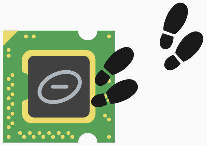
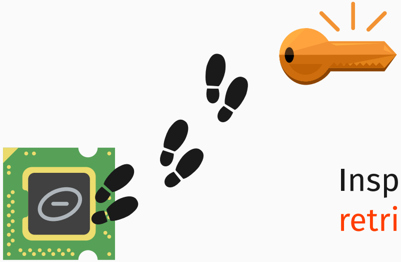
Micro-architectural attacks: from CPU to browser

Clémentine Maurice, CNRS, CRIStAL
@BloodyTangerine

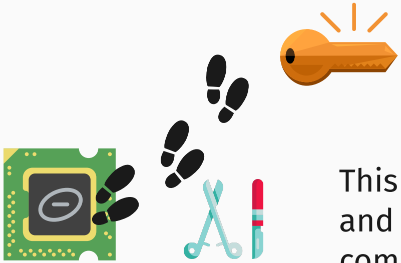
26 October 2022—RAID 2022 keynote



Execution leaves traces in components



Inspecting these traces allows retrieving secrets!



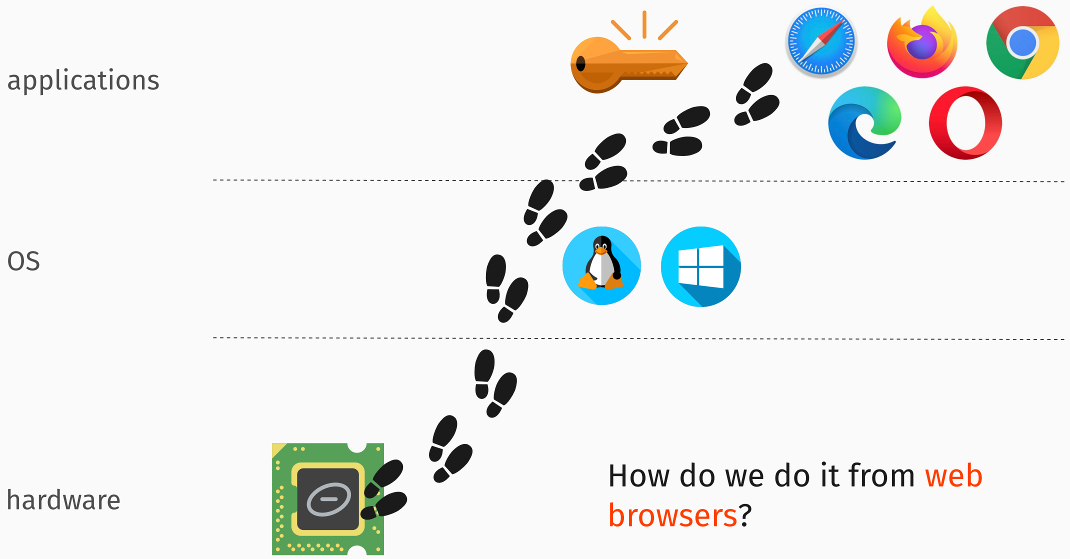
This requires surgical precision and a great control over CPU components...

applications

OS

hardware

This requires surgical precision and a great control over CPU components...



How do we do it from web browsers?

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

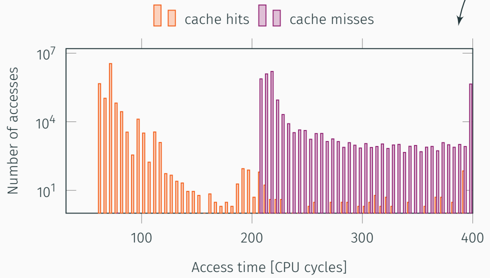
Attacks on micro-architecture

- 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

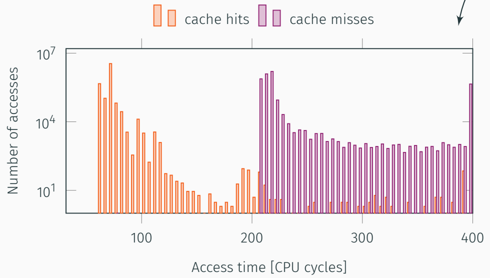
identification



Attacks on micro-architecture

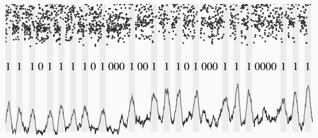
- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing hardware errors
  - side channels: observing side effects of hardware on computations

identification



→

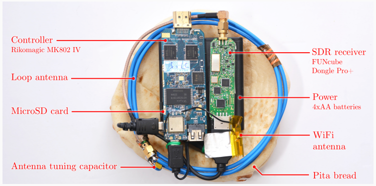
attack



- 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

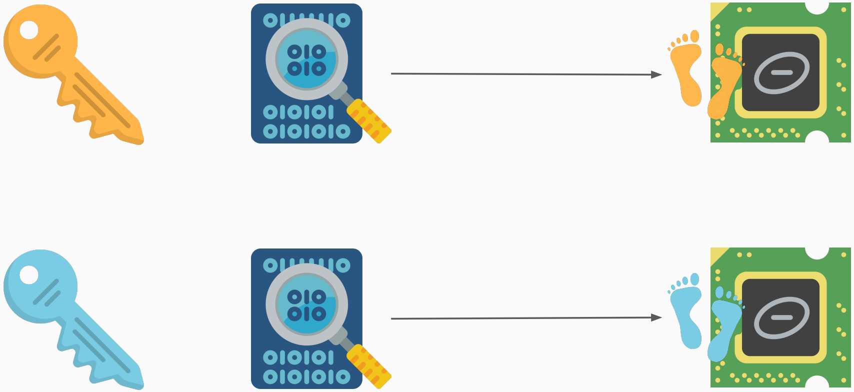
... To microarchitectural side-channel attacks

- 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 $\rightarrow$ hard to detect

Side-channel attacks



Outline

Overview of micro-architectural attacks

Outline

Overview of micro-architectural attacks

Porting micro-architectural attacks to the Web

Overview of micro-architectural attacks

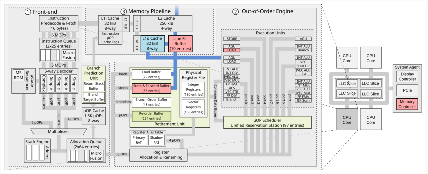
Micro-architectural attacks: Two faces of the same coin

