Microarchitectural Attacks and Defenses in JavaScript

Michael Schwarz, Daniel Gruss, Moritz Lipp
25.01.2018

www.iaik.tugraz.at
Microarchitecture...

- is not defined on the architectural state
Microarchitecture...

- is not defined on the architectural state
- should not be visible to software
Microarchitecture...

- is not defined on the architectural state
- should not be visible to software
- is hardware specific and not fully documented
Microarchitecture...

- is not defined on the architectural state
- should not be visible to software
- is hardware specific and not fully documented
- changes to some extend with new processor generations
Microarchitectural states can be used for attacks

- Cache state ⇒ data access
Microarchitectural states can be used for attacks

- Cache state $\Rightarrow$ data access
- DRAM buffers $\Rightarrow$ data access
Microarchitectural states can be used for attacks

- Cache state $\Rightarrow$ data access
- DRAM buffers $\Rightarrow$ data access
- Interrupts $\Rightarrow$ keystrokes
Microarchitectural states can be used for attacks

- Cache state ⇒ data access
- DRAM buffers ⇒ data access
- Interrupts ⇒ keystrokes
- Branch predictors ⇒ program flow
Microarchitectural states can be used for attacks

- Cache state $\Rightarrow$ data access
- DRAM buffers $\Rightarrow$ data access
- Interrupts $\Rightarrow$ keystrokes
- Branch predictors $\Rightarrow$ program flow
- Timings $\Rightarrow$ data values
Side-channel attacks exploit side effects of operations

- Microarchitectural attacks are usually side-channel attacks
Side-channel attacks exploit side effects of operations

- Microarchitectural attacks are usually side-channel attacks
- Sensors $\Rightarrow$ user activity
Side-channel attacks exploit side effects of operations

- Microarchitectural attacks are usually side-channel attacks
- Sensors $\Rightarrow$ user activity
- Timings $\Rightarrow$ data values, activity
A core component of many such attacks: **Timers**
One Important Component

- A core component of many such attacks: **Timers**
- Side-channel attacks often require high-resolution timers
• A core component of many such attacks: Timers
• Side-channel attacks often require high-resolution timers
• Differences to measure are often in the range of nanoseconds or microseconds
• A core component of many such attacks: **Timers**
• Side-channel attacks often require high-resolution timers
• Differences to measure are often in the range of nanoseconds or microseconds
• Microarchitectural attacks usually require highest precision
Attacks in JavaScript
First side-channel attack in JavaScript

- Stone et al. (2013): Pixel perfect timing attacks with HTML5
First side-channel attack in JavaScript

- Stone et al. (2013): Pixel perfect timing attacks with HTML5
- Timing of various redraw events (e.g., visited state of links)
First side-channel attack in JavaScript

- Stone et al. (2013): Pixel perfect timing attacks with HTML5
- Timing of various redraw events (e.g., visited state of links)
- SVG filter timing to extract individual pixels (already 2011)
First side-channel attack in JavaScript

- Stone et al. (2013): Pixel perfect timing attacks with HTML5
- Timing of various redraw events (e.g., visited state of links)
- SVG filter timing to extract individual pixels (already 2011)
- High-resolution timer was available in browser
First microarchitectural attack in JavaScript

- Oren et al. (2015): The Spy in the Sandbox
First microarchitectural attack in JavaScript

- Oren et al. (2015): The Spy in the Sandbox
- Timing of memory accesses
First microarchitectural attack in JavaScript

- Oren et al. (2015): The Spy in the Sandbox
- Timing of memory accesses
- Allows to determine whether data is cached or uncached
First microarchitectural attack in JavaScript

- Oren et al. (2015): The Spy in the Sandbox
- Timing of memory accesses
- Allows to determine whether data is cached or uncached
- Possibility to infer info about other programs from browser
FANTASTIC TIMERS
AND WHERE TO FIND THEM
HIGH-RESOLUTION MICROARCHITECTURAL ATTACKS IN JAVASCRIPT
• We need a high-resolution timer to measure such small differences
• We need a high-resolution timer to measure such small differences
• Native: `rdtsc` - timestamp in CPU cycles
Timers in JavaScript

