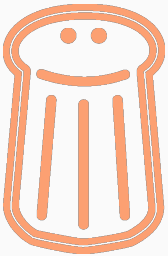
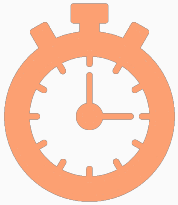
In Search of Lost Time: A Review of JavaScript Timers in Browsers

Thomas Rokicki
Clémentine Maurice
Pierre Laperdrix

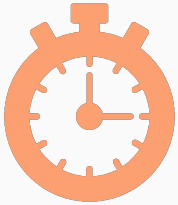
Pass The Salt - 07/07/21



JavaScript Timing Attacks



JavaScript **Timing** Attacks

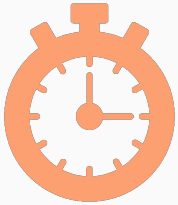


JavaScript-based timing attacks

JavaScript **Timing** Attacks

↓

Exploit timing differences to infer secrets from the JavaScript sandbox.



JavaScript-based timing attacks

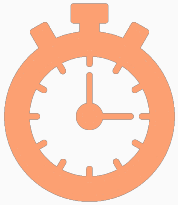
JavaScript **Timing** Attacks

↓

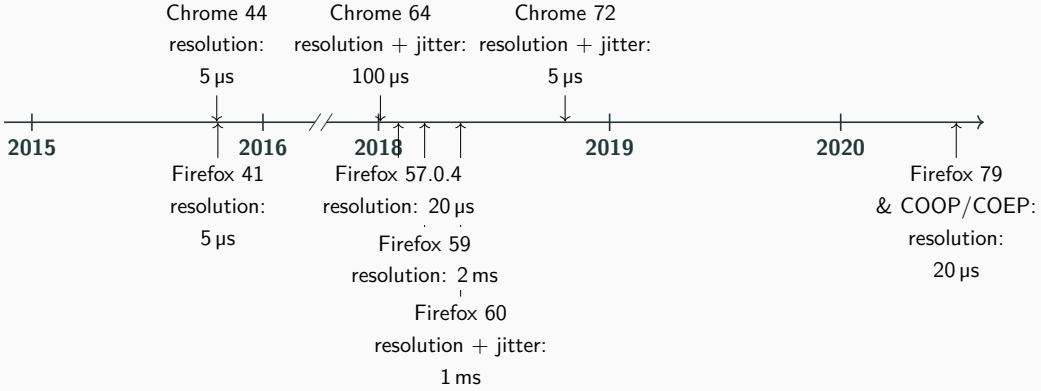
Exploit timing differences to infer secrets from the JavaScript sandbox.

↓

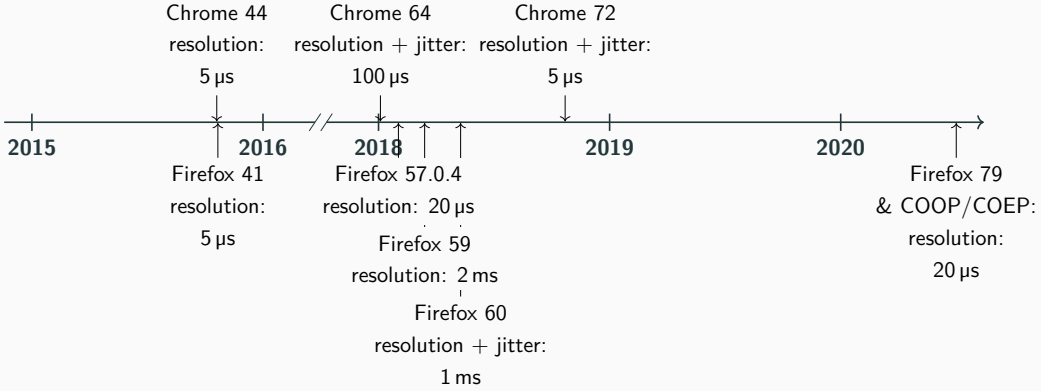
Resolution of 10 -100 ns



JS and timers: A complicated history



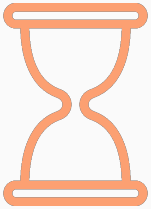
JS and timers: A complicated history



What are the security implications of changing the timers' resolution?