Implementation

Hardware

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

&



1 Front-end
3 Memory Pipeline
2 Out-of-Order Engine
Instruction Predecode & Fetch (16 bytes)
L1i Cache 32 kiB 8-way
L2 Cache 256 kiB 4-way
L1d Cache 32 kiB 8-way
Line Fill Buffer (10 entries)
Instruction Queue (2x25 entries)
5-way Decoder
Branch Prediction Unit
µOP Cache 1.5K µOPs 8-way
MS ROM
Multiplexer
Stack Engine
Allocation Queue (2x64 entries)
Load Buffer (72 entries)
Store & Forward Buffer (56 entries)
Branch Order Buffer (48 entries)
Re-order Buffer (224 entries)
Retirement Unit
Physical Register File
Integer Registers (180 entries)
Vector Registers (168 entries)
Register Alias Table
Register Allocation & Renaming
Execution Units
µOP Scheduler
Unified Reservation Station (97 entries)
CPU Core
LLC Slice
System Agent
Display Controller
PCIe
Memory Controller

Research questions

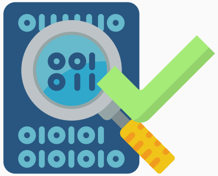
1. Which software implementation is vulnerable?
2. Which hardware component is vulnerable?

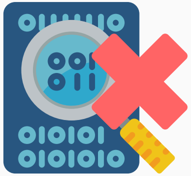
1. Which software implementation is vulnerable?

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
   $\rightarrow$ 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

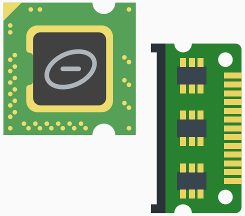




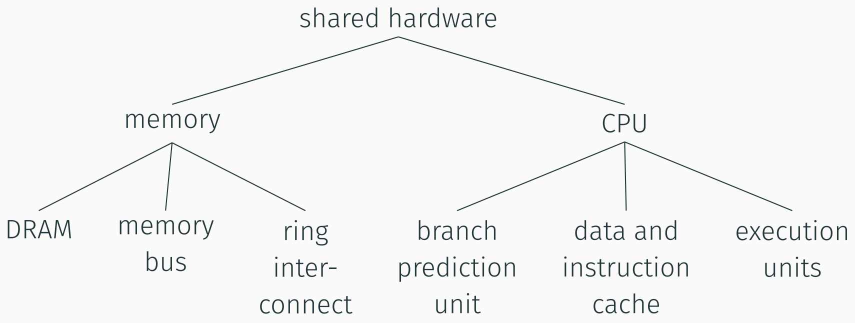
2. Which hardware component leaks information?

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 using a known vulnerability
4. Publish
5. goto step 1



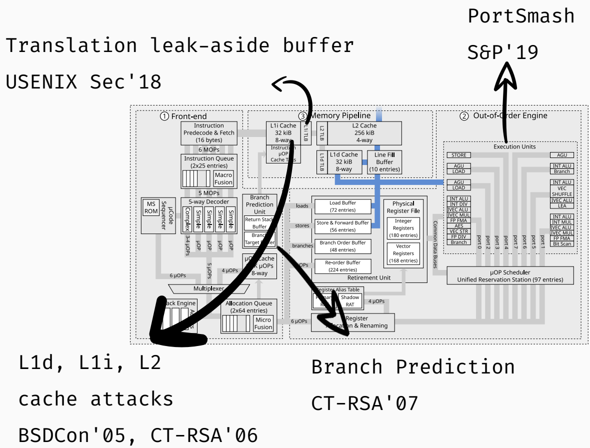
Shared hardware



Each component shared by two processes
is a potential micro-architectural side-channel vector

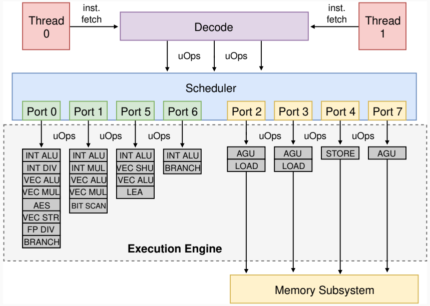
Hyper-threading: Same-core attacks

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



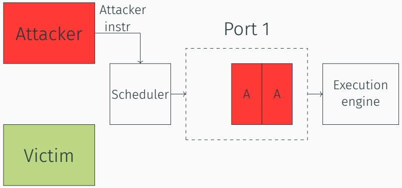
Background: Execution pipeline

- instructions are decomposed in uops to optimize Out-of-Order execution
- uops are dispatched to specialized execution units through CPU ports
- deterministic decomposition of instructions into uops



Port contention

No contention

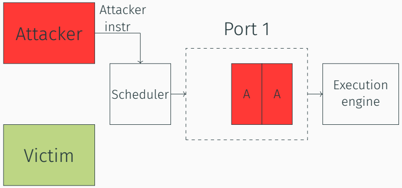


All attacker instructions are executed in a row
→ fast execution time

A. C. Aldaya et al. “Port Contention for Fun and Profit”. In: *S&P*. 2019.

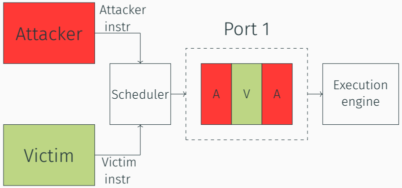
Port contention

No contention



All attacker instructions are executed in a row
→ fast execution time

Contention



Victim instructions delay the attacker instructions
→ slow execution time

A. C. Aldaya et al. "Port Contention for Fun and Profit". In: *S&P*. 2019.

Port contention side-channel attack

Victim

`secret == 0` | `secret == 1`

```
POPCNT %r8,%r8
POPCNT %r8,%r8
...
POPCNT %r8,%r8
POPCNT %r8,%r8
```

Contention on Port 1

```
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
...
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 5

Monitors port usage

Port contention side-channel attack

Victim

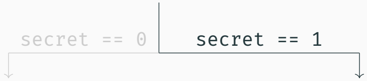
```
secret == 0                    secret == 1

POPCNT %r8,%r8                 VPBROADCASTD %xmm0, %ymm0
POPCNT %r8,%r8                 VPBROADCASTD %xmm0, %ymm0
...                            ...
POPCNT %r8,%r8                 VPBROADCASTD %xmm0, %ymm0
POPCNT %r8,%r8                 VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 1

Secret is 0!

