Evolution of micro-architectural attacks

Clémentine Maurice, CNRS, CRIStAL

17 December 2021—32nd HP/HPE (Virtual) Colloquium On Information Security

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

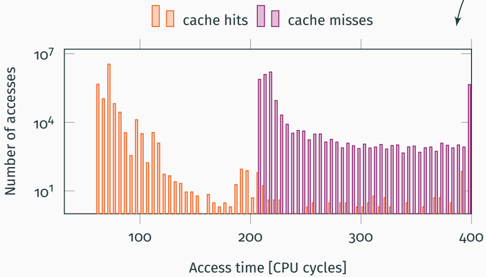
Attacks on micro-architecture

- 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

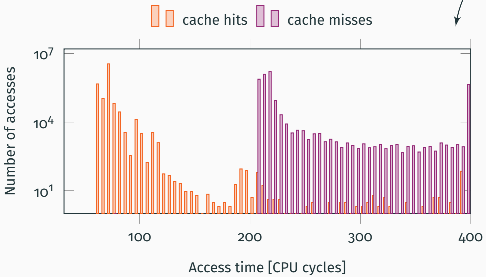
identification



Attacks on micro-architecture

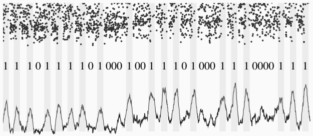
- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing hardware errors
  - side channels: observing side effects of hardware on computations

identification



→

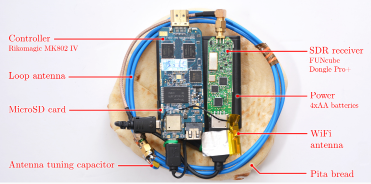
attack



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

Attacker model

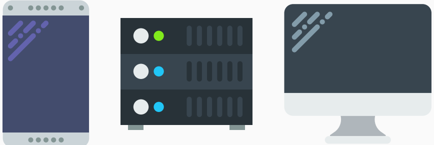
Hardware-based attacks a.k.a physical attacks



Physical access to hardware
→ embedded devices

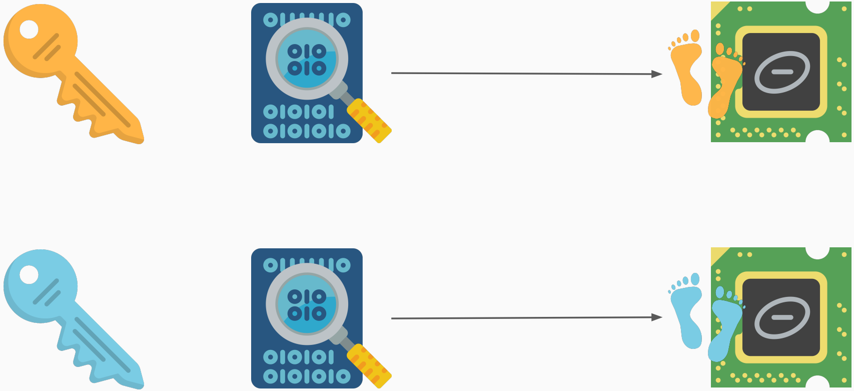
VS

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

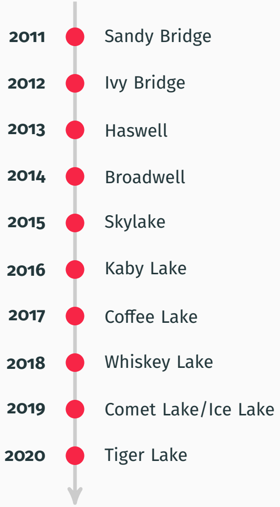


Co-located or remote attacker
→ complex systems

Side-channel attacks



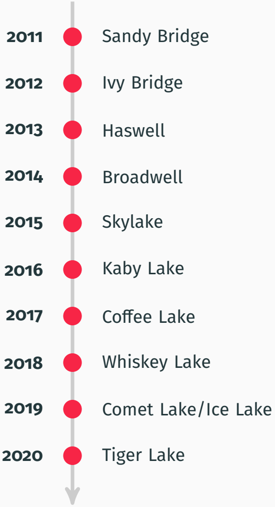
From small optimizations...



