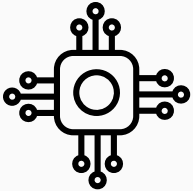
Microarchitectural Attacks and Defenses in JavaScript

Michael Schwarz, Daniel Gruss, Moritz Lipp

25.01.2018

www.iaik.tugraz.at

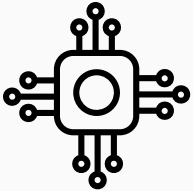
Microarchitecture



Microarchitecture...

- is not defined on the architectural state

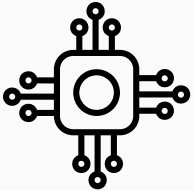
Microarchitecture



Microarchitecture...

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

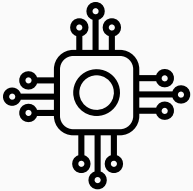
Microarchitecture



Microarchitecture...

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

Microarchitecture



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 Attacks

Microarchitectural states can be used for attacks

- Cache state $\Rightarrow$ data access

Microarchitectural Attacks

Microarchitectural states can be used for attacks

- Cache state $\Rightarrow$ data access
- DRAM buffers $\Rightarrow$ data access

Microarchitectural Attacks

Microarchitectural states can be used for attacks

- Cache state ⇒ data access
- DRAM buffers ⇒ data access
- Interrupts ⇒ keystrokes

Microarchitectural Attacks

Microarchitectural states can be used for attacks

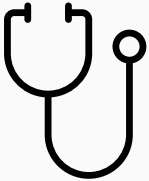
- Cache state ⇒ data access
- DRAM buffers ⇒ data access
- Interrupts ⇒ keystrokes
- Branch predictors ⇒ program flow

Microarchitectural Attacks

Microarchitectural states can be used for attacks

- Cache state ⇒ data access
- DRAM buffers ⇒ data access
- Interrupts ⇒ keystrokes
- Branch predictors ⇒ program flow
- Timings ⇒ data values

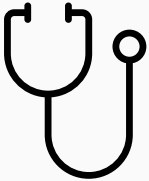
Side-Channel Attacks



Side-channel attacks exploit side effects of operations

- Microarchitectural attacks are usually side-channel attacks

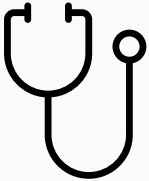
Side-Channel Attacks



Side-channel attacks exploit side effects of operations

- Microarchitectural attacks are usually side-channel attacks
- Sensors $\Rightarrow$ user activity

Side-Channel Attacks



Side-channel attacks exploit side effects of operations

- Microarchitectural attacks are usually side-channel attacks
- Sensors $\Rightarrow$ user activity
- Timings $\Rightarrow$ data values, activity

One Important Component

- 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

One Important Component

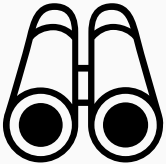
- A core component of many such attacks: Timers
- Side-channel attacks often require high-resolution timers
- Differences to measure are often in the range of nanoseconds or microseconds

One Important Component

- 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

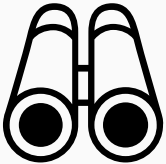
First Side-Channel Attack



First side-channel attack in JavaScript

- Stone et al. (2013): Pixel perfect timing attacks with HTML5

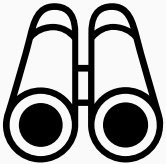
First Side-Channel Attack



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)

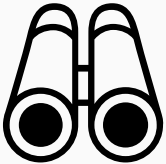
First Side-Channel Attack



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)

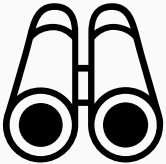
First Side-Channel Attack



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

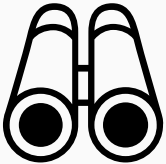
First Microarchitectural Attack



First microarchitectural attack in JavaScript

- Oren et al. (2015): The Spy in the Sandbox

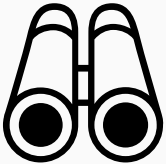
First Microarchitectural Attack



First microarchitectural attack in JavaScript

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

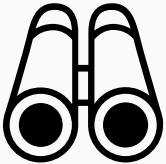
First Microarchitectural Attack



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

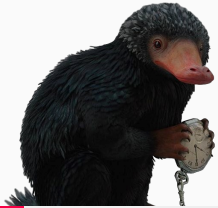


First microarchitectural attack in JavaScript

- 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



Timers in JavaScript

- We need a high-resolution timer to measure such small differences

Timers in JavaScript

- 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

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

performance.now()