Port contention side-channel attack

Victim



```
POPCNT %r8,%r8                 VPBROADCASTD %xmm0, %ymm0
POPCNT %r8,%r8                 VPBROADCASTD %xmm0, %ymm0
...                            ...
POPCNT %r8,%r8                 VPBROADCASTD %xmm0, %ymm0
POPCNT %r8,%r8                 VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 5

Secret is 1!

Port contention: applications

- end-to-end attack on a TLS server (OpenSSL 1.1.0h): recovers a P-384 ECDSA private key
  - $\rightarrow$ secret dependent on double-and-add operations of `ec_wNAF_mul` point multiplication
- SMoTherSpectre, a speculative code-reuse attack

A. C. Aldaya et al. "Port Contention for Fun and Profit". In: *S&P*. 2019.

A. Bhattacharyya et al. "SMoTherSpectre: Exploiting Speculative Execution through Port Contention". In: *CCS*. 2019.

Easy solution #1

Possible side channels using components shared by a core?

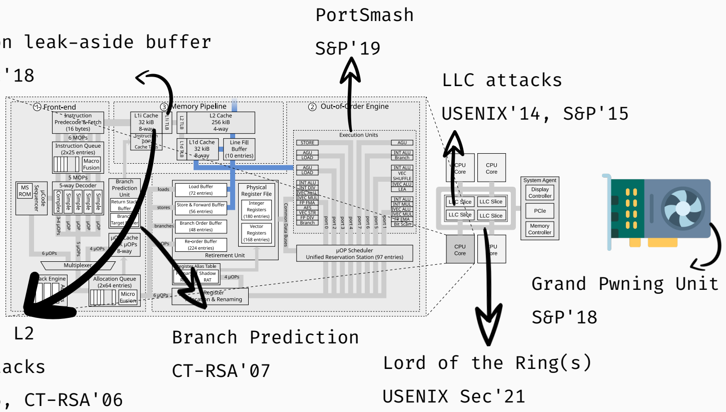
Easy solution #1

Possible side channels using components shared by a core?

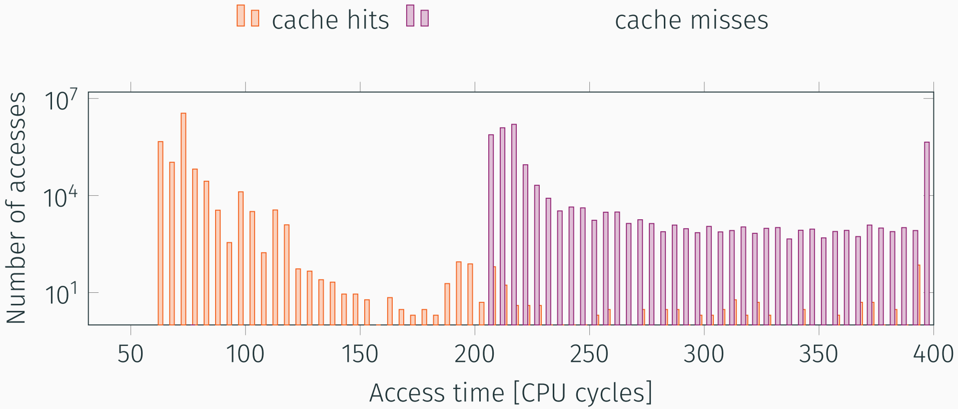
Stop sharing a core!

Cross-core attacks!

- cores also share resources: L3 cache, Ring Interconnect, GPU...



Cache timing differences



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
- even more interest in 2018 after the disclosure of Spectre and Meltdown

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.

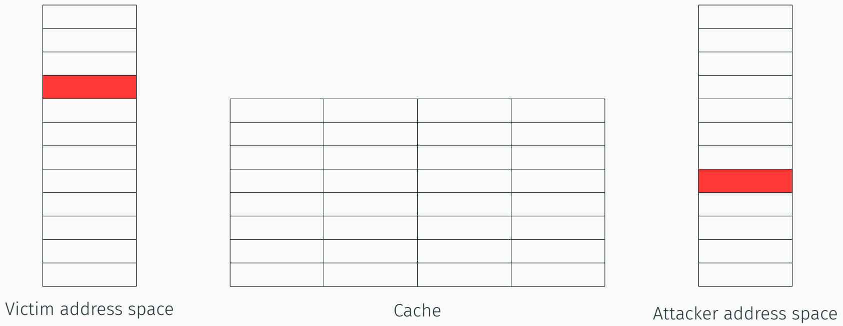
D. A. Osvik, A. Shamir, and E. Tromer. “Cache Attacks and Countermeasures: the Case of AES”. In: *CT-RSA 2006*. 2006.

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

P. Kocher et al. “Spectre Attacks: Exploiting Speculative Execution”. In: *S&P*. 2019.

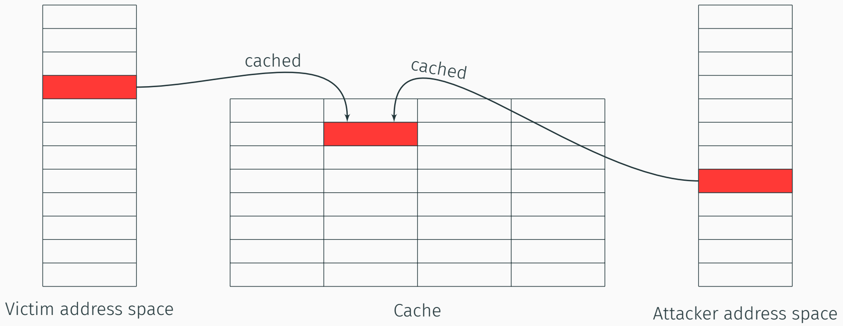
M. Lipp et al. “Meltdown: Reading Kernel Memory from User Space”. In: *USENIX Security Symposium*. 2018.

Cache attacks: Flush+Reload



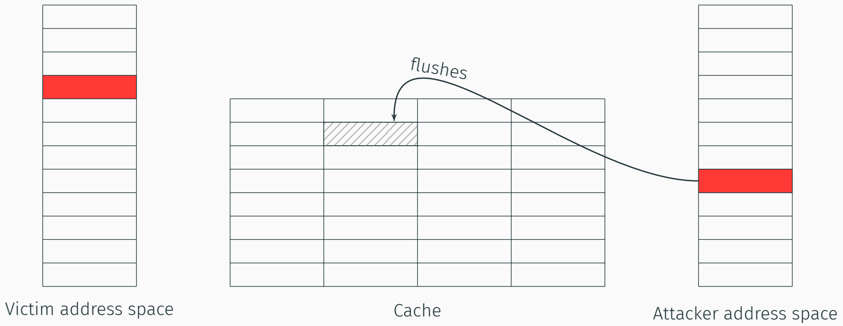
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attacks: Flush+Reload



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

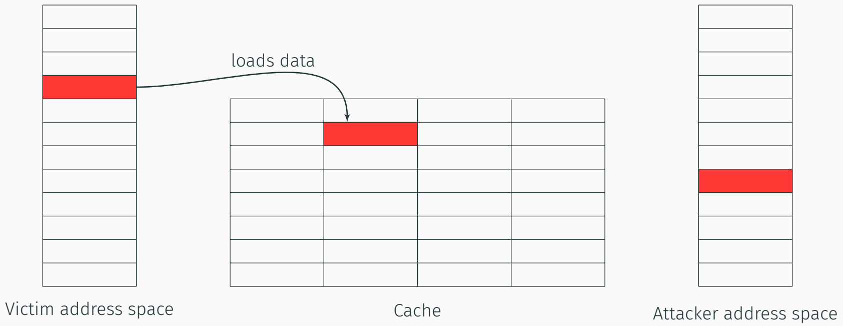
Cache attacks: Flush+Reload



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

Step 2: Attacker flushes the shared cache line

Cache attacks: Flush+Reload

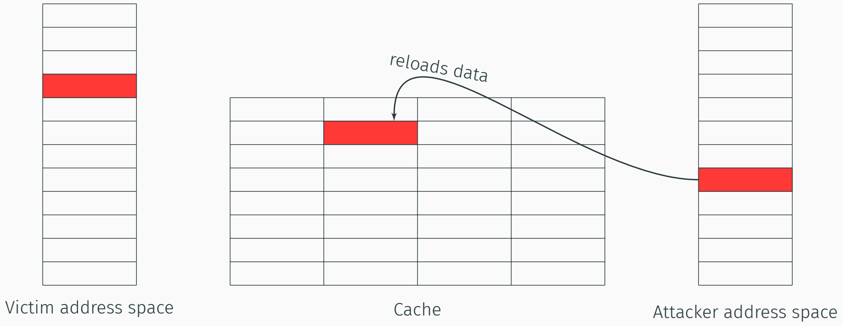


Step 1: Attacker maps shared library (shared memory, in cache)

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

- side channel attacks on cryptographic primitives:
  - RSA: 96.7% of secret key bits in a single signature
  - AES: full key recovery in 30000 dec. (a few seconds)
- attacks against pseudorandom number generators
- attacks against RSA key generation
- revival of Bleichenbacher attacks on TLS

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: *COSADE*. 2015.

S. Cohney et al. “Pseudorandom Black Swans: Cache Attacks on CTR_DRBG”. In: *S&P*. 2020.

A. C. Aldaya et al. “Cache-Timing Attacks on RSA Key Generation”. In: *TCHES* (2019).

E. Ronen et al. “The 9 Lives of Bleichenbacher’s CAT: New Cache ATtacks on TLS Implementations”. In: *S&P*. 2019.

Easy solution #2

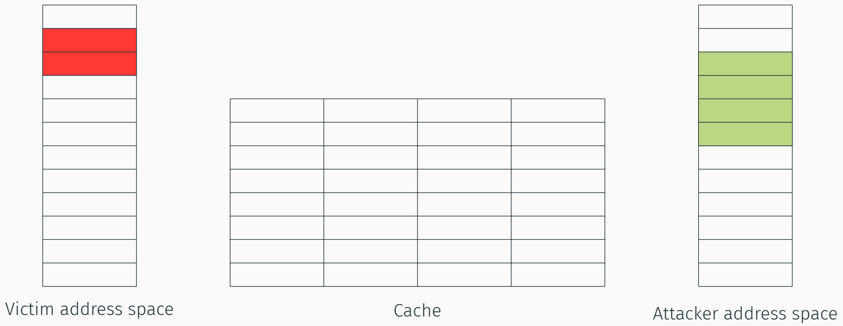
Possible side channels using memory deduplication?

Easy solution #2

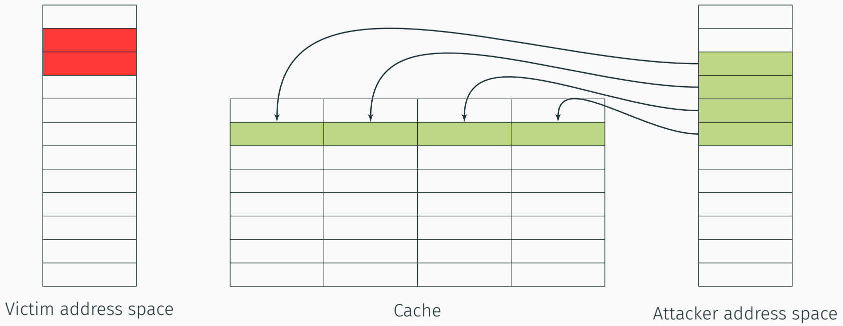
Possible side channels using memory deduplication?

Disable memory deduplication!

Cache attacks: Prime+Probe