Classification of JavaScript timing attacks

- Hardware-contention-based attacks
- Transient execution attacks
- Attacks based on system resources
- Attacks based on browser resources



Classification of JavaScript timing attacks

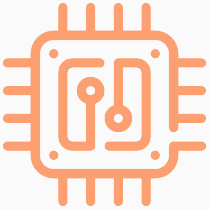
- **Hardware-contention-based attacks**

  Principle: The attacker infers secrets from timing differences caused by hardware state

  Prerequisites: High resolution timers & Shared hardware resources

  Examples: JavaScript Prime+Probe, Rowhammer.js

- Transient execution attacks
- Attacks based on system resources
- Attacks based on browser resources



Classification of JavaScript timing attacks

- Hardware-contention-based attacks
- **Transient execution attacks**

  Principle: The attacker infers secrets from traces of transient execution on the hardware.

  Prerequisites: Transient execution, high resolution timers & shared hardware resources

  Examples: Spectre, RIDL
- Attacks based on system resources
- Attacks based on browser resources

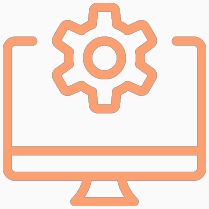
Classification of JavaScript timing attacks

- Hardware-contention-based attacks
- Transient execution attacks
- **Attacks based on system resources**

  Principle: The attacker infers secrets from shared system resources.

  Prerequisites: High resolution timers & shared system resources.

  Examples: Keystroke attacks, memory deduplication attacks.
- Attacks based on browser resources



Classification of JavaScript timing attacks

- Hardware-contention-based attacks
- Transient execution attacks
- Attacks based on system resources
- **Attacks based on browser resources**

  Principle: The attacker infers secrets from shared browser resources.

  Prerequisites: High resolution timers & shared browser resources.

  Examples: History sniffing, fingerprinting.

JavaScript Timers

```
performance.now()
```

JavaScript Timers

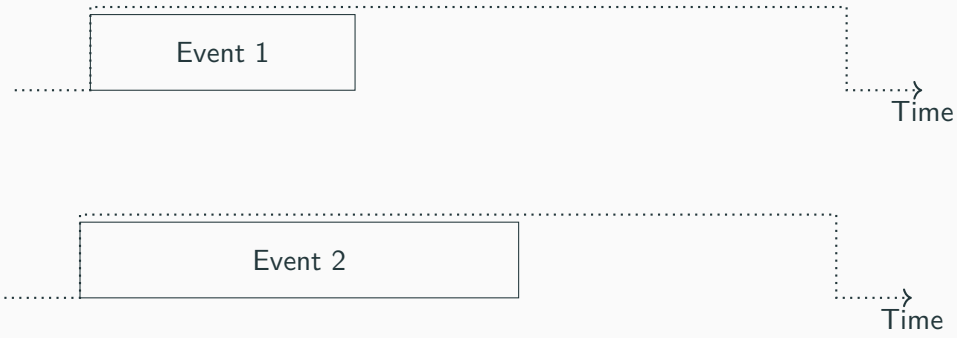
`performance.now()` : Resolution ranges from 5 µs to 1 ms.

JavaScript Timers

`performance.now()` : Resolution ranges from 5 µs to 1 ms.