- We need a high-resolution timer to measure such small differences
- Native: `rdtsc` - timestamp in CPU cycles
- JavaScript: `performance.now()` has the highest resolution
• We need a high-resolution timer to measure such small differences
• 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
...up to microsecond precision?

| Firefox $\leq 36$ | $1 \cdot 10^{-3}$ |
...up to microsecond precision?

Edge 38
Firefox \leq 36

1

\begin{array}{c|c}
\hline
0 & 1 \\
0 & 1 \cdot 10^{-3} \\
\hline
\end{array}
...up to microsecond precision?

- W3C standard
- Edge 38
- Firefox ≤ 36

- 1
- $1 \cdot 10^{-3}$
...up to microsecond precision?

Firefox ≥ 37/Chrome/Safari: 5
W3C standard: 5
Edge 38: 1
Firefox ≤ 36: $1 \cdot 10^{-3}$
...up to microsecond precision?

<table>
<thead>
<tr>
<th>Browser</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tor</td>
<td>0</td>
</tr>
<tr>
<td>Firefox ≥ 37/Chrome/Safari</td>
<td>5</td>
</tr>
<tr>
<td>W3C standard</td>
<td>5</td>
</tr>
<tr>
<td>Edge 38</td>
<td>1</td>
</tr>
<tr>
<td>Firefox ≤ 36</td>
<td>$1 \cdot 10^{-3}$</td>
</tr>
<tr>
<td></td>
<td>$1 \cdot 10^5$</td>
</tr>
</tbody>
</table>
...up to microsecond precision?

<table>
<thead>
<tr>
<th>Browser/Version</th>
<th>Fuzzyfox</th>
<th>Firefox ≥ 37/Chrome/Safari</th>
<th>W3C standard</th>
<th>Edge 38</th>
<th>Firefox ≤ 36</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 · 10^5</td>
<td>1 · 10^5</td>
<td>5</td>
<td>1</td>
<td>1 · 10^{-3}</td>
</tr>
</tbody>
</table>
New timer
We require a higher resolution

- Current precision is not sufficient to measure cycle differences
We require a higher resolution

- Current precision is not sufficient to measure cycle differences
- We have two possibilities
We require a higher resolution

- Current precision is not sufficient to measure cycle differences
- We have two possibilities
- Recover a higher resolution from the available timer
We require a higher resolution

- Current precision is not sufficient to measure cycle differences
- We have two possibilities
  - Recover a higher resolution from the available timer
  - Build our own high-resolution timer
• Measure how often we can increment a variable between two timer ticks
• **Measure** how often we can **increment** a variable between two timer ticks

• Average number of increments is the **interpolation step**
Recovering resolution - Clock interpolation

• Measure how often we can increment a variable between two timer ticks
• Average number of increments is the interpolation step
• To measure with high resolution:
Recovering resolution - Clock interpolation

- **Measure** how often we can **increment** a variable between two timer ticks
- Average number of increments is the **interpolation step**
- To measure with high resolution:
  - Start measurement at **clock edge**
• Measure how often we can increment a variable between two timer ticks

• Average number of increments is the interpolation step

• To measure with high resolution:
  • Start measurement at clock edge
  • Increment a variable until next clock edge
• **Measure** how often we can **increment** a variable between two timer ticks
• **Average number of increments** is the **interpolation step**
• To measure with high resolution:
  • Start measurement at **clock edge**
  • **Increment** a variable until next clock edge
• **Highly accurate:** 500 ns (Firefox/Chrome), 15 µs (Tor)
• We can get a higher resolution for a classifier only
• We can get a higher resolution for a classifier only
• Often sufficient to see which of two functions takes longer
• We can get a higher resolution for a classifier only
• Often sufficient to see which of two functions takes longer
• We can get a higher resolution for a classifier only
• Often sufficient to see which of two functions takes longer