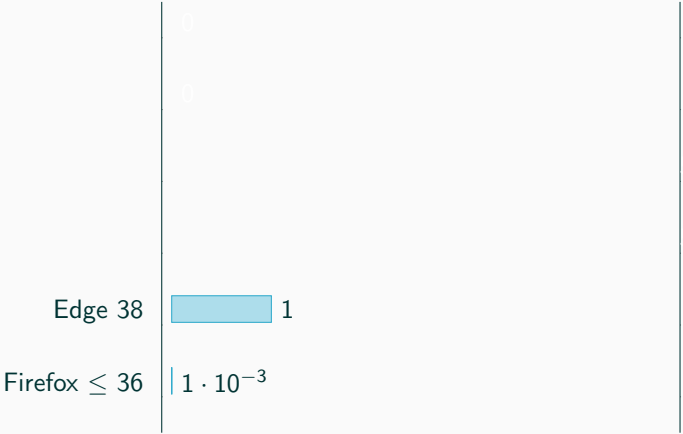
[...] represent times as floating-point numbers with up to microsecond precision.

— Mozilla Developer Network

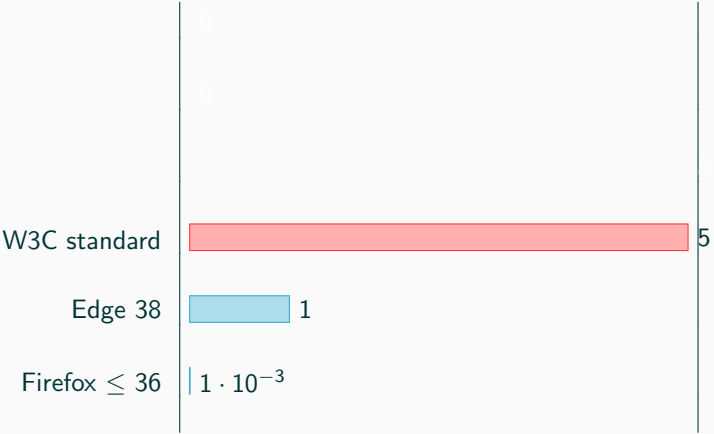
...up to microsecond precision?

Firefox ≤ 36	$1 \cdot 10^{-3}$

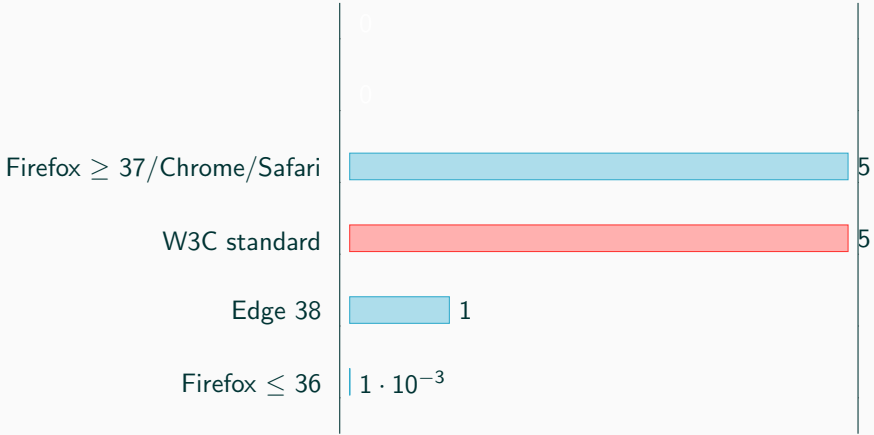
...up to microsecond precision?