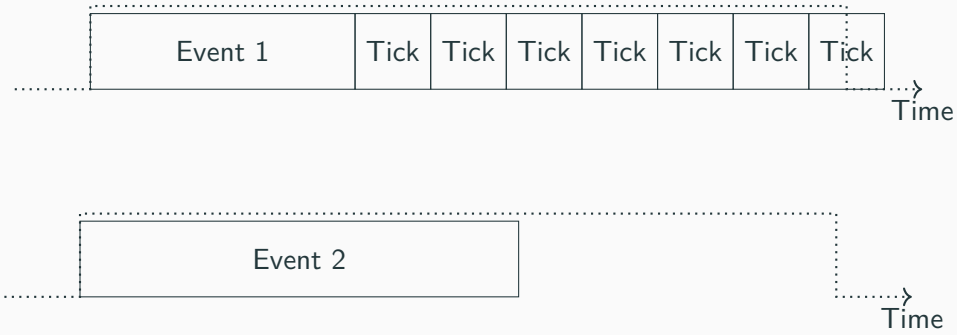
We need to time events in the order of 10 ns.

performance.now() interpolation



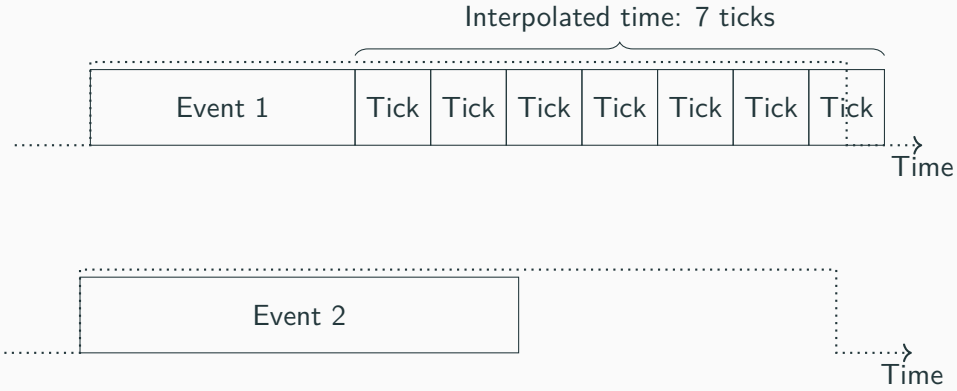
Michael Schwarz et al. "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript". In: International Conference on Financial Cryptography and Data Security. 2017

`performance.now()` interpolation



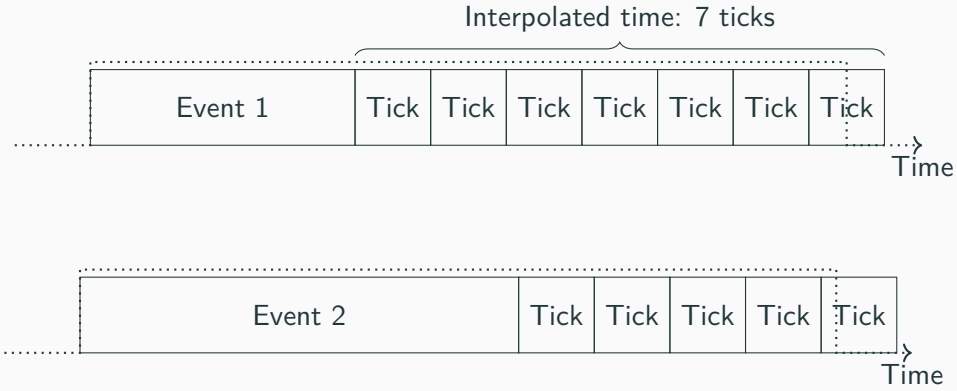
Schwarz et al., "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript"

`performance.now()` interpolation



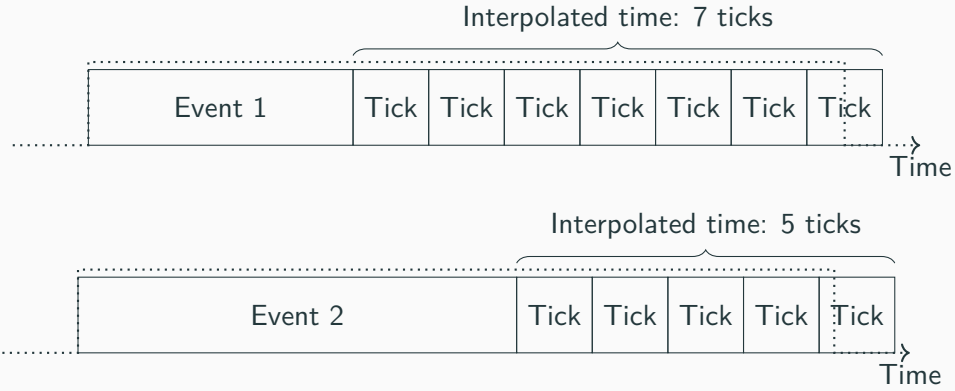
Schwarz et al., "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript"