Edge thresholding: apply padding such that the slow function crosses one more clock edge than the fast function.
Recovering resolution - Edge thresholding

Yields nanosecond resolution

Firefox/Tor (2 ns), Edge (10 ns), Chrome (15 ns)
Yields nanosecond resolution
• Yields **nanosecond** resolution
• Firefox/Tor (2 ns), Edge (10 ns), Chrome (15 ns)
Goal: counter that does not block main thread
• **Goal**: counter that does not block main thread

• **Baseline setTimeout**: 4 ms (except Edge: 2 ms)
Goal: counter that does not block main thread

Baseline `setTimeout`: 4 ms (except Edge: 2 ms)

CSS animation: increase width of element as fast as possible
• **Goal**: counter that does not block main thread
• Baseline `setTimeout`: 4 ms (except Edge: 2 ms)
• **CSS animation**: increase width of element as fast as possible
• Width of element is timestamp
Building a timer

- **Goal**: counter that does not block main thread
- Baseline `setTimeout`: 4 ms (except Edge: 2 ms)
- **CSS animation**: increase width of element as fast as possible
- Width of element is timestamp
- However, animation is limited to 60 fps $\rightarrow$ 16 ms
Building a timer - Web worker

- JavaScript can spawn new threads called web worker
• JavaScript can spawn **new threads** called web worker
• Web worker communicate using **message passing**
Building a timer - Web worker

- JavaScript can spawn new threads called web worker
- Web worker communicate using message passing
- Let worker count and request timestamp in main thread
• JavaScript can spawn new threads called web worker
• Web worker communicate using message passing
• Let worker count and request timestamp in main thread
• Multiple possibilities: postMessage, MessageChannel or BroadcastChannel
• JavaScript can spawn **new threads** called web worker
• Web worker communicate using **message passing**
• Let **worker count** and request timestamp in main thread
• Multiple **possibilities**: `postMessage`, `MessageChannel` or `BroadcastChannel`
• Yields **microsecond** resolution (even on Tor and Fuzzyfox)
- **Experimental** feature to share data: `SharedArrayBuffer`
Building a timer - Web worker

- Experimental feature to share data: SharedArrayBuffer
- Web worker can simultaneously read/write data
• Experimental feature to share data: SharedArrayBuffer
• Web worker can simultaneously read/write data
• No message passing overhead
Building a timer - Web worker

- Experimental 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

• Experimental 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
Building a timer - Web worker

- **Experimental** 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
- Sufficient for *microarchitectural* attacks
Timer evaluation

Access time [SharedArrayBuffer increments]

Number of cases

- cache hit
- cache miss

Michael Schwarz, Daniel Gruss, Moritz Lipp — www.iaik.tugraz.at
Attack Requirements
• Timers were always the main focus
• Timers were always the main focus
• Reducing timer resolution is not sufficient
Attack requirements

- Timers were always the main focus
- Reducing timer resolution is not sufficient
- Timers can (always) be built
Attack requirements

- Timers were always the main focus
- Reducing timer resolution is not sufficient
- Timers can (always) be built
- Some attacks do not require timers at all
• Timers were always the main focus
• Reducing timer resolution is not sufficient
• Timers can (always) be built
• Some attacks do not require timers at all
• Important to understand requirements before designing countermeasures
JavaScript
zero
REAL
JavaScript
AND ZERO
SIDE-CHANNEL
ATTACKS
• Currently 11 microarchitectural and side-channel attacks in JavaScript
Identify requirements

- Currently 11 microarchitectural and side-channel attacks in JavaScript
- Analyse requirements for every attack
Identify requirements

- Currently 11 microarchitectural and side-channel attacks in JavaScript
- Analyse requirements for every attack
- Results in 5 categories
Identify requirements

