Évolution des attaques sur la micro-architecture

Clémentine Maurice, CNRS, CRIStAL

15 Octobre 2021—Journée Sécurité SIF, GDR Sécurité Informatique, RSD et SOC2

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

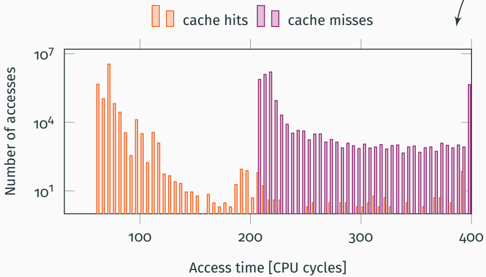
Attacks on micro-architecture

- 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

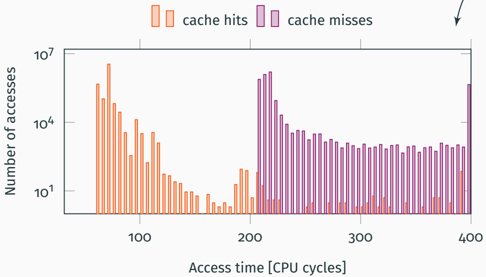
identification



Attacks on micro-architecture

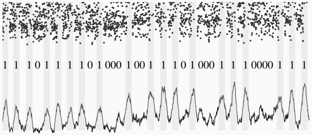
- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing hardware errors
  - side channels: observing side effects of hardware on computations

identification



→

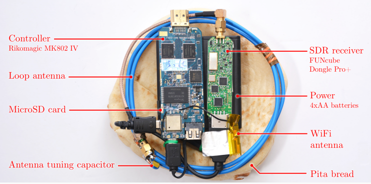
attack



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

Attacker model

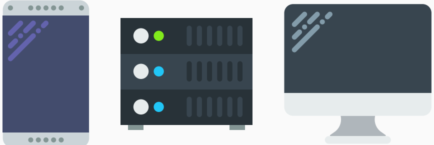
Hardware-based attacks a.k.a physical attacks



Physical access to hardware
→ embedded devices

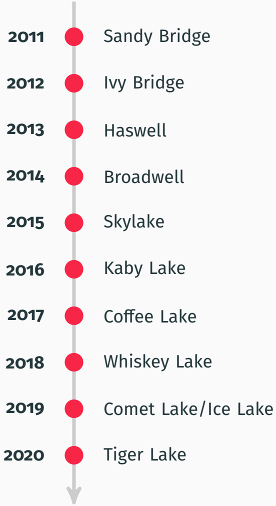
VS

Software-based attacks a.k.a micro-architectural attacks



Co-located or remote attacker
→ complex systems

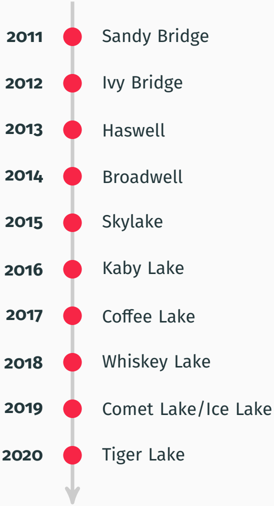
From small optimizations...



2011	Sandy Bridge
2012	Ivy Bridge
2013	Haswell
2014	Broadwell
2015	Skylake
2016	Kaby Lake
2017	Coffee Lake
2018	Whiskey Lake
2019	Comet Lake/Ice Lake
2020	Tiger Lake

- new microarchitectures yearly

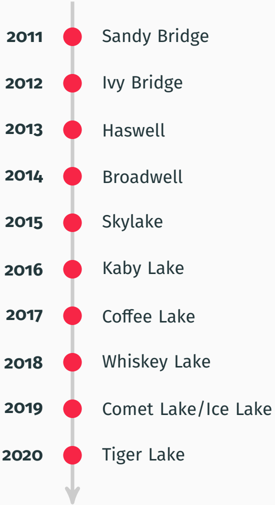
From small optimizations...



2011	Sandy Bridge
2012	Ivy Bridge
2013	Haswell
2014	Broadwell
2015	Skylake
2016	Kaby Lake
2017	Coffee Lake
2018	Whiskey Lake
2019	Comet Lake/Ice Lake
2020	Tiger Lake

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

From small optimizations...