`performance.now()` interpolation



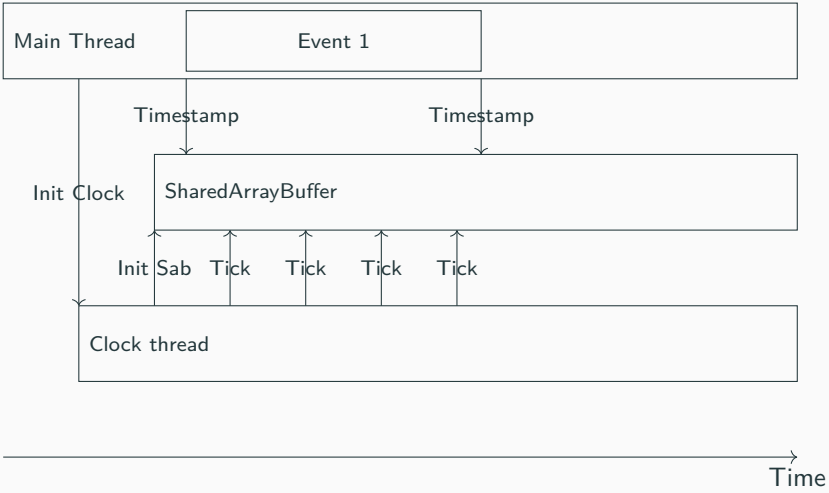
Schwarz et al., "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript"

`performance.now()` interpolation



Schwarz et al., "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript"

SharedArrayBuffer



Event one lasts 4 ticks

How to remove timers

Reducing the resolution alone is not sufficient because of interpolation. [1]

[1] This applies to other all timing-based functions such as callbacks, animation functions and others.

How to remove timers

Reducing the resolution alone is not sufficient because of interpolation. [1]

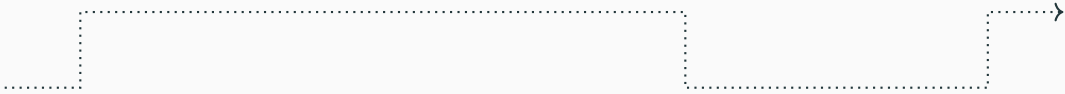
Add **jitter** to the measurement.

[1] This applies to other all timing-based functions such as callbacks, animation functions and others.

How to remove timers

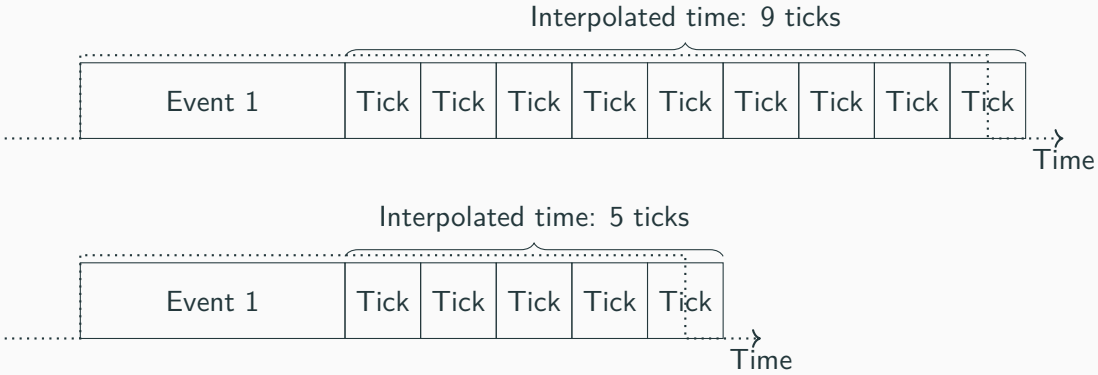
Reducing the resolution alone is not sufficient because of interpolation. [1]

Add **jitter** to the measurement.