• Currently 11 microarchitectural and side-channel attacks in JavaScript
• Analyse requirements for every attack
• Results in 5 categories
  • Memory addresses
  • Accurate timing
  • Multithreading
  • Shared data
  • Sensor API
Identify requirements

- Currently **11** microarchitectural and side-channel attacks in JavaScript
- Analyse requirements for every attack
- Results in **5 categories**
  - Memory addresses
  - Accurate timing
  - Multithreading
  - Shared data
  - Sensor API
- Every attack is in **at least one category**
## Attacks and Categories

<table>
<thead>
<tr>
<th>Practical Memory Deduplication Attacks in Sandboxed Javascript</th>
<th>Memory addresses</th>
<th>Accurate timing</th>
<th>Multithreading</th>
<th>Shared data</th>
<th>Sensor API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rowhammer.js</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Fantastic Timers and Where to Find Them</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>ASLR on the Line</td>
<td>●</td>
<td>● †</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>The spy in the sandbox</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Loophole</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Pixel perfect timing attacks with HTML5</td>
<td>○</td>
<td>● †</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>The clock is still ticking</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Practical Keystroke Timing Attacks in Sandboxed JavaScript</td>
<td>○</td>
<td>● †</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>TouchSignatures</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Stealing sensitive browser data with the W3C Ambient Light Sensor API</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
</tbody>
</table>

† If accurate timing is not available, it can be approximated using a combination of multithreading and shared data.
• Language does not provide addresses to programmer
• Language does not provide *addresses* to programmer
• Closest to virtual address: *array* indices
Memory Addresses

- Language does not provide addresses to programmer
- Closest to virtual address: array indices
- ArrayBuffer is page aligned, leaks 12 bits of address
Memory Addresses

- Language does not provide addresses to programmer
- Closest to virtual address: array indices
- `ArrayBuffer` is page aligned, leaks 12 bits of address
- If 2 MB backing pages are used, 21 bits of address known
• Language does not provide addresses to programmer
• Closest to virtual address: array indices
• ArrayBuffer is page aligned, leaks 12 bits of address
• If 2 MB backing pages are used, 21 bits of address known
• If not page aligned: detect page faults through timing
• Nearly all attacks require **accurate timing**
Accurate Timing

- Nearly all attacks require **accurate timing**
- No absolute timestamps required, only **time differences**
• Nearly all attacks require accurate timing
• No absolute timestamps required, only time differences
• Required accuracy varies between milliseconds and nanoseconds
• Nearly all attacks require **accurate timing**
• No absolute timestamps required, only **time differences**
• Required accuracy varies between milliseconds and nanoseconds
• Such timers can be **built** if not available (e.g., message passing)
• Nearly all attacks require accurate timing
• No absolute timestamps required, only time differences
• Required accuracy varies between milliseconds and nanoseconds
• Such timers can be built if not available (e.g., message passing)
• If attack is repeatable, less accurate timing can be sufficient
Multithreading

- JavaScript introduced multi threading with web workers
• JavaScript introduced *multi threading* with web workers
• Enables new side-channel attacks
Multithreading

- JavaScript introduced multi threading with web workers
- Enables new side-channel attacks
- Dispatch latency of event queue allows to infer activity of other tabs
Multithreading