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

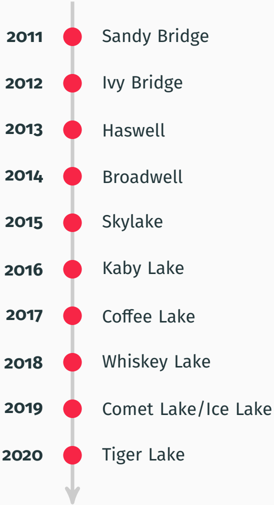
From small optimizations...



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

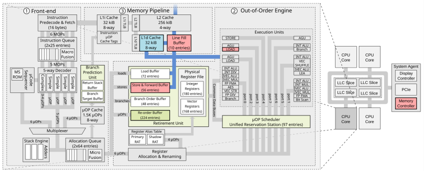
Implementation

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
    $X \leftarrow \text{multiply}(X, X)$
    if $e_i = 1$ then
        $X \leftarrow \text{multiply}(X, b)$
    end
end
return $X$
```

&

Hardware



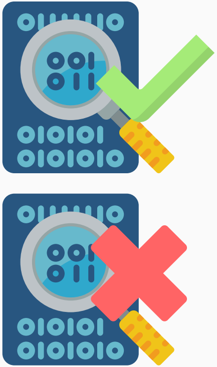
Research questions

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

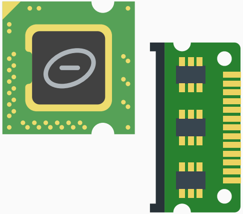


For example: CVE-2016-0702, CVE-2016-2178, CVE-2016-7440, CVE-2016-7439, CVE-2016-7438, CVE-2018-0495, CVE-2018-0737, CVE-2018-10846, CVE-2019-9495, CVE-2019-13627, CVE-2019-13628, CVE-2019-13629, CVE-2020-16150

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

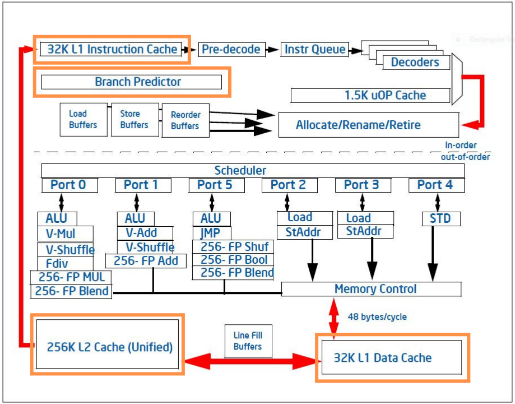


Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

Easy solution #1

Possible side channels using

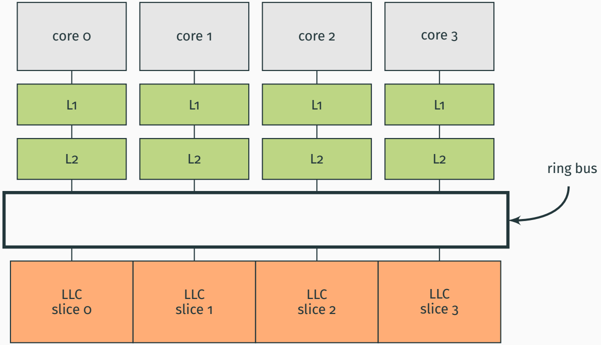
components shared by a core?

Easy solution #1

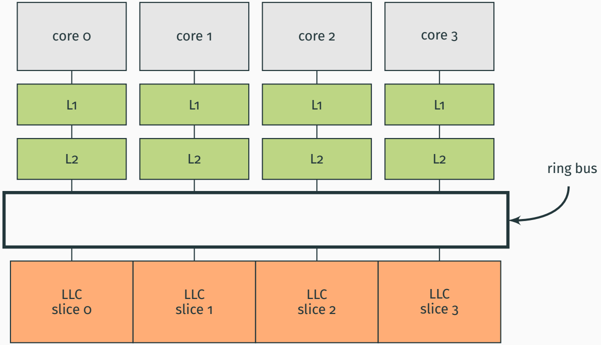
Possible side channels using components shared by a core?

Stop sharing a core!

Caches on Intel CPUs

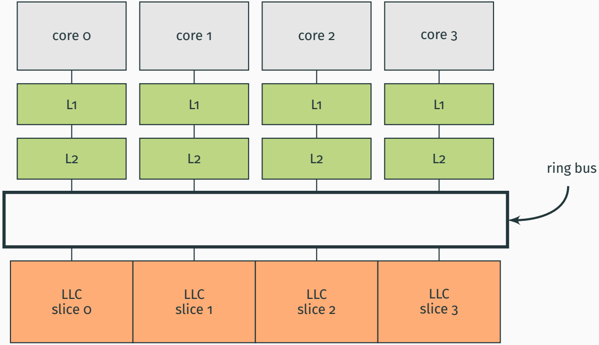


Caches on Intel CPUs



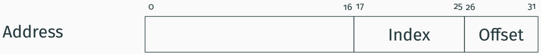
- L1 and L2 are private

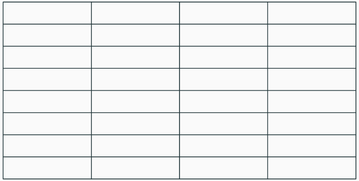
Caches on Intel CPUs