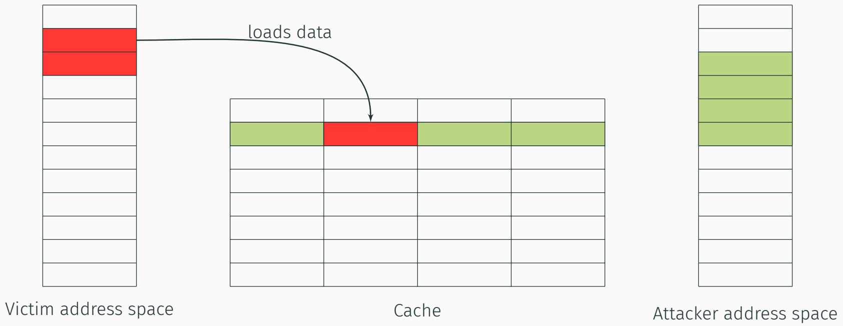


Cache attacks: Prime+Probe



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

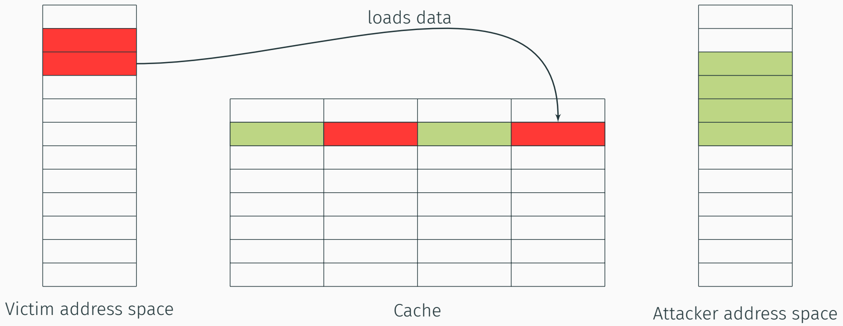
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

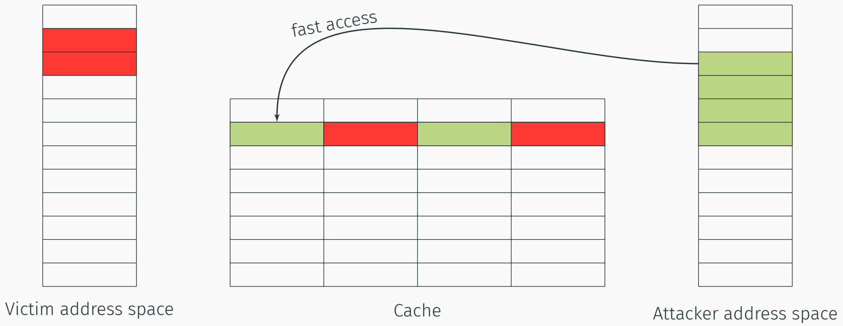
Cache attacks: Prime+Probe



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

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

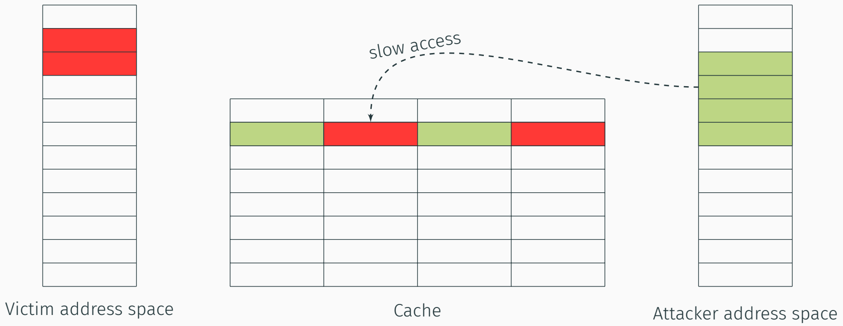


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

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 implementations:
    - El Gamal (sliding window): full key recovery in 12 min.
- 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.

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

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

Prime+Probe: Applications

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

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

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

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

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!?

Cross-CPU attacks!

- CPUs also share resources: DRAM

Translation leak-aside buffer

USENIX Sec'18

PortSmash

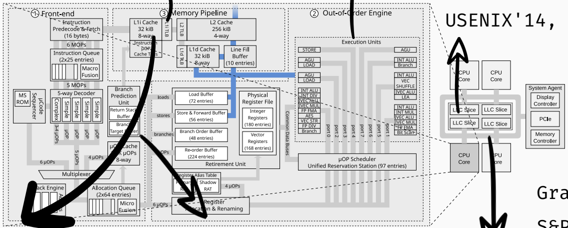
S&P'19

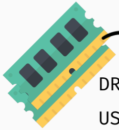
LLC attacks

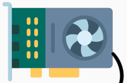
USENIX'14, S&P'15

DRAMA

USENIX Sec'16







Grand Pwning Unit

S&P'18

L1d, L1i, L2

cache attacks

BSDCon'05, CT-RSA'06

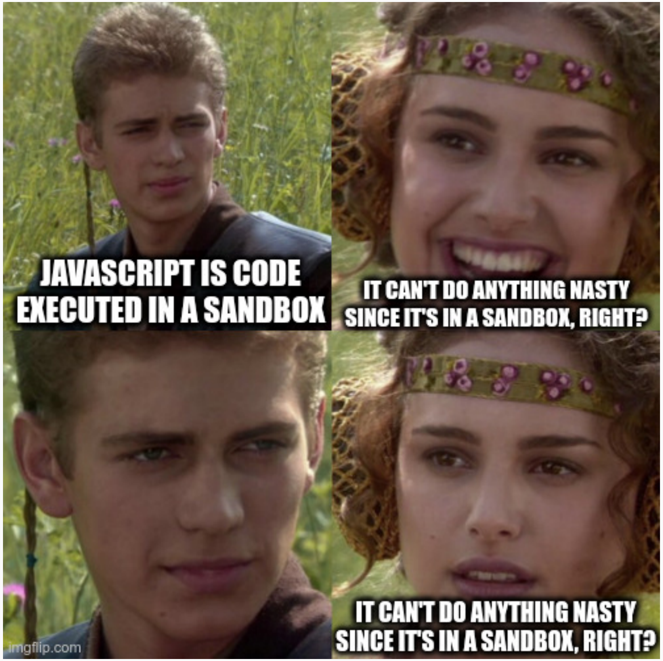
Branch Prediction

CT-RSA'07

Lord of the Ring(s)

USENIX Sec'21

Porting micro-architectural attacks to the Web


JAVASCRIPT IS CODE EXECUTED IN A SANDBOX
IT CAN'T DO ANYTHING NASTY SINCE IT'S IN A SANDBOX, RIGHT?
IT CAN'T DO ANYTHING NASTY SINCE IT'S IN A SANDBOX, RIGHT?
imgflip.com

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

Side-channel attacks in JavaScript?

- side channels are only doing benign operations

Side-channel attacks in JavaScript?

- side channels are only doing benign operations
    - all side-channel attacks: measuring time

Side-channel attacks in JavaScript?

- side channels are only doing benign operations
    - all side-channel attacks: measuring time
    - cache attacks: accessing their own memory
    - port contention attacks: executing specific instructions

Measuring time

High-resolution timers?

- measure small timing differences: need a high-resolution timer

High-resolution timers?

- measure small timing differences: need a high-resolution timer
- native: `rdtsc`, timestamp in CPU cycles

High-resolution timers?

- measure small timing differences: need a high-resolution timer
- native: `rdtsc`, timestamp in CPU cycles
- JavaScript: `performance.now()` has the highest resolution

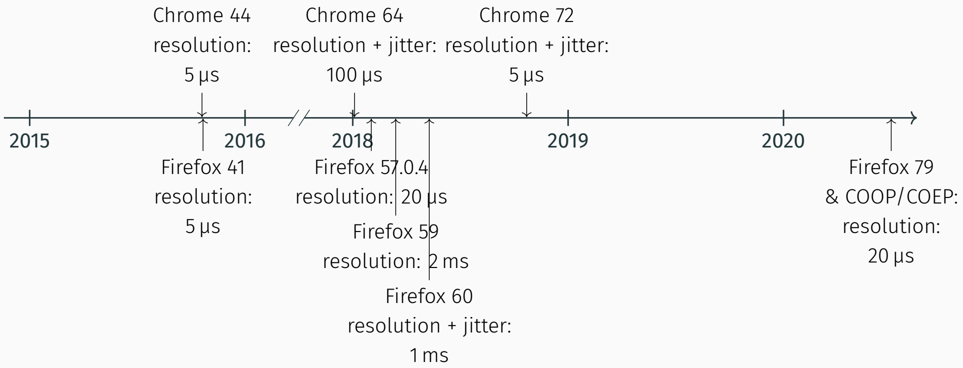
High-resolution timers?

- measure small timing differences: need a high-resolution timer
- native: `rdtsc`, timestamp in CPU cycles
- JavaScript: `performance.now()` has the highest resolution

`performance.now()`

[...] represent times as floating-point numbers with up to microsecond precision.
— Mozilla Developer Network

Evolution of timers until today



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

It was better before

- before September 2015: `performance.now()` had a nanosecond resolution

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS'15*. 2015.