- JavaScript introduced *multi threading* with web workers
- Enables new side-channel attacks
- Dispatch latency of event queue allows to infer activity of other tabs
- Endless loop in worker allows to detect hardware interrupts
• Usually no shared data between threads due to synchronization issues
• Usually no shared data between threads due to synchronization issues
• Exception: SharedArrayBuffer
Usually no shared data between threads due to synchronization issues
Exception: SharedArrayBuffer
Only useful in combination with web workers
• Usually no shared data between threads due to synchronization issues
• Exception: SharedArrayBuffer
• Only useful in combination with web workers
• Allows to build timers with extremely high resolution (up to 1 ns)
• Usually no **shared data** between threads due to synchronization issues
• **Exception:** `SharedArrayBuffer`
• Only useful in combination with web workers
• Allows to build timers with extremely high resolution (up to 1 ns)
• Not enabled by default
• Some side-channel attacks only require access to sensors
• Some side-channel attacks only require access to sensors
• Several sensors are available in JavaScript
- Some side-channel attacks only require access to sensors
- Several sensors are available in JavaScript
- Some require user consent, e.g., microphone
• Some side-channel attacks only require access to sensors
• Several sensors are available in JavaScript
• Some require user consent, e.g., microphone
• Other can be used without user consent, e.g., ambient light
• Some side-channel attacks only require access to sensors
• Several sensors are available in JavaScript
• Some require user consent, e.g., microphone
• Other can be used without user consent, e.g., ambient light
• There are attacks with these sensors
Defenses
• Countermeasures have to address all categories
• Countermeasures have to address all categories
• Should not be visible to the programmer
Countermeasures

- Countermeasures have to address all categories
- Should not be visible to the programmer
- Implementation is on the “microarchitectural” level of JavaScript
Countermeasures

- Countermeasures have to address all categories
- Should not be visible to the programmer
- Implementation is on the “microarchitectural” level of JavaScript
- If no category is usable for attacks anymore, future attacks are hard
#1: Buffer ASLR
#1: Buffer ASLR

- Ensure arrays are not page aligned
#1: Buffer ASLR

- Ensure arrays are not page aligned
- Attacker cannot assume that least significant 12 bits are ‘0’
#1: Buffer ASLR

- Ensure arrays are **not page aligned**
- Attacker cannot assume that least significant 12 bits are ‘0’
- Only works for the first page
#1: Buffer ASLR

- Ensure arrays are not page aligned
- Attacker cannot assume that least significant 12 bits are ‘0’
- Only works for the first page
- Consecutive page borders can be detected through page faults
#2: Preloading

- Instead of lazy initialization for arrays, ensure that they are always memory backed.
#2: Preloading

• Instead of lazy initialization for arrays, ensure that they are always memory backed
• Attacker cannot detect page borders through page faults anymore
#2: Preloading

- Instead of lazy initialization for arrays, ensure that they are **always** memory backed
- Attacker cannot detect page borders through page faults anymore
- Does not work if swapping or page deduplication is enabled
#2: Preloading

- Instead of lazy initialization for arrays, ensure that they are always memory backed
- Attacker cannot detect page borders through page faults anymore
- Does not work if swapping or page deduplication is enabled
- Has to be combined with Buffer ASLR
#3: Non-determinism

- For every array access, add another random access
#3: Non-determinism

- For every array access, add another random access
- Makes page border detection infeasible without requiring significantly more memory
#3: Non-determinism

- For every array access, add another random access
- Makes page border detection infeasible without requiring significantly more memory
- Attacker always times two accesses
#3: Non-determinism

- For every array access, add another random access
- Makes page border detection infeasible without requiring significantly more memory
- Attacker always times two accesses
- Distinguishing cached from non-cached addresses is hard
#4: Array Index Randomization

- Ensures arrays are not linear

Use a random linear function to map array index to underlying buffer. Index $x$ maps to $f(x) = ax + b \mod n$, where $n$ is array length and $a$ and $b$ are randomly chosen. Has to be combined with Buffer ASLR and either Preloading or Non-determinism.
#4: Array Index Randomization

- Ensures arrays are not linear
- Use a random linear function to map array index to underlying buffer
#4: Array Index Randomization

- Ensures arrays are not linear
- Use a random linear function to map array index to underlying buffer
- Index $x$ maps to $f(x) = ax + b \mod n$, where $n$ is array length and $a$ and $b$ are randomly chosen
#4: Array Index Randomization