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

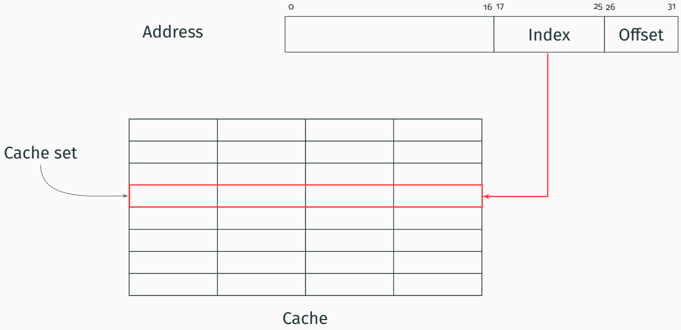
Set-associative caches





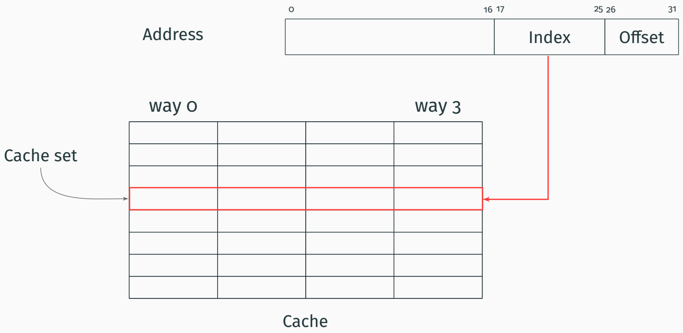
Cache

Set-associative caches



Data loaded in a specific set depending on its address

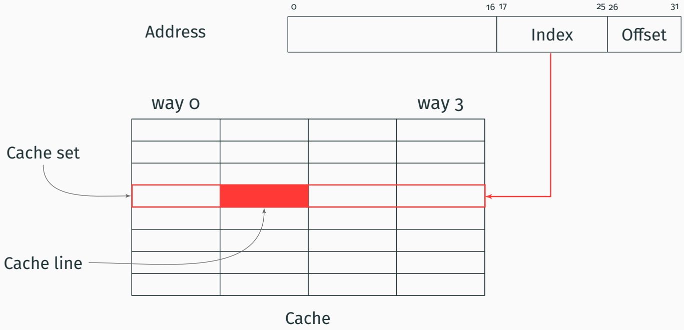
Set-associative caches



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 → 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

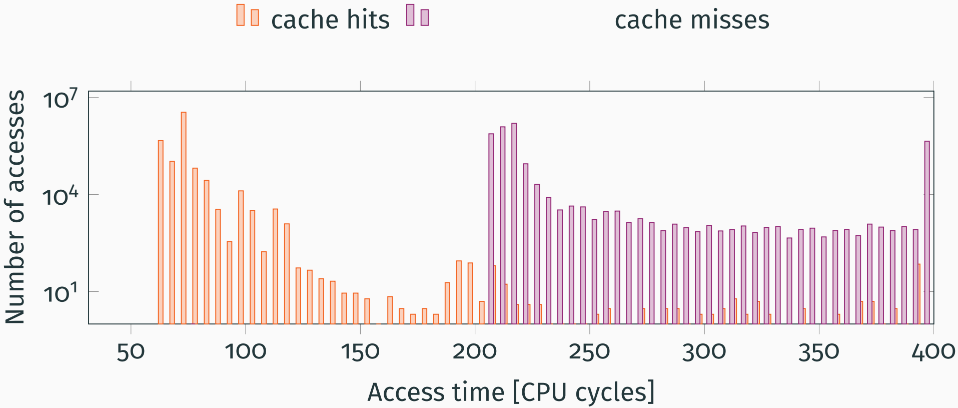
Cache attacks

- 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

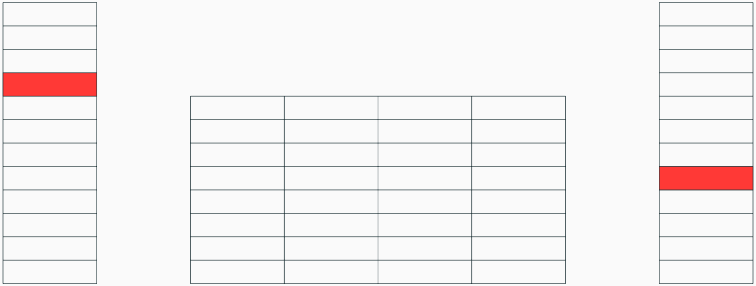
Cache attacks

- 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



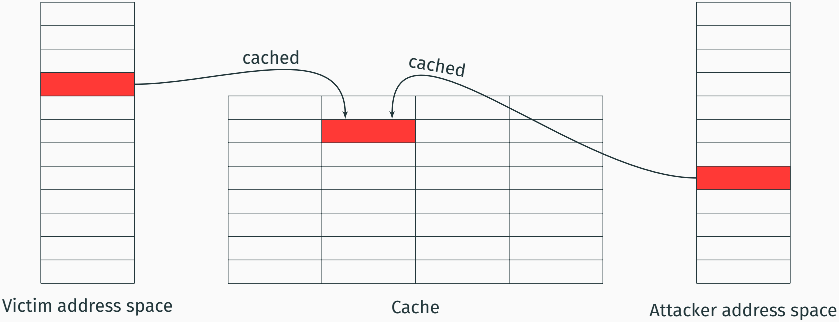
Victim address space

Cache

Attacker address space

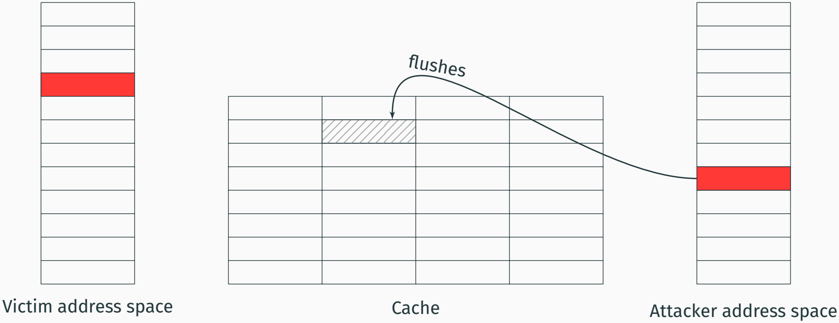
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attacks: Flush+Reload



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

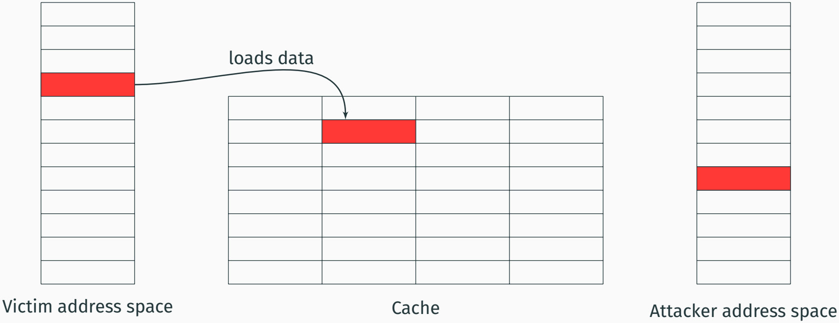
Cache attacks: Flush+Reload



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

Step 2: Attacker flushes the shared cache line

Cache attacks: Flush+Reload

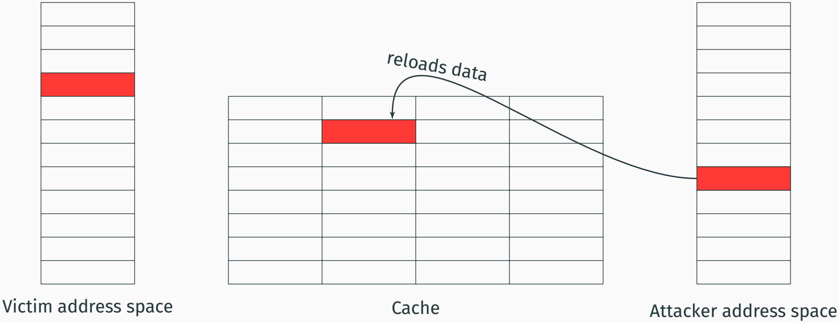


Step 1: Attacker maps shared library (shared memory, in cache)

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

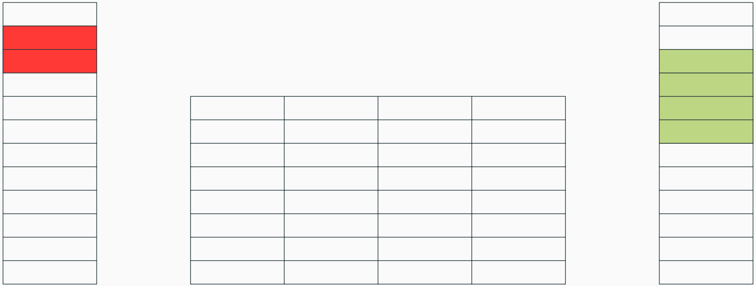
Possible side channels using memory deduplication?

Easy solution #2

Possible side channels using memory deduplication?