`https://www.mozilla.org/en-US/security/advisories/mfsa2015-114/`

It was better before

- before September 2015: `performance.now()` had a nanosecond resolution
- Oren et al. demonstrated cache side-channel attacks in JavaScript

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015.
`https://www.mozilla.org/en-US/security/advisories/mfsa2015-114/`

It was better before

- before September 2015: `performance.now()` had a nanosecond resolution
- Oren et al. demonstrated cache side-channel attacks in JavaScript
- “fixed” in Firefox 41: rounding to 5 µs

Y. Oren et al. “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications”. In: *CCS’15*. 2015.
`https://www.mozilla.org/en-US/security/advisories/mfsa2015-114/`

We can do better!

- microsecond resolution is not enough

M. Schwarz et al. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC*. 2017.

We can do better!

- microsecond resolution is not enough
- two approaches

M. Schwarz et al. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC*. 2017.

We can do better!

- microsecond resolution is not enough
- two approaches
    1. recover a higher resolution from the available timer

M. Schwarz et al. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC*. 2017.

We can do better!

- microsecond resolution is not enough
- two approaches
    1. recover a higher resolution from the available timer
    2. build our own high-resolution timer

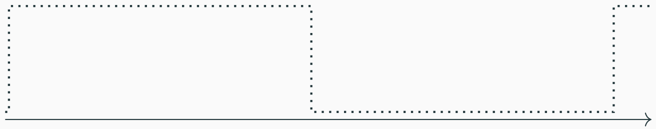
M. Schwarz et al. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC*. 2017.

Recovering resolution: Clock interpolation

- measure how often we can increment a variable between two timer ticks

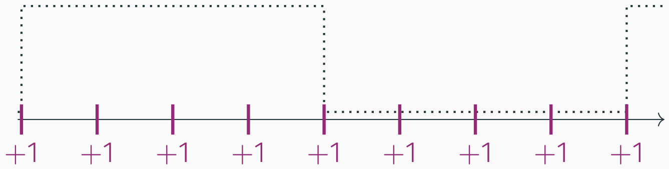
Recovering resolution: Clock interpolation

- measure how often we can increment a variable between two timer ticks



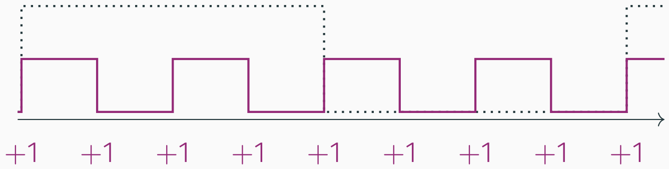
Recovering resolution: Clock interpolation

- measure how often we can increment a variable between two timer ticks



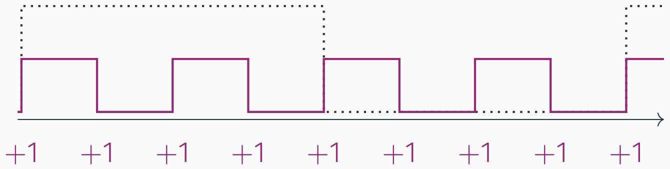
Recovering resolution: Clock interpolation

- measure how often we can increment a variable between two timer ticks



Recovering resolution: Clock interpolation

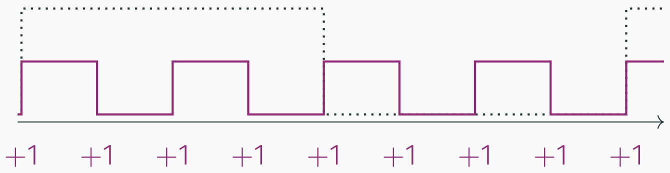
- measure how often we can increment a variable between two timer ticks



- to measure with high resolution

Recovering resolution: Clock interpolation

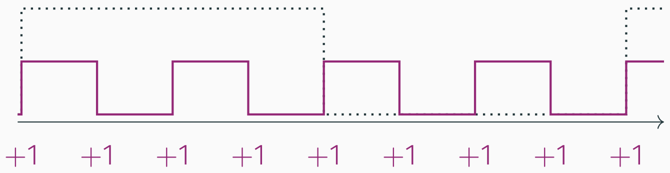
- measure how often we can increment a variable between two timer ticks



- to measure with high resolution
  - start measurement at clock edge

Recovering resolution: Clock interpolation

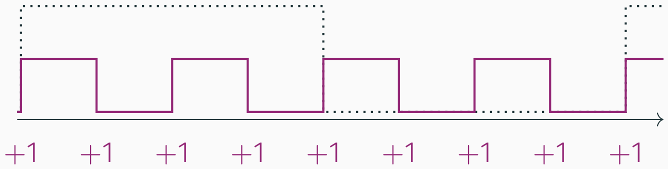
- measure how often we can increment a variable between two timer ticks



- to measure with high resolution
  - start measurement at clock edge
  - increment a variable until next clock edge

Recovering resolution: Clock interpolation

- measure how often we can increment a variable between two timer ticks



- to measure with high resolution
  - start measurement at clock edge
  - increment a variable until next clock edge
- Firefox/Chrome: 500 ns, Tor: 15 µs

Building a timer: Web worker

- feature to share data: `SharedArrayBuffer`

Building a timer: Web worker

- feature to share data: `SharedArrayBuffer`
- web worker can simultaneously read/write data

Building a timer: Web worker