- Ensures arrays are **not** linear
- Use a random linear function to map array index to underlying buffer
- Index \( x \) maps to \( f(x) = ax + b \mod n \), where \( n \) is array length and \( a \) and \( b \) are randomly chosen
- Has to be combined with Buffer ASLR and either Preloading or Non-determinism
The four defenses prevent attackers from getting virtual and physical addresses.
• The four defenses prevent attackers from getting virtual and physical addresses
• Prevents many microarchitectural attacks
• The four defenses prevent attackers from getting virtual and physical addresses
• Prevents many microarchitectural attacks
• Have to be combined for maximum security
• The four defenses prevent attackers from getting virtual and physical addresses
• Prevents many microarchitectural attacks
• Have to be combined for maximum security
• Side effect: make exploits harder where addresses are required
- Reducing the resolution of `performance.now()` is a first step.
• Reducing the resolution of `performance.now()` is a first step
• Only rounding the timestamps is not sufficient
- Reducing the resolution of `performance.now()` is a first step
- Only rounding the timestamps is not sufficient
- Fuzzy time (Vattikonda et al.) adds random jitter
• Reducing the resolution of `performance.now()` is a first step
• Only rounding the timestamps is not sufficient
• **Fuzzy time** (Vattikonda et al.) adds random jitter
• Timestamps are still monotonic, but clock edges are randomized
• Only real solution is to prevent multithreading
- Only real solution is to prevent multithreading
- We used a polyfill to not completely break websites
• Only real solution is to prevent multithreading
• We used a polyfill to not completely break websites
• Some attacks can be prevented by adding random delays to `postMessage`
• Only real solution is to prevent multithreading
• We used a polyfill to not completely break websites
• Some attacks can be prevented by adding random delays to `postMessage`
• Prevents certain timing primitives and attacks on the event-queue latency
• Best countermeasures: do not allow shared data
• Best countermeasures: do not allow shared data
• Many attacks are impossible without SharedArrayBuffer
• Best countermeasures: do not allow shared data
• Many attacks are impossible without `SharedArrayBuffer`
• Alternative: delay access to buffer
• Best countermeasures: do **not** allow shared data
• Many attacks are impossible without `SharedArrayBuffer`
• Alternative: delay access to buffer
• Still faster than message passing
Shared Data

- Best countermeasures: do not allow shared data
- Many attacks are impossible without `SharedArrayBuffer`
- Alternative: delay access to buffer
- Still faster than message passing
- Degrades resolution of timing primitive to microseconds
• Reduce resolution and update frequency of sensors
• Reduce resolution and update frequency of sensors
• Sensor APIs should always ask user for permission
Sensor API

- Reduce resolution and update frequency of sensors
- Sensor APIs should always ask user for permission
- Every sensor is usable for attacks, even ambient light sensor
• Reduce **resolution and update frequency** of sensors
• Sensor APIs should always ask user for **permission**
• Every sensor is usable for attacks, even ambient light sensor
• To not break existing applications, sensors return constant value
Implementation
• Best solution is to implement defenses in the browser core
- Best solution is to implement defenses in the browser core
- Maintaining a browser fork is hard work
Designing the Countermeasure

- Best solution is to implement defenses in the browser core
- Maintaining a browser fork is hard work
- We want a generic solution for multiple browsers
Designing the Countermeasure

- Best solution is to implement defenses in the browser core
- Maintaining a browser fork is hard work
- We want a generic solution for multiple browsers
- Parsing JavaScript is hard
Designing the Countermeasure