Disable memory deduplication!

Cache attacks: Prime+Probe

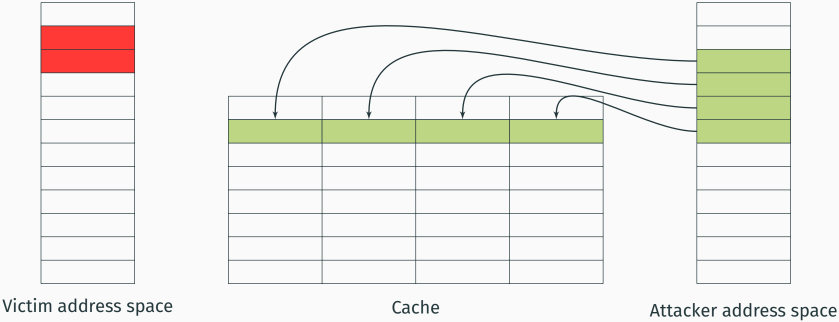


Victim address space

Cache

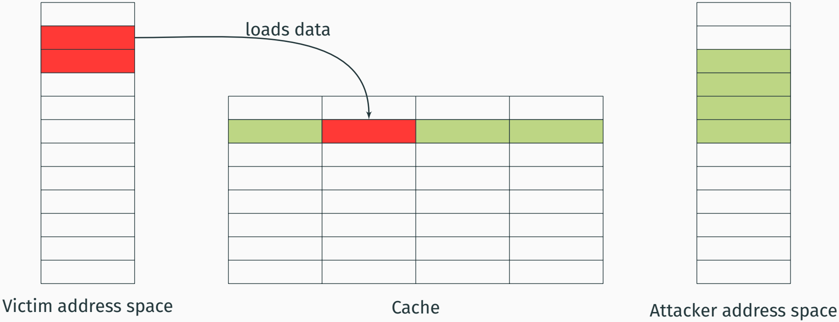
Attacker address space

Cache attacks: Prime+Probe



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

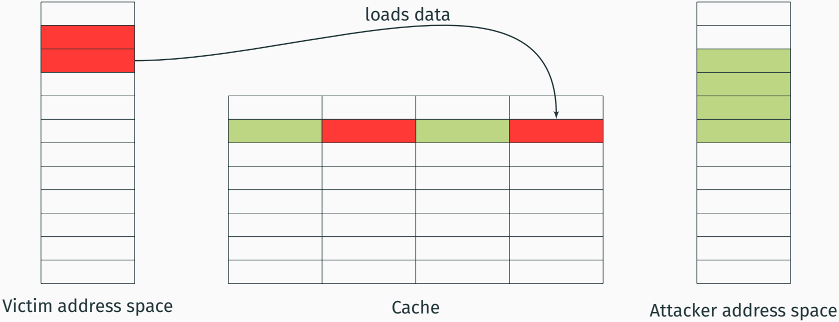
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

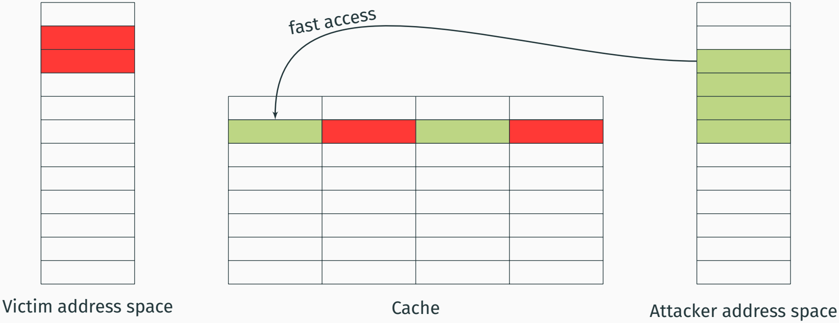
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

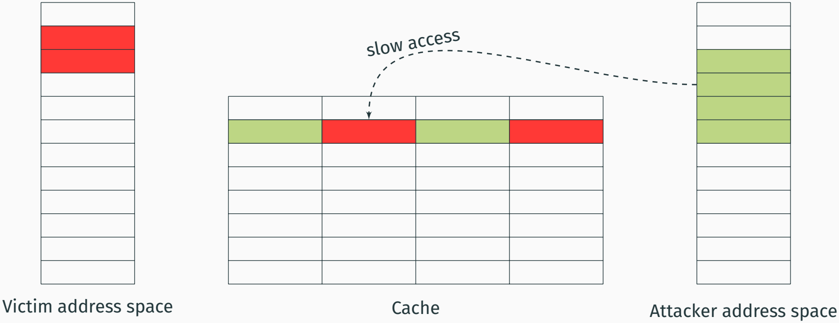


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

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?

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

Increasing the attack surface

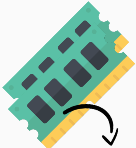
It’s not just caches: DRAM, GPU, TLB, CPU ports, Ring interconnect...!

Translation leak-aside buffer

USENIX Sec'18

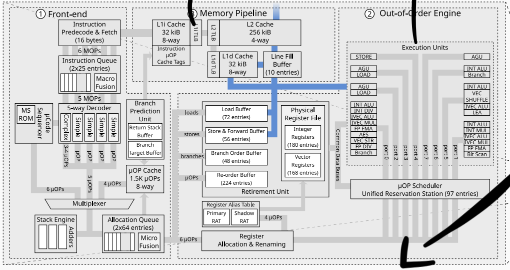
PortSmash

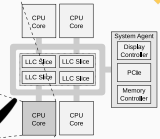
S&P'19



DRAMA

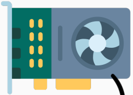
USENIX Sec'16





Lord of the Ring(s)

USENIX Sec'21



Grand Pwning Unit

S&P'18

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 to break security. A novel benefit for the attackers is that 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.

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 regulators 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 operating frequency and voltage settings to minimize power consumption for those demands. Thu