- feature to share data: `SharedArrayBuffer`
- web worker can simultaneously read/write data
- no message passing overhead

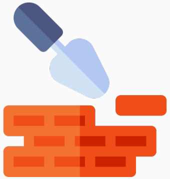
Building a timer: Web worker

- feature to share data: `SharedArrayBuffer`
- web worker can simultaneously read/write data
- no message passing overhead
- one dedicated worker for incrementing the shared variable

Building a timer: Web worker

- feature to share data: `SharedArrayBuffer`
- web worker can simultaneously read/write data
- no message passing overhead
- one dedicated worker for incrementing the shared variable
- Firefox/Fuzzyfox: 2 ns, Chrome: 15 ns

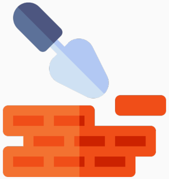
Jitter?



- lowering timer resolution is not enough
- adding jitter $\rightarrow$ makes clock interpolation inefficient (need to redo the measurements to get rid of noise)

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

Jitter?

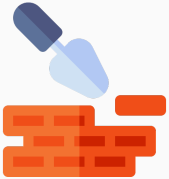


- lowering timer resolution is not enough
- adding jitter $\rightarrow$ makes clock interpolation inefficient (need to redo the measurements to get rid of noise)

$\rightarrow$ has no impact on SharedArrayBuffers!

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

Jitter?



- lowering timer resolution is not enough
- adding jitter $\rightarrow$ makes clock interpolation inefficient (need to redo the measurements to get rid of noise)

$\rightarrow$ has no impact on SharedArrayBuffers!

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

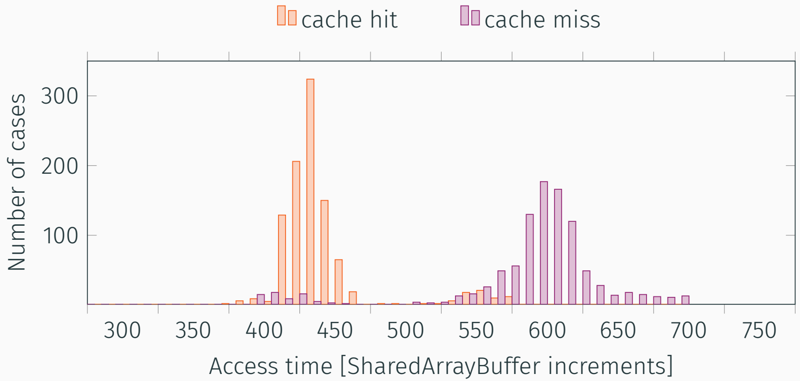
T. Rokicki, C. Maurice, and P. Laperdrix. “Sok: In search of lost time: A review of javascript timers in browsers”. In: *EuroS&P’21*. 2021

Cache attacks in browsers

Cache attacks: Challenges with JavaScript

1. No high-resolution timers
2. No instruction to flush the cache
3. No knowledge about physical addresses

Eviction sets in JavaScript

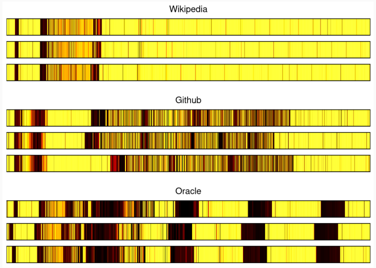


→ we can distinguish cache hits from cache misses (only ≈ 150 cycles difference)!

M. Schwarz et al. “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: *FC*. 2017.

Cache attacks in JavaScript: applications

- spying on user behavior: detect mouse and network activity
- covert channel across origins
- covert channel host-to-VM
- website fingerprinting



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

A. Shusterman et al. “Robust Website Fingerprinting Through the Cache Occupancy Channel”. In: *USENIX Security Symposium*. 2019.

Other micro-architectural attacks in browsers?

Other micro-architectural attacks in browsers

Port Contention Goes Portable: Port Contention Side Channels in Web Browsers

Thomas Rokicki
Univ Rennes, CNRS, IRISA
Rennes, France

Marina Botvinnik
Ben-Gurion University of the Negev
Be'er Sheva, Israël

Clémentine Maurice
Univ Lille, CNRS, Inria
Lille, France

Yossi Oren
Ben-Gurion University of the Negev
Be'er Sheva, Israël

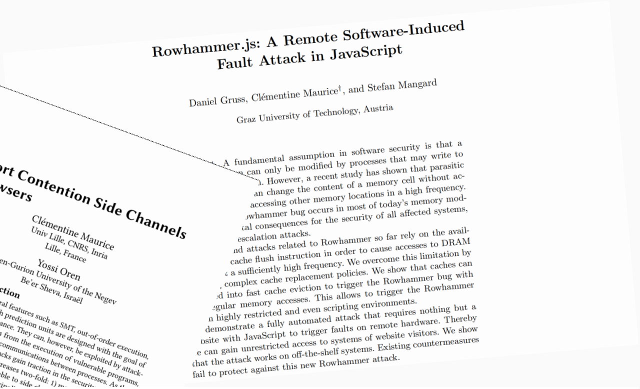
Abstract

Microarchitectural side-channel attacks can derive secrets from the execution of vulnerable programs. Their implementation in web browsers represents a considerable extension of their attack surface, as a user simply browsing a malicious website, or even a malicious third-party advertisement in a benign cross-origin isolated website, can be a victim.