2011	Sandy Bridge
2012	Ivy Bridge
2013	Haswell
2014	Broadwell
2015	Skylake
2016	Kaby Lake
2017	Coffee Lake
2018	Whiskey Lake
2019	Comet Lake/Ice Lake
2020	Tiger Lake

- 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 (vulnerable) 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 (vulnerable) 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 (vulnerable) victim process via hardware usage
- pure-software attacks by unprivileged processes
- sequences of benign-looking actions → hard to detect

Outline

Historical recap of past attacks

Outline

Historical recap of past attacks

Recent advances

Outline

Historical recap of past attacks

Recent advances

Future and challenges

Historical Recap

Micro-architectural attacks: Two faces of the same coin

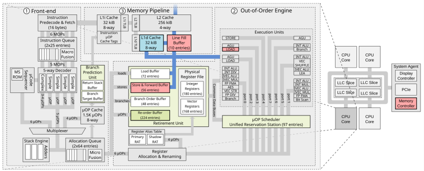
Implementation

Algorithm 1: Square-and-multiply exponentiation

```
Input: base $b$, exponent $e$, modulus $n$
Output: $b^e \mod n$
$X \leftarrow 1$
for $i \leftarrow bitlen(e)$ downto 0 do
    $X \leftarrow \mathrm{multiply}(X, X)$
    if $e_i = 1$ then
        $X \leftarrow \mathrm{multiply}(X, b)$
    end
end
return $X$
```

&

Hardware



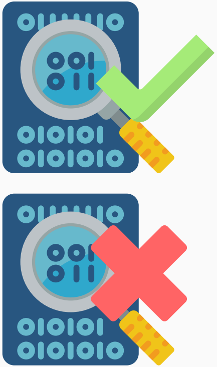
Research questions

1. Which **software implementation** is vulnerable?
2. Which **hardware component** is vulnerable?

1. Which software implementation is vulnerable?

State of the art (more or less)

1. Spend too much time reading OpenSSL code
2. Find vulnerability
3. Exploit it manually using known side channel
   → e.g. CPU cache
4. Publish
5. goto step 1

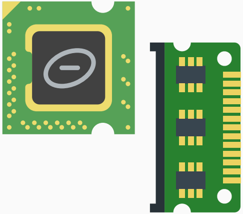


For example: CVE-2016-0702, CVE-2016-2178, CVE-2016-7440, CVE-2016-7439, CVE-2016-7438, CVE-2018-0495, CVE-2018-0737, CVE-2018-10846, CVE-2019-9495, CVE-2019-13627, CVE-2019-13628, CVE-2019-13629, CVE-2020-16150

2. Which hardware component leaks information?

State of the art (more or less)

1. Spend too much time reading Intel manuals
2. Find weird behavior in corner cases
3. Exploit it
4. Publish
5. goto step 1



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.

Hyper-threading: Same-core attacks

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

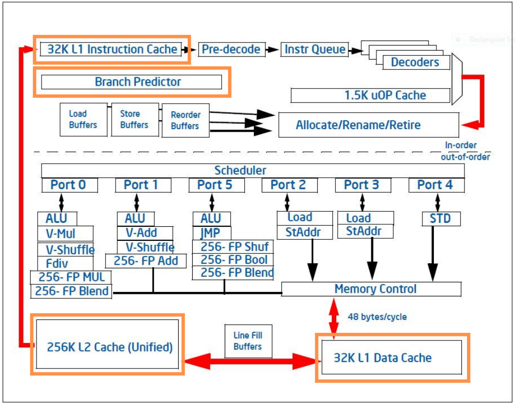


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

Easy solution #1

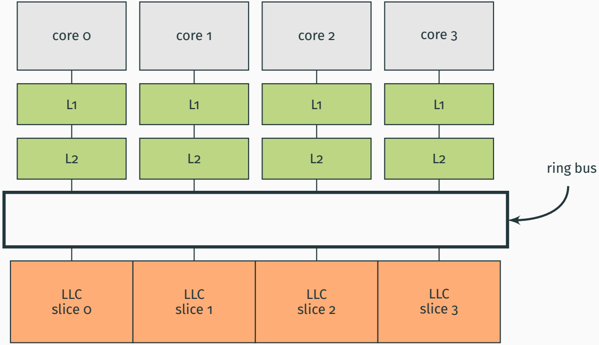
Possible side channels using components shared by a core?

Easy solution #1

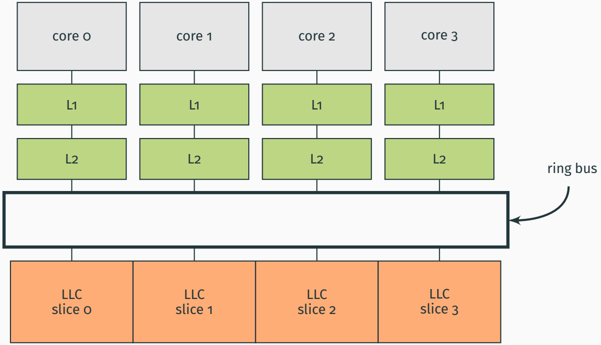
Possible side channels using components shared by a core?