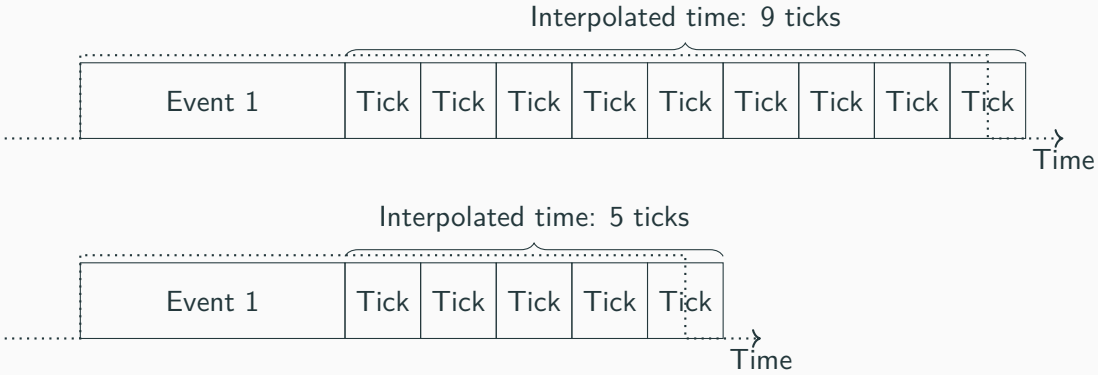


[1] This applies to other all timing-based functions such as callbacks, animation functions and others.

Interpolation and jitter



Interpolation and jitter



Firefox: 1 ms with jitter.

Chrome: 100 µs with jitter.

What can we do about `SharedArrayBuffer`?

Disable them.

What can we do about `SharedArrayBuffer`?

Disable them.

`SharedArrayBuffer` were disabled on Firefox 58 and Chrome 64

Security vs Practicality

- High resolution timers useful for performance measurements, network, animation
- `SharedArrayBuffer` are an important part of the evolution of JavaScript from a single threaded language to multithreading

Security vs Practicality

- High resolution timers useful for performance measurements, network, animation
- `SharedArrayBuffer` are an important part of the evolution of JavaScript from a single threaded language to multithreading

Browser vendors want more efficient, less penalizing countermeasures.

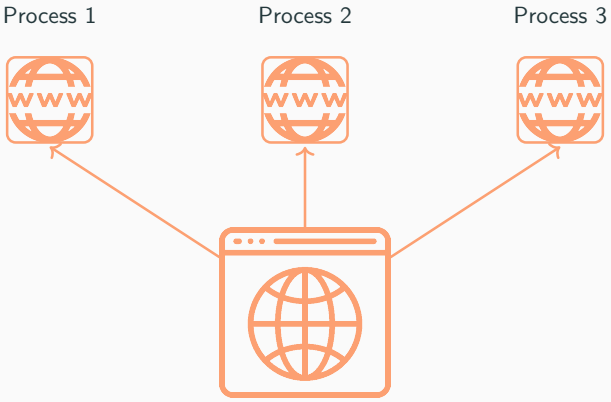
Security vs Practicality

- High resolution timers useful for performance measurements, network, animation
- `SharedArrayBuffer` are an important part of the evolution of JavaScript from a single threaded language to multithreading

Browser vendors want more efficient, less penalizing countermeasures.

Isolation-based countermeasures

Site isolation



Charles Reis, Alexander Moshchuk, and Nasko Oskov. "Site Isolation: Process Separation for Web Sites within the Browser". In: USENIX Security Symposium. 2019

COOP/COEP

Set of HTTP headers between a top level domain and all loaded resources.