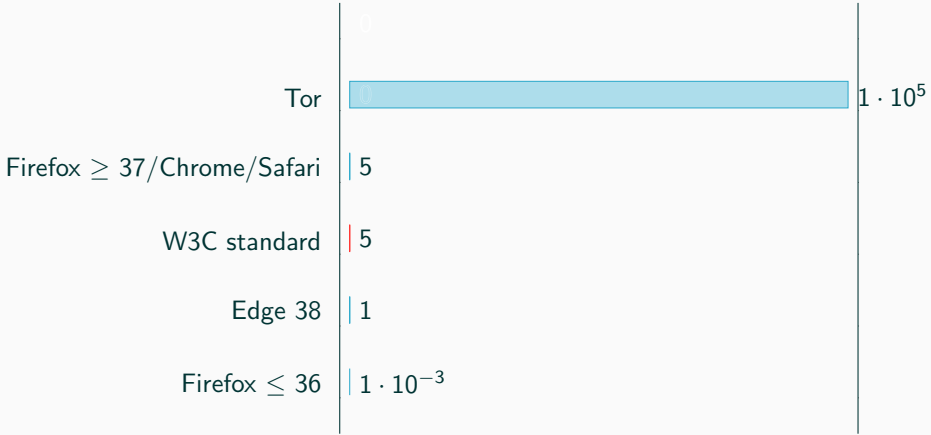
...up to microsecond precision?



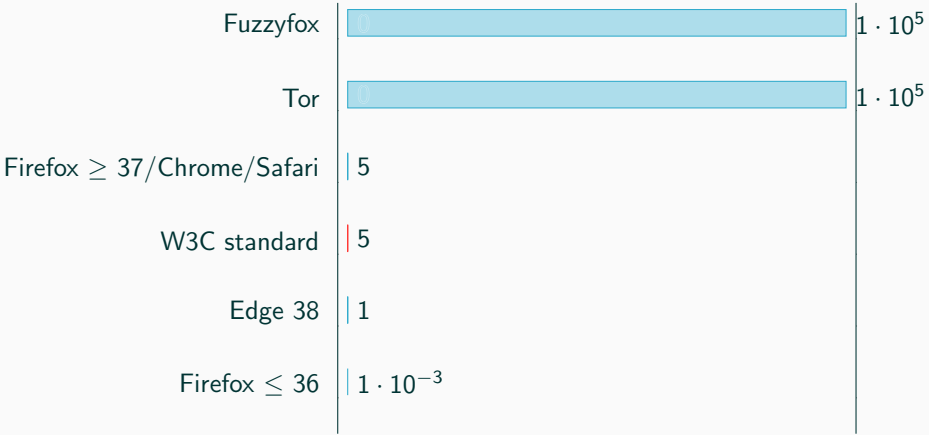
...up to microsecond precision?



...up to microsecond precision?



...up to microsecond precision?



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

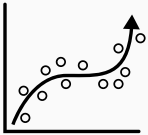
We require a higher resolution

- 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

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
- 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

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
  - Increment a variable until next clock edge

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
  - Increment a variable until next clock edge
- Highly accurate: 500 ns (Firefox/Chrome), 15 µs (Tor)

Recovering resolution - Edge thresholding

- We can get a higher resolution for a classifier only

Recovering resolution - Edge thresholding

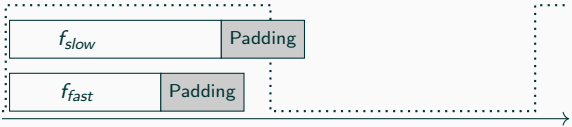
- We can get a higher resolution for a classifier only
- Often sufficient to see which of two functions takes longer

Recovering resolution - Edge thresholding

- We can get a higher resolution for a classifier only
- Often sufficient to see which of two functions takes longer