Stop sharing a core!

Caches on Intel CPUs

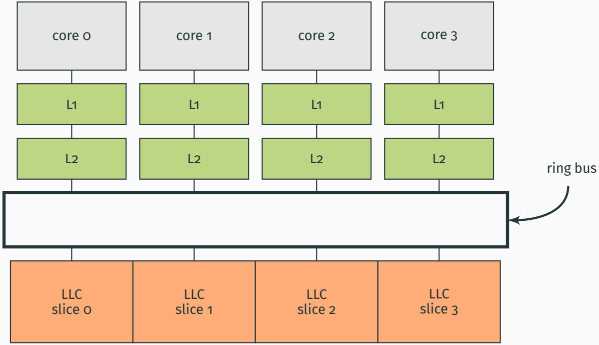


Caches on Intel CPUs



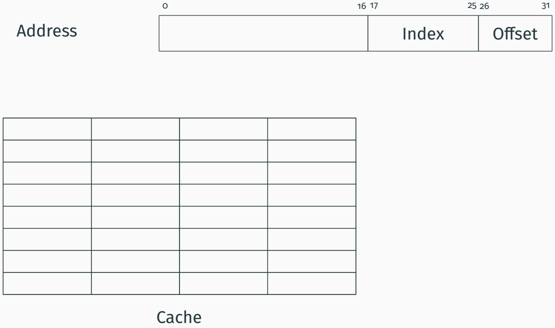
- L1 and L2 are private

Caches on Intel CPUs

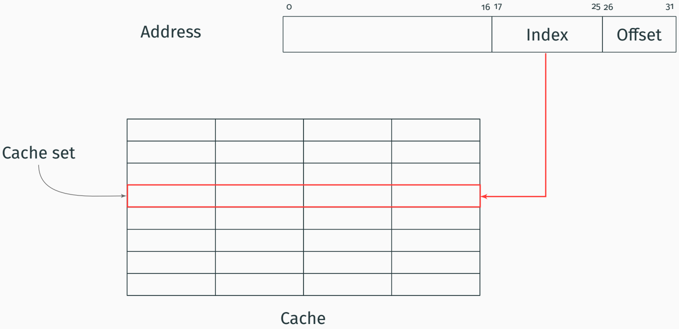


- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive

Set-associative caches

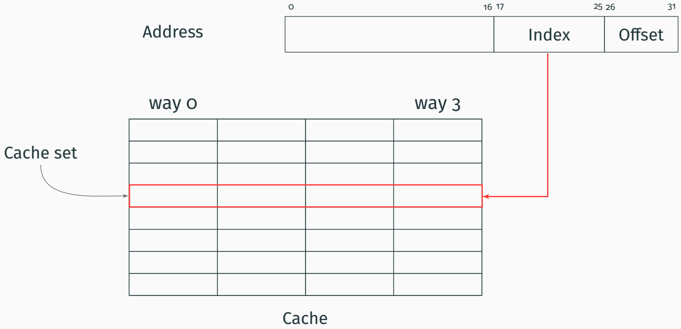


Set-associative caches



Data loaded in a specific set depending on its address

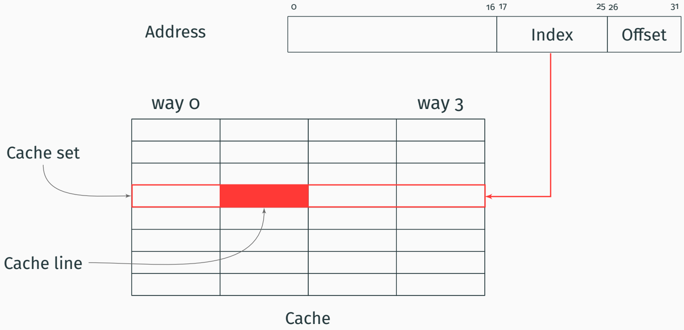
Set-associative caches



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 → 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

