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-based timing attacks

JavaScript Timing Attacks
JavaScript-based timing attacks

JavaScript **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

What are the security implications of changing the timers' resolution?
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()
performance.now(): Resolution ranges from 5μs to 1ms.
performance.now(): Resolution ranges from 5 µs to 1 ms. We need to time events in the order of 10 ns.
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”
Schwarz et al., “Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript”
Schwarz et al., “Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript”
Schwarz et al., “Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript”
Event one lasts 4 ticks
Reducing the resolution alone is not sufficient because of interpolation. ¹

¹This 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 \textit{jitter} to the measurement.

\(^1\)This applies to other all timing-based functions such as callbacks, animation functions and others.
Reducing the resolution alone is not sufficient because of interpolation. \(^1\)

Add \textit{jitter} to the measurement.

\(^1\)This applies to other all timing-based functions such as callbacks, animation functions and others.
Interpolation and jitter

Interpolated time: 9 ticks

Event 1 | Tick | Tick | Tick | Tick | Tick | Tick | Tick | Tick | Tick

Interpolated time: 5 ticks

Event 1 | Tick | Tick | Tick | Tick | Tick

Firefox: 1 ms with jitter.
Chrome: 100 µs with jitter.
Interpolation and jitter

Interpolated time: 9 ticks

Interpolated time: 5 ticks

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*
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.
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
Different processes means:

- Different address spaces → 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 → 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 → 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
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.
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.

Measurement time: 1 clock period.

Event 1
Tick
Tick
Tick
Tick
Tick
Tick
Tick

Measurement time: 1 clock period.

Event 2
Tick
Measurement overhead $\sim=$ the resolution of `performance.now()`
Resolution is hard to evaluate because of the jitter.
Goal: Differentiate cache hits from cache misses
Goal: Differentiate cache hits from cache misses

Interpolated time [performance.now()]

Percentage of occurrence

- Cache miss
- Cache hit
Amplification

Repeat the measurement to reduce the randomness
Amplification

Repeat the measurement to reduce the randomness.

Increasing the repetitions lowers the error rate.
Repeat the measurement to reduce the randomness

- Lowers the error rate
- Increasing the repetitions
  - Increases the measurement time
Time / Precision compromise

![Graph showing error rate (%) vs. number of repetitions for Firefox 81 and Chrome 84. Orange line represents Firefox 81, teal line represents Chrome 84. Dashed line indicates a 5% error rate.](image_url)
## Evaluation at 5% error rate

<table>
<thead>
<tr>
<th>Browser</th>
<th>base resolution</th>
<th>Number of repetitions</th>
<th>Measurement overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox 88 without COOP/COEP</td>
<td>1 ms with jitter</td>
<td>15</td>
<td>18 ms</td>
</tr>
<tr>
<td>Firefox 88 with COOP/COEP</td>
<td>20 µs without jitter</td>
<td>2</td>
<td>45 µs</td>
</tr>
<tr>
<td>Chrome 90</td>
<td>5 µs with jitter</td>
<td>8</td>
<td>44 µs</td>
</tr>
</tbody>
</table>
Resolution: Time of an incrementation in the SharedArrayBuffer \(\rightarrow 10 \text{ ns}\)

Measurement overhead: Twice the time of a read \(\rightarrow 20 \text{ ns}\)
Concrete example: Ideal bit rate

<table>
<thead>
<tr>
<th>Browser</th>
<th>Ideal bit rate [bit/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox 88 without COOP/COEP</td>
<td>60</td>
</tr>
<tr>
<td>Firefox 88 with COOP/COEP</td>
<td>$22 \times 10^4$</td>
</tr>
<tr>
<td>Chrome 90</td>
<td>$22 \times 10^4$</td>
</tr>
<tr>
<td>SharedArrayBuffer</td>
<td>$50 \times 10^6$</td>
</tr>
</tbody>
</table>
Prerequisites for most cache attacks (hence transient execution attacks).

Requires $O(|\text{cache lines}|)$ time measurements.

<table>
<thead>
<tr>
<th>Browser</th>
<th>Practical computation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox 88 without COOP/COEP</td>
<td>$\sim 10 \text{ min}$</td>
</tr>
<tr>
<td>Firefox 88 with COOP/COEP</td>
<td>$\sim 50 \text{ s}$</td>
</tr>
<tr>
<td>Chrome 90</td>
<td>$\sim 50 \text{ s}$</td>
</tr>
<tr>
<td>SharedArrayBuffer</td>
<td>$\sim 1 \text{ s}$</td>
</tr>
</tbody>
</table>
In an environment with COOP/COEP, on Firefox 88 an attacker can:
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.

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.
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