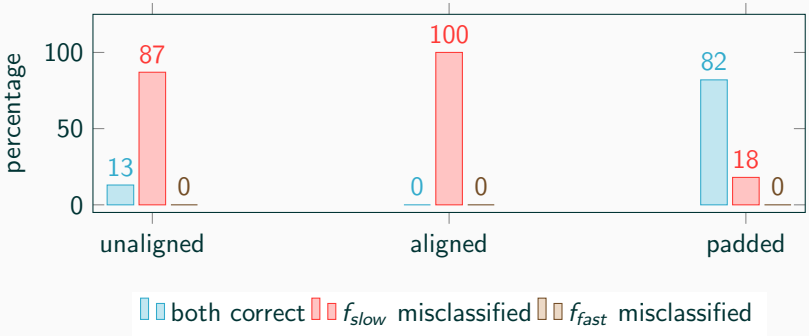
Recovering resolution - Edge thresholding

- We can get a higher resolution for a classifier only
- Often sufficient to see which of two functions takes longer

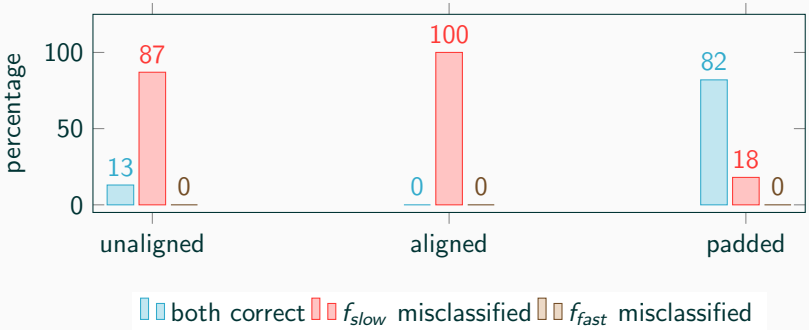


- Edge thresholding: apply padding such that the slow function crosses one more clock edge than the fast function.

Recovering resolution - Edge thresholding

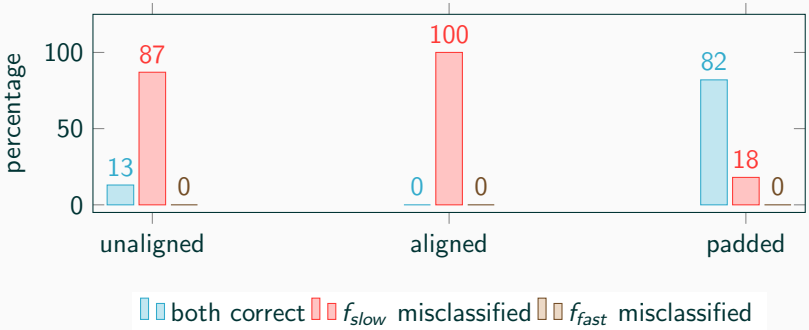


Recovering resolution - Edge thresholding



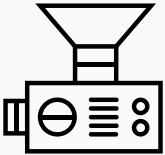
- Yields nanosecond resolution

Recovering resolution - Edge thresholding



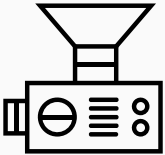
- Yields nanosecond resolution
- Firefox/Tor (2 ns), Edge (10 ns), Chrome (15 ns)

Building a timer



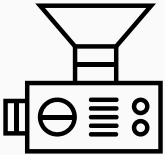
- Goal: counter that does not block main thread

Building a timer



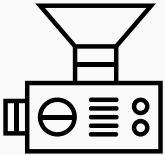
- Goal: counter that does not block main thread
- Baseline setTimeout: 4 ms (except Edge: 2 ms)

Building a timer



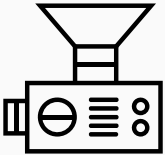
- Goal: counter that does not block main thread
- Baseline setTimeout: 4 ms (except Edge: 2 ms)
- CSS animation: increase width of element as fast as possible

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

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

Building a timer - 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

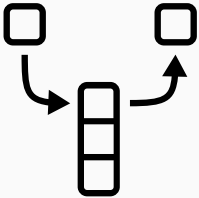
Building a timer - Web worker

- JavaScript can spawn new threads called web worker
- Web worker communicate using message passing
- Let worker count and request timestamp in main thread
- Multiple possibilities: `postMessage`, `MessageChannel` or `BroadcastChannel`