utility of DVFS, hardware and
atively and at very fine

Despite the ubiquity
nisms on commodity syste
eration in the design of thes
of known attacks, given th
software interoperability nee
and time-to-market concerns,
mechanisms have not given mu
rity aspects of these mechanisms,
on optimizing the functional aspe
ment. These combination of facto
vasiveness of these mechanisms ma

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

Daniel Gruss, Clémentine Maurice†, and Stefan Mangard

Graz University of Technology, Austria

ct. A fundamental assumption in software security is that a
ion can only be modified by processes that may write to
n. However, a recent study has shown that parasitic
n change the content of a memory cell without ac-
accessing other memory locations in a high frequency.
whammer bug occurs in most of today's memory mod-
al consequences for the security of all affected systems,
escalation attacks.
d attacks related to Rowhammer so far rely on the avail-
cache flush instruction in order to cause accesses to DRAM
a sufficiently high frequency. We overcome this limitation by
complex cache replacement policies. We show that caches can
d into fast cache eviction to trigger the Rowhammer bug with
gular memory accesses. This allows to trigger the Rowhammer
highly restricted and even scripting environments.
demonstrate a fully automated attack that requires nothing but a
site with JavaScript to trigger faults on remote hardware. Thereby
e can gain unrestricted access to systems of website visitors. We show
hat the attack works on off-the-shelf systems. Existing countermeasures
this new Rowhammer attack.

Drammer: De
on

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

Yanick
UC Santa
yanick@cs.u

Clémentine Mauric
Graz University of Technolog
cmaurice@tugraz.at

Vri

Daniel Gruss
Graz University of Technology
gruss@tugraz.at

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

INTRO

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

Yoongu Kim[1] Ross Daly* Jeremie Kim[1] Chris Fallin* Ji Hye Lee[1]
Donghyuk Lee[1] Chris Wilkerson[2] Konrad Lai Onur Mutlu[1]
[1]Carnegie Mellon University [2]Intel Labs

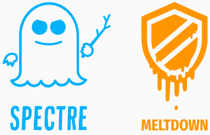
Abstract. *Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology scales down to smaller dimensions, it becomes more difficult to prevent 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]

Recent advances

Transient execution attacks

Transient execution attacks



- novel class of attacks ≠ side-channel attacks

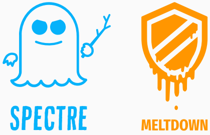
→ transient execution attacks leak the actual target data

- disclosed 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



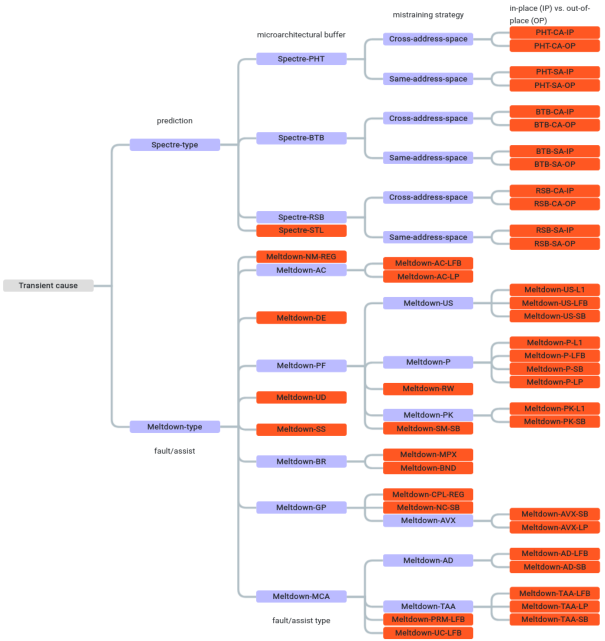
- novel class of attacks $\neq$ side-channel attacks