In this paper, we present the first port contention side channel running entirely in a web browser, despite a highly challenging environment. Our attack can be used to build a cross-browser covert channel with a bit rate of 200 bit/s, one order of magnitude above the state of the art, and has a spatial resolution of 1024 native instructions in a side-channel attack, a performance on-par with Prime+Probe attacks. We provide a framework to evaluate the port contention caused by WebAssembly instructions on Intel processors, allowing to increase the portability of port contention side channels. We conclude from our work that port contention attacks are not only fast, they are also less susceptible to noise than cache attacks, and are immune to countermeasures implemented in browsers as well as most side channel countermeasures, which target the cache in their vast majority.

1 Introduction

Microarchitectural features such as SMT, out-of-order execution,
caches and branch prediction units are designed with the goal of
increasing performance. They can, however, be exploited by attack-
ers to derive secrets from the execution of vulnerable programs,
and to enable covert communications between processes. As th
microarchitectural attacks gain traction in the securit
their attack surface increases two-fold: 1) m
nents are found vulnerable to side
attacks, which were origin
being ported to web brows
crucially increasing the num
While cache side-channel a
attacks most studied in the liter
attacks have also been shown t
technique introduced in 2018 by A
This attack on Intel CPUs is based
ports act as a bottleneck in the exec
with the victim, the attacker can exp
by the contention of different instruc
temporal resolution and can be used,

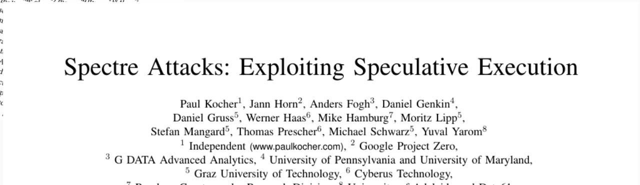


Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript

Daniel Gruss, Clémentine Maurice[†], and Stefan Mangard

Graz University of Technology, Austria

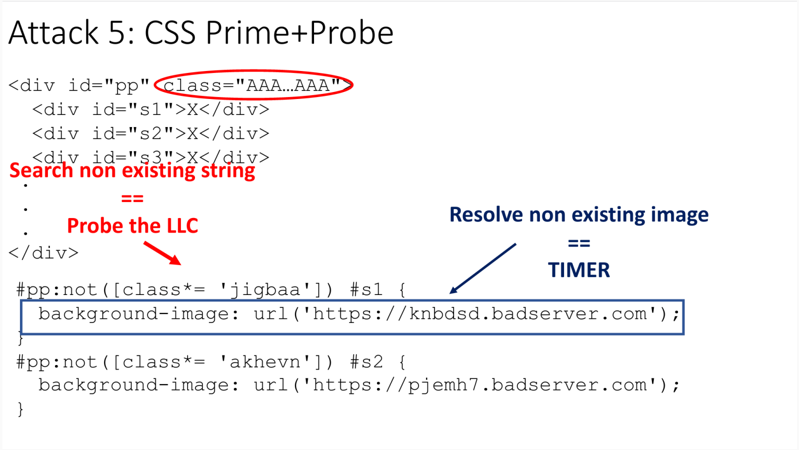
A fundamental assumption in software security is that a
can only be modified by processes that may write to
. However, a recent study has shown that parasitic
an change the content of a memory cell without ac-
accessing other memory locations in a high frequency.
owhammer bug occurs in most of today's memory mod-
al consequences for the security of all affected systems,
escalation attacks.
nd 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
egular memory accesses. This allows to trigger the Rowhammer
n highly restricted and even scripting environments.
demonstrate a fully automated attack that requires nothing but a
bsite with JavaScript to trigger faults on remote hardware. Thereby
e can gain unrestricted access to systems of website visitors. We show
that the attack works on off-the-shelf systems. Existing countermeasures
fail to protect against this new Rowhammer attack.



Spectre Attacks: Exploiting Speculative Execution

Paul Kocher[1], Jann Horn[2], Anders Fogh[3], Daniel Genkin[4],
Daniel Gruss[5], Werner Haas[6], Mike Hamburg[7], Moritz Lipp[5],
Stefan Mangard[5], Thomas Prescher[6], Michael Schwarz[5], Yuval Yarom[8]
[1] Independent (www.paulkocher.com), [2] Google Project Zero,
[3] G DATA Advanced Analytics, [4] University of Pennsylvania and University of Maryland,
[5] Graz University of Technology, [6] Cyberus Technology,

Bonus: you don’t even need JavaScript!



Attack 5: CSS Prime+Probe