- Best solution is to implement defenses in the browser core
- Maintaining a browser fork is hard work
- We want a generic solution for multiple browsers
- Parsing JavaScript is hard
- Implementation in JavaScript → Virtual machine layering
• Best solution is to implement defenses in the browser core
• Maintaining a browser fork is hard work
• We want a generic solution for multiple browsers
• Parsing JavaScript is hard
• Implementation in JavaScript → Virtual machine layering
• Proof-of-concept is implemented as browser extension
Some defenses might impair user experience, e.g., disable multithreading.
• Some defenses might impair user experience, e.g., disable multithreading
• The user can choose one of several pre-defined protection levels
• Some defenses might impair user experience, e.g., disable multithreading
• The user can choose one of several pre-defined protection levels
• Protection levels apply different combinations of defenses
• Some defenses might impair user experience, e.g., disable multithreading
• The user can choose one of several pre-defined protection levels
• Protection levels apply different combinations of defenses
• Each defense can either be disabled, enabled, or require user permission
Functions and properties are replaced by wrappers.
Functions can be re-defined in JavaScript

```javascript
var original_reference = window.performance.now;
window.performance.now = function() { return 0; };
```

Functions can be re-defined in JavaScript

```javascript
var original_reference = window.performance.now;
window.performance.now = function() { return 0; };

// call the new function (via function name)
alert(window.performance.now()); // == alert(0)
```
Functions can be re-defined in JavaScript

```javascript
var original_reference = window.performance.now;
window.performance.now = function() { return 0; };

// call the new function (via function name)
alert(window.performance.now()); // == alert(0)

// call the original function (only via reference)
alert(original_reference.call(window.performance));
```
• Functions can be re-defined in JavaScript

```javascript
var original_reference = window.performance.now;
window.performance.now = function() { return 0; };

// call the new function (via function name)
alert(window.performance.now()); // == alert(0)

// call the original function (only via reference)
alert(original_reference.call(window.performance));
```

• Properties can be replaced by accessor properties
• Objects are proxied

```
new Object
Proxy(Object)
```

All properties and functions are handled by the original object.
Functions and properties can be overwritten in the proxy object.
• Objects are proxied

• All properties and functions are handled by the original object
• Objects are proxied

- All properties and functions are handled by the original object
- Functions and properties can be overwritten in the proxy object
• Attacker tries to circumvent JavaScript Zero
• Attacker tries to circumvent JavaScript Zero
• Self protection is necessary if implemented in JavaScript
• Attacker tries to circumvent JavaScript Zero
• **Self protection** is necessary if implemented in JavaScript
• Use closures to hide all references to original functions

```javascript
(function() {
    // original is only accessible in this scope
    var original = window.performance.now;
    window.performance.now = ...
})();
```
• Attacker tries to circumvent JavaScript Zero
• Self protection is necessary if implemented in JavaScript
• Use closures to hide all references to original functions

```javascript
(function() {
    // original is only accessible in this scope
    var original = window.performance.now;
    window.performance.now = ...}
}());
```

• Prevent objects from being modified: `Object.freeze`
Evaluation
• Border of pages leak 12 or 21 bits (depending on page size)
• Border of pages leak 12 or 21 bits (depending on page size)
• Create huge array
- Border of pages leak 12 or 21 bits (depending on page size)
- Create huge array
- Iterate over array, measure access time
• Border of pages leak 12 or 21 bits (depending on page size)
• Create huge array
• Iterate over array, measure access time
• Page border raise pagefault, taking significantly longer to access
Access time [cycles]

Array offset [KB]

$10^5$

$10^4$
• Find addresses (= array indices) that fall into same cache set
• Find addresses (= array indices) that fall into same cache set
• Physical address defines in which cache set the data is cached
• Find addresses (= array indices) that fall into same cache set
• Physical address defines in which cache set the data is cached
• Enough addresses in one set evicts the set (Prime)
- Find addresses (= array indices) that fall into same cache set
- Physical address defines in which cache set the data is cached
- Enough addresses in one set evicts the set (**Prime**)
- Iterate again over addresses (**Probe**)