If every resource agrees on a shared policy, the group becomes its own process.

COOP/COEP

Set of HTTP headers between a top level domain and all loaded resources.

If every resource agrees on a shared policy, the group becomes its own process.

Not activated by default, must be managed by the website.

If an attacker controls their website, they can activate/deactivate it at will.

Goals of isolation

Different processes means:

- Different address spaces

Goals of isolation

Different processes means:

- Different address spaces $\rightarrow$ Prevents Spectre v1 and other attacks that target the same address space

Goals of isolation

Different processes means:

- Different address spaces $\rightarrow$ Prevents Spectre v1 and other attacks that target the same address space

What site isolation does not prevent:

Goals of isolation

Different processes means:

- Different address spaces $\rightarrow$ Prevents Spectre v1 and other attacks that target the same address space

What site isolation does not prevent:

- Hardware contention timing attacks.

Goals of isolation

Different processes means:

- Different address spaces $\rightarrow$ Prevents Spectre v1 and other attacks that target the same address space

What site isolation does not prevent:

- Hardware contention timing attacks.
- Cross address space (transient execution) attacks [2].

.

[2] For instance https://leaky.page/ was published a few days after our paper

A change in paradigm

With the introduction of site isolation and COOP/COEP, browser vendors considered the main security issue fixed.

A change in paradigm

With the introduction of site isolation and COOP/COEP, browser vendors considered the main security issue fixed.

Firefox 79 reallowed `SharedArrayBuffer` and set the resolution of `performance.now()` to 20 µs with COOP/COEP.

Chrome 76 reallowed `SharedArrayBuffer` with COOP/COEP and set the resolution of `performance.now()` to 5 µs with jitter in all cases

Impact of these changes?

- Timing-based countermeasures are efficient against most timing attacks.

Impact of these changes?

- Timing-based countermeasures are efficient against most timing attacks.
- New, isolation-based countermeasures are strong countermeasures, but focused on Spectre or software-based timing attacks.

Impact of these changes?

- Timing-based countermeasures are efficient against most timing attacks.
- New, isolation-based countermeasures are strong countermeasures, but focused on Spectre or software-based timing attacks.
- Hardware-based timing attacks as well as other transient execution attacks are only mitigated by timing-based countermeasures.

Impact of these changes?

- Timing-based countermeasures are efficient against most timing attacks.
- New, isolation-based countermeasures are strong countermeasures, but focused on Spectre or software-based timing attacks.
- Hardware-based timing attacks as well as other transient execution attacks are only mitigated by timing-based countermeasures.
- Recent changes in timers have not been motivated or evaluated.

Impact of these changes?

- Timing-based countermeasures are efficient against most timing attacks.
- New, isolation-based countermeasures are strong countermeasures, but focused on Spectre or software-based timing attacks.
- Hardware-based timing attacks as well as other transient execution attacks are only mitigated by timing-based countermeasures.
- Recent changes in timers have not been motivated or evaluated.

What are the security implications of reintroducing high resolution timers?

What are the security implications of reintroducing high resolution timers?

Automated framework to evaluate JavaScript timers using Selenium.

What are the security implications of reintroducing high resolution timers?

Automated framework to evaluate JavaScript timers using Selenium.

Works on Chrome and Firefox, including past and future versions.

What are the security implications of reintroducing high resolution timers?

Automated framework to evaluate JavaScript timers using Selenium.

Works on Chrome and Firefox, including past and future versions.

Our goal is that this analysis can be helpful not only at this point in time, but also in the future.

The code is available here: https://github.com/thomasrokicki/in-search-of-lost-time

How to evaluate the efficiency of a timer

Resolution: Smallest operation a timer can measure.