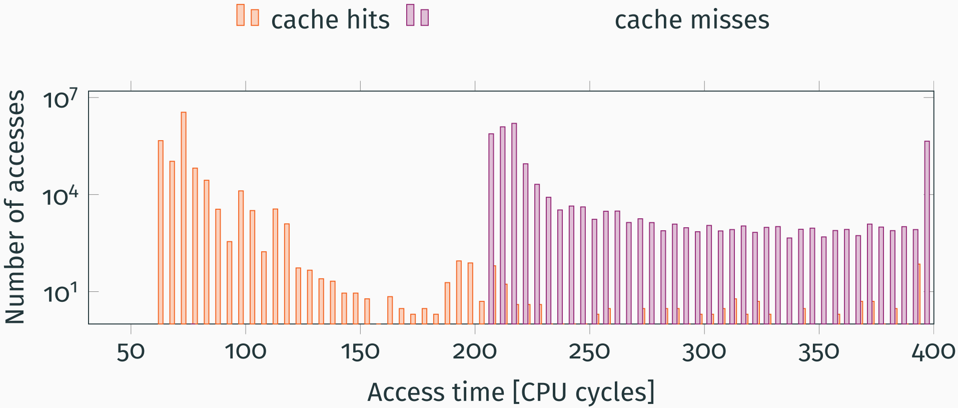
Cache attacks

- 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

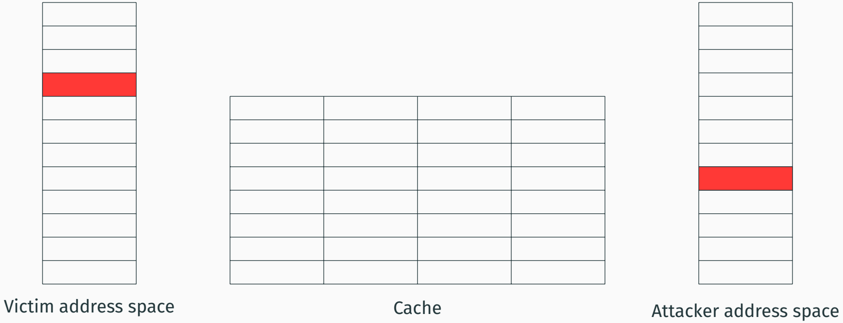
Cache attacks

- 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

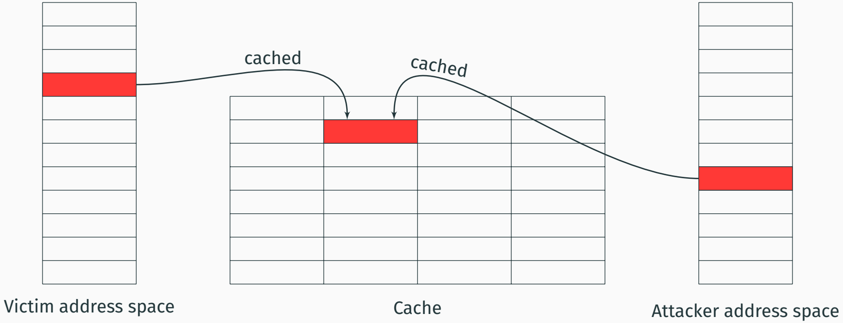


Cache attacks: Flush+Reload



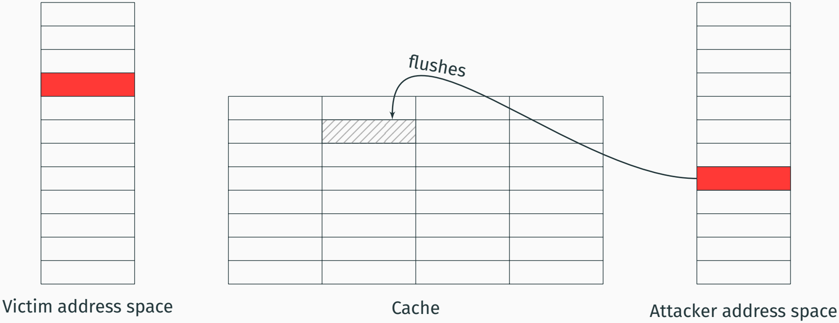
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attacks: Flush+Reload



