on optimizing the functional aspects of these mechanisms. These combination of factors

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

Daniel Gruss, Clémentine Maurice¹, and Stefan Mangard
Graz University of Technology, Austria

→4. A fundamental assumption in software security is that a system can only be modified by processes that may write to memory. However, a recent study has shown that parasitic attacks can change the content of a memory cell without accessing other memory locations in a high frequency. Rowhammer bug occurs in most of today's memory modules and has severe consequences for the security of all affected systems, including cloud services.

Rowhammer attacks related to Rowhammer so far rely on the availability of cache flush instruction in order to cause accesses to DRAM at a sufficiently high frequency. We overcome this limitation by using complex cache replacement policies. We show that caches can be triggered to trigger the Rowhammer bug with high frequency. This allows to trigger the Rowhammer bug in regular memory accesses. This allows to trigger the Rowhammer bug in highly restricted and even scripting environments. We demonstrate a fully automated attack that requires nothing but a website with JavaScript to trigger faults on remote hardware. Thereby, we can gain unrestricted access to systems of website visitors. We show that the attack works on off-the-shelf systems. Existing countermeasures do not protect against this new Rowhammer attack.

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Yoongu Kim¹ Ross Daly* Jeremie Kim¹ Chris Fallin* Ji Hye Lee¹
Donghyuk Lee¹ Chris Wilkerson² Konrad Lai Onur Mutlu¹

¹Carnegie Mellon University ²Intel Labs

Abstract. Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should have no unintended side effects on data stored in other addresses. However, as DRAM process technology scales down to smaller dimensions, it becomes more difficult to prevent DRAM cells from electrically interacting with each other. In this paper, we expose the vulnerability of commodity

disturbance errors, DRAM manufacturers have been employing a two-pronged approach: (i) improving inter-cell isolation through circuit-level techniques [22, 32, 49, 61, 73] and (ii) screening for disturbance errors during post-production testing [3, 4, 64]. We demonstrate that their efforts to contain disturbance errors have not always been successful, and that erroneous DRAM chips have been slipping into the field.¹

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**?

Future: More transient execution attacks?

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. Evtushkin, 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

Evolution of microarchitectural attacks

Clémentine Maurice, CNRS, IRISA

December 11, 2018–WOS 8