How to evaluate the efficiency of a timer

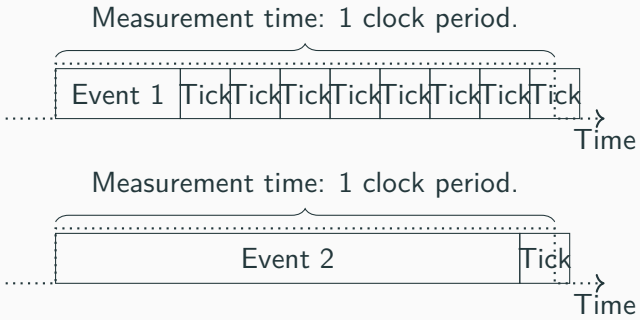
Resolution: Smallest operation a timer can measure.

Measurement overhead: Time it takes to make the measurement.

How to evaluate the efficiency of a timer

Resolution: Smallest operation a timer can measure.

Measurement overhead: Time it takes to make the measurement.



Performance.now interpolation

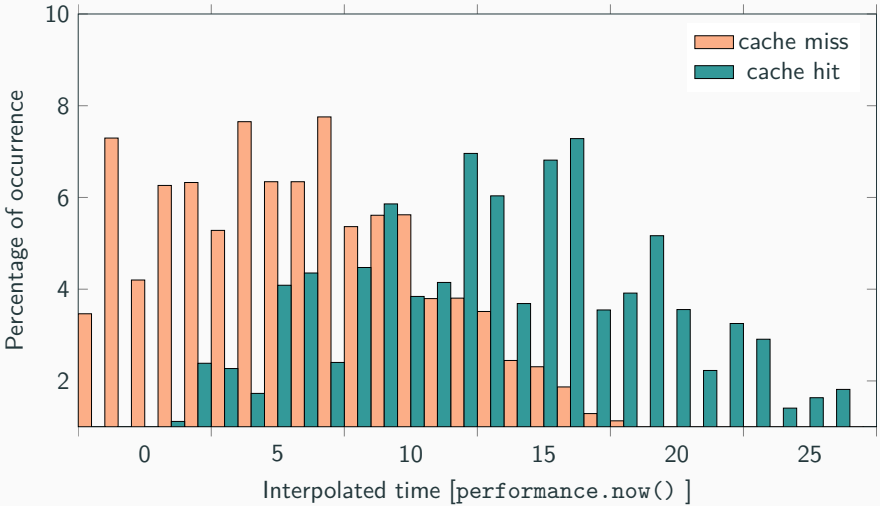
Measurement overhead $\sim=$ the resolution of `performance.now()`

Resolution is hard to evaluate because of the jitter.