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

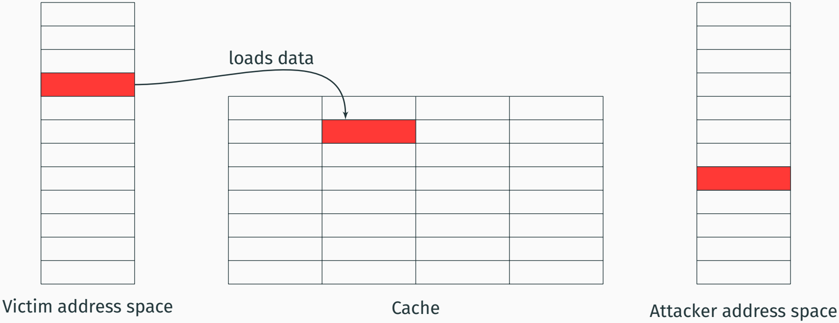
Cache attacks: Flush+Reload



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

Step 2: Attacker flushes the shared cache line

Cache attacks: Flush+Reload

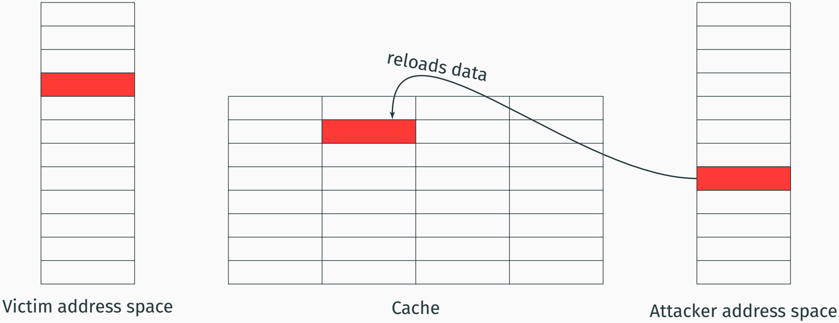


Step 1: Attacker maps shared library (shared memory, in cache)

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)
- covert channels in native environments cross-VM: 298 KBps

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

B. Gülmezoglu et al. "A Faster and More Realistic Flush+Reload Attack on AES". In: *Constructive Side-Channel Analysis and Secure Design (COSADE)*. 2015

Flush+Reload: Pros and cons

- high spatial resolution: 1 cache line (64 Bytes)

Flush+Reload: Pros and cons

- high spatial resolution: 1 cache line (64 Bytes)
- but requires shared memory + `clflush` instruction

Flush+Reload: Pros and cons

- high spatial resolution: 1 cache line (64 Bytes)
- but requires shared memory + `clflush` instruction

→ memory deduplication between VMs

Easy solution #2

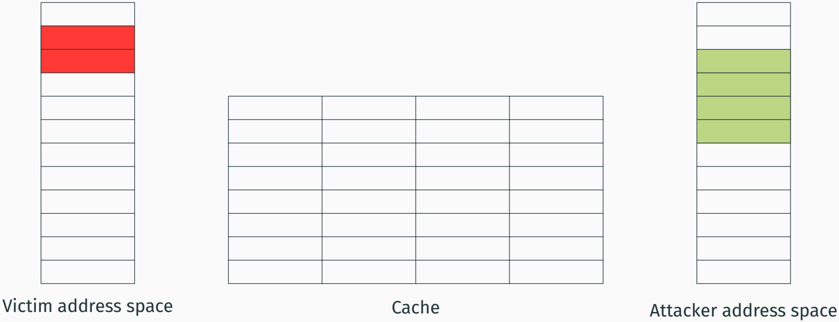
Possible side channels using memory deduplication?

Easy solution #2

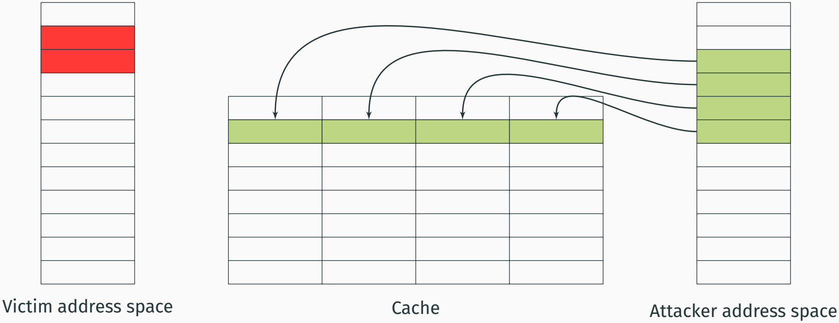
Possible side channels using memory deduplication?

Disable memory deduplication!