```
<div id="pp" class="AAA…AAA">
  <div id="s1">X</div>
  <div id="s2">X</div>
  <div id="s3">X</div>
   .
   .
   .
</div>

 #pp:not([class*= 'jigbaa']) #s1 {
   background-image: url('https://knbdsd.badserver.com');
 }
 #pp:not([class*= 'akhevn']) #s2 {
   background-image: url('https://pjemh7.badserver.com');
 }
```

Search non existing string == Probe the LLC

Resolve non existing image == TIMER

A. Shusterman et al. “Prime+Probe 1, JavaScript 0: Overcoming Browser-based Side-Channel Defenses”. In: *USENIX Security Symposium*. 2021.

Conclusions

- first paper by Kocher in 1996: 25 years of research in this area

Conclusions

- first paper by Kocher in 1996: 25 years of research in this area
- domain still in expansion: increasing number of papers published since 2015

Conclusions

- first paper by Kocher in 1996: 25 years of research in this area
- domain still in expansion: increasing number of papers published since 2015
- any shared component is a potential side-channel vector

Conclusions

- first paper by Kocher in 1996: 25 years of research in this area
- domain still in expansion: increasing number of papers published since 2015
- any shared component is a potential side-channel vector
- it's **really** hard not to share a component

Conclusions

- first paper by Kocher in 1996: 25 years of research in this area
- domain still in expansion: increasing number of papers published since 2015
- any shared component is a potential side-channel vector
- it's **really** hard not to share a component
- micro-architectural attacks require a low-level understanding and control over the components, usually achieved with native code

Conclusions

- first paper by Kocher in 1996: 25 years of research in this area
- domain still in expansion: increasing number of papers published since 2015
- any shared component is a potential side-channel vector
- it's **really** hard not to share a component
- micro-architectural attacks require a low-level understanding and control over the components, usually achieved with native code
- but it's still possible to carry these attacks on from web browsers

Thank you!

Contact

✉ `clementine.maurice@inria.fr`

@BloodyTangerine

Micro-architectural attacks: from CPU to browser

Clémentine Maurice, CNRS, CRIStAL
@BloodyTangerine

26 October 2022—RAID 2022 keynote