→ transient execution attacks leak the actual target data

- disclosed 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 cause
prediction
Spectre-type
fault/assist
Meltdown-type
microarchitectural buffer
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Meltdown-NM-REG
Meltdown-AC
Meltdown-DE
Meltdown-PF
Meltdown-UD
Meltdown-SS
Meltdown-BR
Meltdown-GP
Meltdown-MCA
fault/assist type
mistraining strategy
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Meltdown-AC-LFB
Meltdown-AC-LP
Meltdown-US
Meltdown-P
Meltdown-RW
Meltdown-PK
Meltdown-SM-SB
Meltdown-MPX
Meltdown-BND
Meltdown-CPL-REG
Meltdown-NC-SB
Meltdown-AVX
Meltdown-AD
Meltdown-TAA
Meltdown-PRM-LFB
Meltdown-UC-LFB
in-place (IP) vs. out-of-place (OP)
PHT-CA-IP
PHT-CA-OP
PHT-SA-IP
PHT-SA-OP
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP
BTB-SA-OP
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
Meltdown-US-L1
Meltdown-US-LFB
Meltdown-US-SB
Meltdown-P-L1
Meltdown-P-LFB
Meltdown-P-SB
Meltdown-P-LP
Meltdown-PK-L1
Meltdown-PK-SB
Meltdown-AVX-SB
Meltdown-AVX-LP
Meltdown-AD-LFB
Meltdown-AD-SB
Meltdown-TAA-LFB
Meltdown-TAA-LP
Meltdown-TAA-SB

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

- attacker uses a covert channel to encode the secret
- 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

- attacker uses a covert channel to encode the secret
- 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

- attacker uses a covert channel to encode the secret