Cache attacks: Prime+Probe

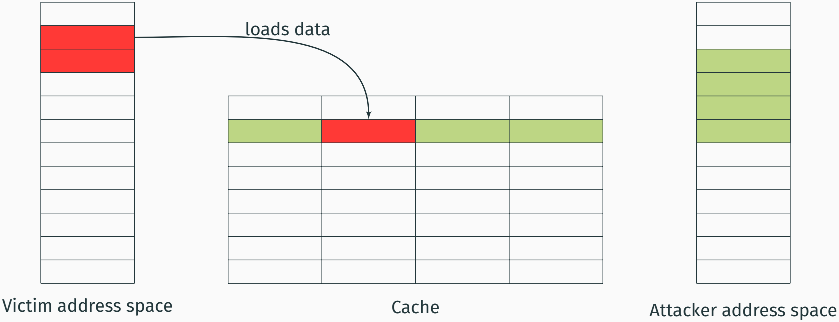


Cache attacks: Prime+Probe



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

Cache attacks: Prime+Probe



loads data

Victim address space

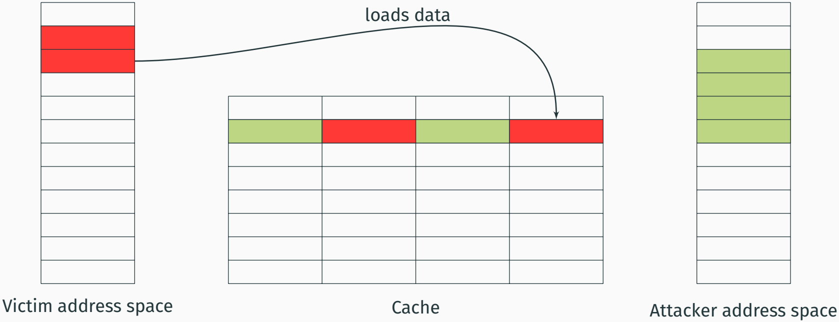
Cache

Attacker address space

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

Step 2: Victim evicts cache lines while running

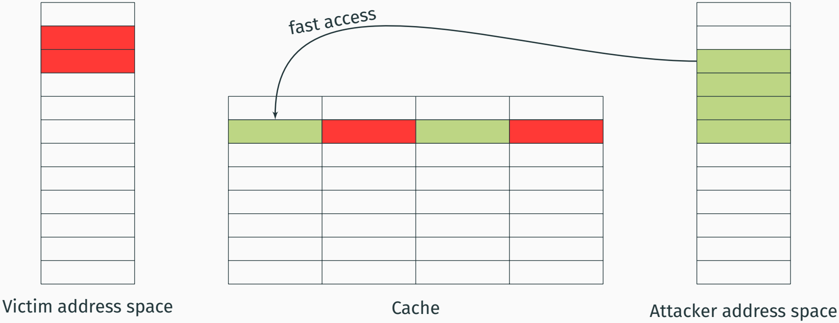
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

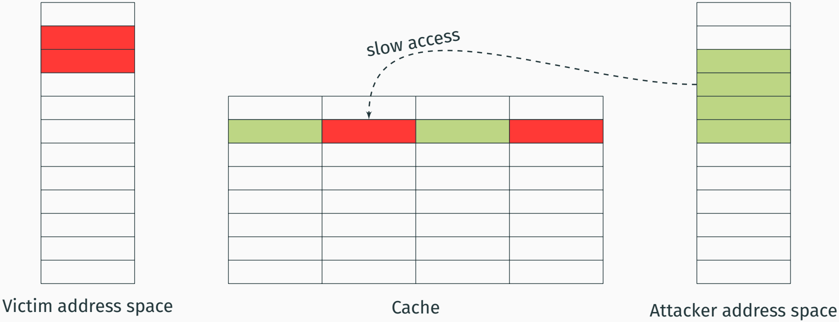


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

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?

We need:

1. an eviction set: addresses in the same set, in the same slice (issue #1 and #2)
2. an eviction strategy (issue #3)

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

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

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

Easy solution #3

Possible side channels using components shared by a CPU?

Easy solution #3

Possible side channels using components shared by a CPU?

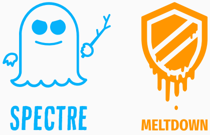
Stop sharing a CPU!?

Recent Advances

Recent advances

Transient execution attacks

Transient execution attacks

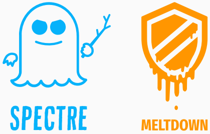


- novel class of attacks $\neq$ side-channel attacks
- discovered in 2018 with Spectre and Meltdown

C. Canella et al. “A Systematic Evaluation of Transient Execution Attacks and Defenses”. In: *USENIX Security Symposium*. 2019

`https://transient.fail/`

Transient execution attacks



- novel class of attacks $\neq$ side-channel attacks
- discovered in 2018 with Spectre and Meltdown
- SO MANY VARIANTS

C. Canella et al. “A Systematic Evaluation of Transient Execution Attacks and Defenses”. In: *USENIX Security Symposium*. 2019

`https://transient.fail/`

Transient execution attacks

- CPU avoids waiting for input data or availability of execution units

→ out-of-order execution and speculation

- sequential semantics is preserved

Transient execution attacks

- CPU avoids waiting for input data or availability of execution units

→ out-of-order execution and speculation

- sequential semantics is preserved

- some instructions are never committed, *i.e.*, finally executed
  - instructions that cause an exception + following instructions
  - instructions in branches that are mispredicted
- these instructions are called transient instructions