---

Michael Schwarz, Daniel Gruss, Moritz Lipp — www.iaik.tugraz.at
• Find addresses (= array indices) that fall into same cache set
• Physical address defines in which cache set the data is cached
• Enough addresses in one set evicts the set (Prime)
• Iterate again over addresses (Probe)
• If it is fast, they are still cached
- Find addresses (= array indices) that fall into same cache set
- Physical address defines in which cache set the data is cached
- Enough addresses in one set evicts the set (Prime)
- Iterate again over addresses (Probe)
- If it is fast, they are still cached
- If it is slow, someone used this cache set and evicted our addresses
Prime+Probe with Random Access

![Graph showing the distribution of cache hits and misses over access time. The x-axis represents access time in cycles, ranging from 1,500 to 2,400. The y-axis represents the number of cases, ranging from 0 to 1.5 * 10^4. The graph displays a bell-shaped curve for cache hits and a declining trend for cache misses.](image-url)
Multithreading allows to detect interrupts.
• Multithreading allows to detect *interrupts*
• Endless loop which counts number of increments in time window
• Multithreading allows to detect **interrupts**
• Endless loop which counts number of increments in time window
• Different number of increments indicate interrupt
• Multithreading allows to detect interrupts
• Endless loop which counts number of increments in time window
• Different number of increments indicate interrupt
• Fuzzy time prevents deterministic equally-sized time window
Interrupt Detection

Delta [counter]

0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Runtime [s]

1,700 1,750 1,800 1,850

tap tap tap tap
• Messages between web workers are handled in the event queue
• Messages between web workers are handled in the event queue
• User activity is also handled in the event queue
- Messages between web workers are handled in the event queue
- User activity is also handled in the event queue
- Posting many messages allows to measure latency
• Messages between web workers are handled in the event queue
• User activity is also handled in the event queue
• Posting many messages allows to measure latency
• Latency indicates user input
Event Queue Spying
Event Queue Spying with Message Delay
• **SharedArrayBuffer** allows to build a timing primitive with the highest resolution
• SharedArrayBuffer allows to build a timing primitive with the highest resolution
• One web worker continuously increments variable in the shared array
SharedArrayBuffer allows to build a timing primitive with the highest resolution.

One web worker continuously increments variable in the shared array.

Other worker uses this as a timestamp.
• `SharedArrayBuffer` allows to build a timing primitive with the highest resolution
• One web worker continuously increments variable in the shared array
• Other worker uses this as a timestamp
• Adding random delay to access degrades resolution
SharedArrayBuffer with Random Delay

![Graph showing cache hits and misses over access time.](image-url)
<table>
<thead>
<tr>
<th>Defense</th>
<th>Prevents</th>
<th>Rowhammer.js</th>
<th>Page Duplication</th>
<th>DRAM Covert Channel</th>
<th>Anti-ASLR</th>
<th>Cache Eviction</th>
<th>Keystore Timing</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer ASLR</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Array preloading</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Non-deterministic array</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Array index randomization</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Low-resolution timestamp</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Fuzzy time</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>WebWorker polyfill</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Message delay</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Slow SharedArrayBuffer</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>No SharedArrayBuffer</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Symbols indicate whether a policy fully prevents an attack, (●), partly prevents and attack by making it more difficult (○), or does not prevent an attack (●). A star (*) indicates that all policies marked with a star must be combined to prevent an attack.
User Experience

Top 25 Alexa domains
• Just rounding timers is **not sufficient**
• Just rounding timers is not sufficient
• Multithreading and shared data allow to build new timers
• Just rounding timers is not sufficient
• Multithreading and shared data allow to build new timers
• Microarchitectural attacks in the browser are possible at the moment
Conclusion

- Just rounding timers is not sufficient
- Multithreading and shared data allow to build new timers
- Microarchitectural attacks in the browser are possible at the moment
- Efficient countermeasures can be implemented in browsers
• Just rounding timers is not sufficient
• Multithreading and shared data allow to build new timers
• Microarchitectural attacks in the browser are possible at the moment
• Efficient countermeasures can be implemented in browsers
• More microarchitectural attacks in JavaScript will appear