- 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

- attacker uses a covert channel to encode the secret
- 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

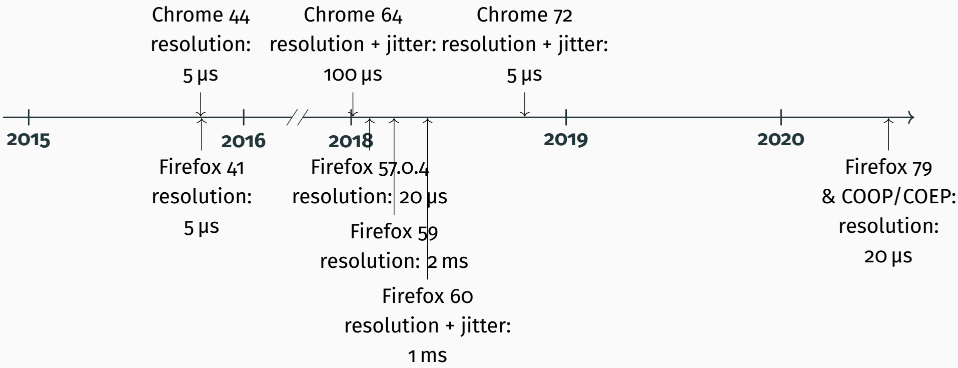
Porting micro-architectural attacks to the Web

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

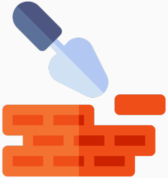
Firefox 59
resolution: 2 ms

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



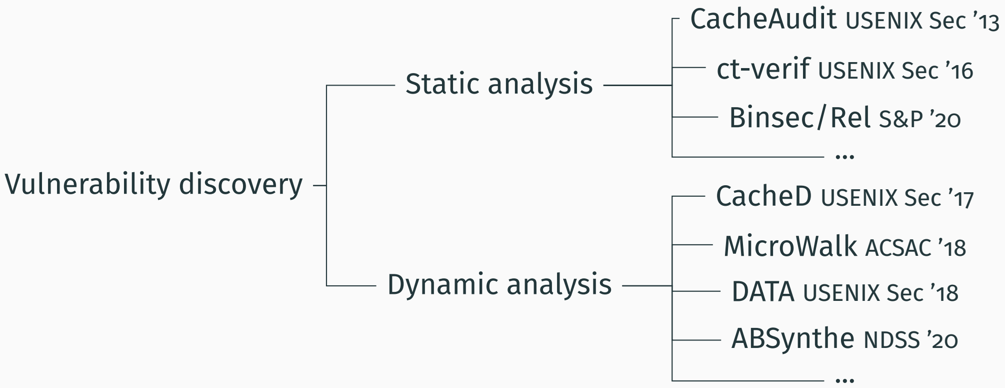
- initial countermeasures: lowering timer resolution
- browsers are adopting better isolation between websites (e.g., Site Isolation) to counter transient execution attacks
- back to higher timer resolution for usability → side-channel attacks are possible again!

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



Vulnerability discovery

Static analysis

- CacheAudit USENIX Sec ’13
- ct-verif USENIX Sec ’16
- Binsec/Rel S&P ’20
- ...

Dynamic analysis

- CacheD USENIX Sec ’17
- MicroWalk ACSAC ’18
- DATA USENIX Sec ’18
- ABSynthe NDSS ’20
- ...

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?

CVE-2018-5407, CVE-2019-1563, CVE-2018-10844, CVE-2018-16868, CVE-2019-19960, CVE-2019-19963, CVE-2020-10932, CVE-2020-11713

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

Evolution of micro-architectural attacks

Clémentine Maurice, CNRS, CRIStAL

17 December 2021—32nd HP/HPE (Virtual) Colloquium On Information Security