Transient execution attacks

- CPU avoids waiting for input data or availability of execution units

→ out-of-order execution and speculation

- sequential semantics is preserved

- some instructions are never committed, *i.e.*, finally executed
  - instructions that cause an exception + following instructions
  - instructions in branches that are mispredicted
- these instructions are called transient instructions

- architectural state → everything is fine

Transient execution attacks

- issue: instructions not committed leave traces in microarchitecture
- microarchitectural state is not supposed to be visible…
- … but we know how to recover the state of caches

Transient execution attacks

- issue: instructions not committed leave traces in microarchitecture
- microarchitectural state is not supposed to be visible...
- ... but we know how to recover the state of caches
- microarchitectural state → everything is not fine

Transient execution attacks

- issue: instructions not committed leave traces in microarchitecture
- microarchitectural state is not supposed to be visible…
- … but we know how to recover the state of caches
- microarchitectural state → everything is not fine

- leaking kernel memory, recovering passwords...

Transient execution attacks

- issue: instructions not committed leave traces in microarchitecture
- microarchitectural state is not supposed to be visible...
- ... but we know how to recover the state of caches
- microarchitectural state → everything is not fine

- leaking kernel memory, recovering passwords...
- difficult to fix: lazy error handling was a bug, but speculative execution is a feature!

Recent advances

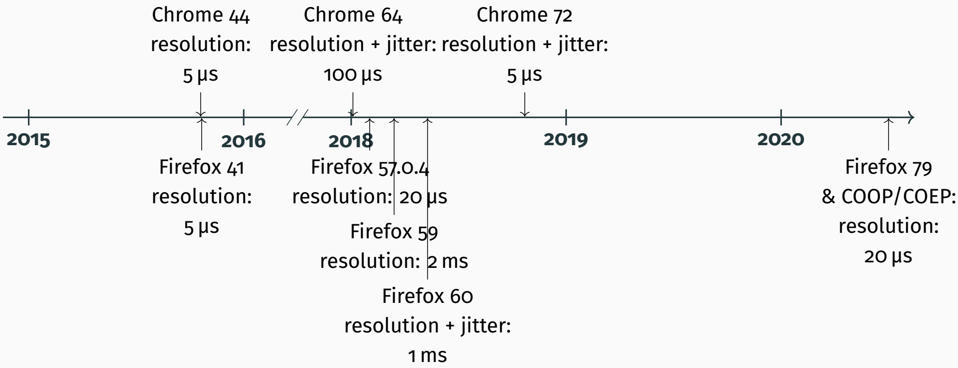
Porting micro-architectural attacks to the Web

Porting micro-architectural attacks to the Web

- side-channel attacks on the cache, DRAM, MMU, (...), and transient execution attacks like Spectre, ret2spec, RIDL, (...), are coming to web browsers
- very low-level attacks in a high-level language with many abstraction layers in between
- complex but not impossible to perform
- fundamentally hard or impossible to fix in the browser

T. Rokicki, C. Maurice, and P. Laperdrix. “Sok: In search of lost time: A review of javascript timers in browsers”. In: *EuroS&P'21*. 2021

JS and timers: A complicated history



Chrome 44 resolution: 5 µs

Chrome 64 resolution + jitter: 100 µs

Chrome 72 resolution + jitter: 5 µs

Firefox 41 resolution: 5 µs

Firefox 57.0.4 resolution: 20 µs

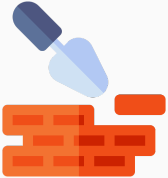
Firefox 59 resolution: 2 ms

Firefox 60 resolution + jitter: 1 ms

Firefox 79 & COOP/COEP: resolution: 20 µs

T. Rokicki, C. Maurice, and P. Laperdrix. “Sok: In search of lost time: A review of javascript timers in browsers”. In: *EuroS&P’21*. 2021

JS and timers: A complicated history



- initial countermeasures: lowering timer resolution
- browsers are adopting better isolation between websites (e.g., Site Isolation) to counter transient execution attacks
- back to higher timer resolution for usability → side-channel attacks are possible again!