Distinguishing hits and misses on Chrome 84

Goal: Differentiate cache hits from cache misses

Distinguishing hits and misses on Chrome 84



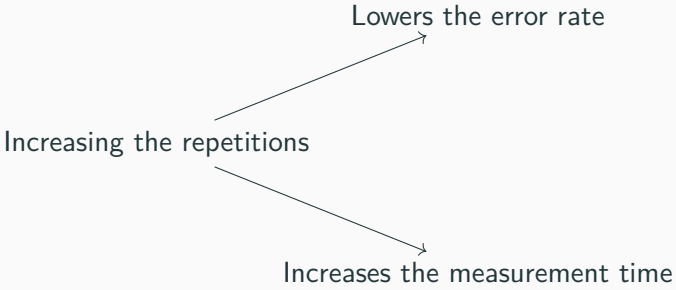
Amplification

Repeat the measurement to reduce the randomness

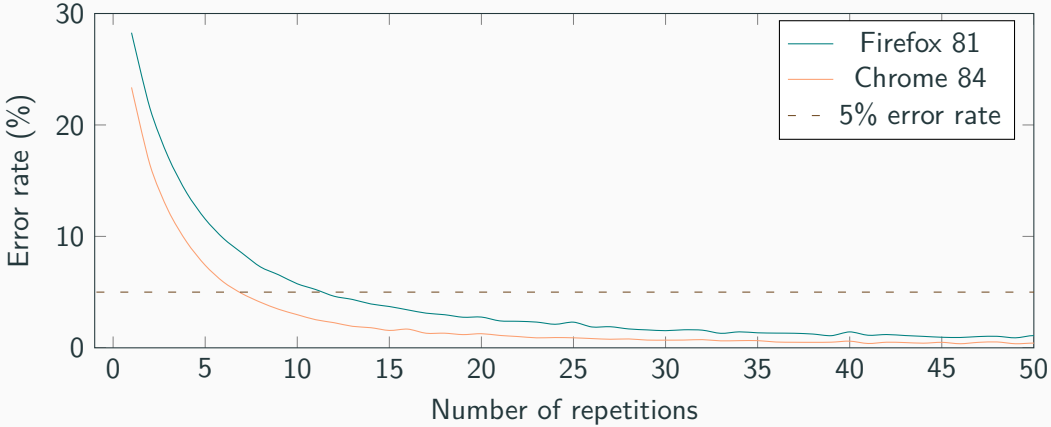
Amplification

Repeat the measurement to reduce the randomness

Repeat the measurement to reduce the randomness



Time / Precision compromise



Evaluation at 5% error rate

Browser	base resolution	Number of repetitions	Measurement overhead
Firefox 88 without COOP/COEP	1 ms with jitter	15	18 ms
Firefox 88 with COOP/COEP	20 µs without jitter	2	45 µs
Chrome 90	5 µs with jitter	8	44 µs

Resolution: Time of an incrementation in the `SharedArrayBuffer` $\rightarrow$ 10 ns

Measurement overhead: Twice the time of a read $\rightarrow$ 20 ns

Concrete example: Ideal bit rate

Browser	Ideal bit rate [bit/s]
Firefox 88 without COOP/COEP	60
Firefox 88 with COOP/COEP	22×10^4
Chrome 90	22×10^4
`SharedArrayBuffer`	50×10^6

Concrete example: building an eviction set

Prerequisites for most cache attacks (hence transient execution attacks).

Requires **O(|cache lines|)** time measurements.

Browser	Practical computation time
Firefox 88 without COOP/COEP	$\sim$ 10 min
Firefox 88 with COOP/COEP	$\sim$ 50 s
Chrome 90	$\sim$ 50 s
`SharedArrayBuffer`	$\sim$ 1 s

Some perspective

In an environment with COOP/COEP, on Firefox 88 an attacker can:

Some perspective

In an environment with COOP/COEP, on Firefox 88 an attacker can:

- Create a covert channel with an ideal bandwidth 800,000 times superior compared to Firefox 78

Some perspective

In an environment with COOP/COEP, on Firefox 88 an attacker can:

- Create a covert channel with an ideal bandwidth 800,000 times superior compared to Firefox 78
- Compute an eviction set in a matter of seconds, whereas it required tens of minutes on Firefox 78

Some perspective

In an environment with COOP/COEP, on Firefox 88 an attacker can:

- Create a covert channel with an ideal bandwidth 800,000 times superior compared to Firefox 78
- Compute an eviction set in a matter of seconds, whereas it required tens of minutes on Firefox 78

Timers are more of a threat than two years ago.

Conclusion

- Powerful and fast timers with a 10-100 ns resolution exist.

Conclusion

- Powerful and fast timers with a 10-100 ns resolution exist.
- Site isolation and COOP/COEP only apply to Spectre v1 and some system resource attacks.

Conclusion

- Powerful and fast timers with a 10-100 ns resolution exist.
- Site isolation and COOP/COEP only apply to Spectre v1 and some system resource attacks.
- Browsers are potentially vulnerable to many hardware or transient execution attacks.

Conclusion

- Powerful and fast timers with a 10-100 ns resolution exist.
- Site isolation and COOP/COEP only apply to Spectre v1 and some system resource attacks.
- Browsers are potentially vulnerable to many hardware or transient execution attacks.
- More viable countermeasures must be found, but it is not suited for browsers.

Thank you for your attention

Contact me here: thomas.rokicki@irisa.fr

Feel free to read the paper for more technical details!

Find the code here:

https://github.com/thomasrokicki/in-search-of-lost-time