T. Rokicki, C. Maurice, and P. Laperdrix. “Sok: In search of lost time: A review of javascript timers in browsers”. In: *EuroS&P’21*. 2021

Recent advances

Automating vulnerability and side channel discovery

Automating vulnerability and side channel discovery

CacheD: Identifying Cache-Based Timing Channels in Production Software

Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang, and Dinghao Wu
The Pennsylvania State University
{szw175, pxw172, xvl5190}@ist.psu.edu, zhang@cse.psu.edu, dwu@ist.psu.edu

Abstract

Side-channel attacks recover secret information by analyzing the physical implementation of cryptosystems based on non-functional computational characteristics, e.g. time, power, and memory usage. Among all well-known side channels, cache-based timing channels are notoriously severe, leading to practical attacks against certain implementations of theoretically secure crypto algorithms, such as RSA, ElGamal and AES. Such attacks target the hierarchical design of the modern computer memory system, where different memory access patterns of a program can bring observable timing difference.

In this work, we propose a novel technique to help software developers identify potential vulnerabilities that can lead to cache-based timing attacks. Our technique leverages symbolic execution and constraint solving to detect potential cache differences at each program point. We adopt a cache model that is general enough to capture various threat models that are employed in practical timing attacks. Our modeling and analysis are based on the formulation of cache access at different program locations along execution traces. We have implemented the proposed technique as a practical tool named CacheD (Cache Difference), and evaluated CacheD towards mul-

teristics. Typical attributes exploited in such attacks include time [30], power [37], memory consumption [28], network traffic [16], and electromagnetic [46].

Among all side-channel attacks, cache-based timing attacks steal confidential information based on the program's runtime cache behaviors. Cache-based timing attacks are perhaps the most practical and im since those attacks does not requir to the confidential compu carries enough inform AES [8, 11, 42, 53, 2 than cryptosystems, rese based timing channels ma mation [47, 57, 62, 58].

The mitigation mechanis ing channels can be categori ware based solutions. Hardw on new cache designs such as 31, 61], randomized/remapping line-locking cache [54]. But su sumes that crucial memory acces security experts) in the first place. solutions only consider cache-based to secret-dependent control flow [4,

ABSynthe: Auto Synthesis on Com

Giuffrida*, Michael Kurth

Osiris: Automated Discovery of Microarchitectural Side Channels

ARTIFACT EVALUATED usenix ASSOCIATION PASSED

niel Weber, Ahmad Ibrahim, Hamed Nemati, Michael Schwarz, Christian Rossow
CISPA Helmholtz Center for Information Security

ave been dis- e been used in plementations, or acks such as Spec- es, discovering side ss. zzing-based framework itectural side channels. ification of a CPU's ISA, ence triples and automati- timing-based side channel. eir usability as a side channel as the microarchitectural

Side channels often arise from abstraction and optimization [79]. For example, due to the internal complexity of modern CPUs, the actual implementation, *i.e.*, the microarchitecture, is abstracted into the documented architecture. This abstraction also enables CPU vendors to introduce transparent optimizations in the microarchitecture without requiring changes in the architecture. However, these optimizations regularly introduce new side channels that attackers can exploit [3, 10, 56, 69, 74, 80, 86, 89].

Although new side channels are commonly found, discovering a side channel typically requires manual effort and a deep understanding of the underlying microarchitecture. Moreover, with multiple thousand variants of instructions available on the x86 architecture alone [1], the number of possible side effects that can occur when combining instructions is too large Hence, manually identified side channels

DATA – Differential Address Trace Analysis: Finding Address-based Side-Channels in Binaries

Samuel Weiser[1], Andreas Zankl[2], Raphael Spreitzer[1], Katja Miller[2], Stefan Mangard[1], and Georg Sigl[2,3]

[1]Graz University of Technology, [2]Fraunhofer AISEC, [3]Technical University of Munich

Abstract

target for various side-channel attacks [11, 45, 77], as a successful attack undermines cryptographic security

Recent advances

Reproducible research?

Reproducible research?

ARCHI–SEC

- research on micro-architectural attacks is very difficult to reproduce
- advances: code sharing and Artifact Evaluation at major conferences like USENIX Security or ASPLOS
- ANR ARCHI-SEC: leveraging gem5 to improve reproducibility and build countermeasures (joint project with Télécom Paris, LIRMM, Laboratoire Hubert Curien, SECURE-IC, CRIStAL)

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?
- why do we still manually find vulnerabilities when we have automated tools?
- how to prevent attacks based on performance optimizations without removing performance?

Conclusion

- first paper by Kocher in 1996: 25 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!
- quick fixes don't work
- still a lot more work needed to find satisfying countermeasures

Thank you!

Contact

✉ `clementine.maurice@inria.fr`

🐦 @BloodyTangerine

Évolution des attaques sur la micro-architecture

Clémentine Maurice, CNRS, CRIStAL

15 Octobre 2021—Journée Sécurité SIF, GDR Sécurité Informatique, RSD et SOC2