Building a timer - Web worker

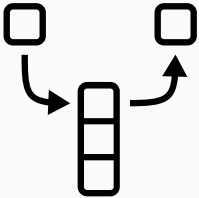
- JavaScript can spawn new threads called web worker
- Web worker communicate using message passing
- Let worker count and request timestamp in main thread
- Multiple possibilities: `postMessage`, `MessageChannel` or `BroadcastChannel`
- Yields microsecond resolution (even on Tor and Fuzzyfox)

Building a timer - Web worker



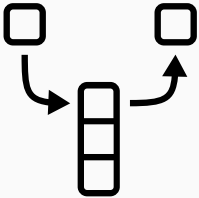
- Experimental feature to share data: `SharedArrayBuffer`

Building a timer - Web worker



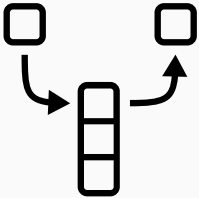
- Experimental feature to share data: `SharedArrayBuffer`
- Web worker can simultaneously read/write data

Building a timer - Web worker



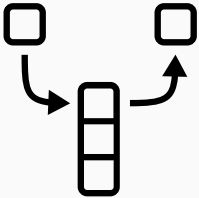
- Experimental feature to share data: `SharedArrayBuffer`
- Web worker can simultaneously read/write data
- No message passing overhead

Building a timer - Web worker



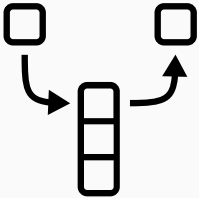
- Experimental feature to share data: `SharedArrayBuffer`
- Web worker can simultaneously read/write data
- No message passing overhead
- One dedicated worker for incrementing the shared variable

Building a timer - Web worker



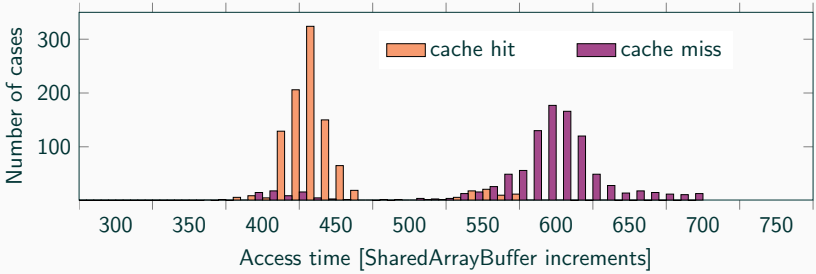
- Experimental feature to share data: `SharedArrayBuffer`
- Web worker can simultaneously read/write data
- No message passing overhead
- One dedicated worker for incrementing the shared variable
- Firefox/Fuzzyfox: 2 ns, Chrome: 15 ns

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



Attack Requirements

Attack requirements

- Timers were always the main focus

Attack requirements

- 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

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
- Important to understand requirements before designing countermeasures

JavaScript zero

REAL JavaScript AND ZERO SIDE-CHANNEL ATTACKS

Identify requirements

- 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

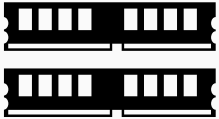
- Currently 11 microarchitectural and side-channel attacks in JavaScript
- Analyse requirements for every attack
- Results in 5 categories
  - Memory addresses
  - Accurate timing
  - Multithreading
  - Shared data
  - Sensor API
- Every attack is in at least one category

Attacks and Categories

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

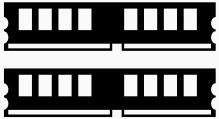
† If accurate timing is not available, it can be approximated using a combination of multithreading and shared data.

Memory Addresses



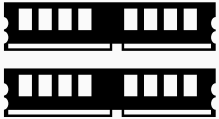
- Language does not provide addresses to programmer

Memory Addresses



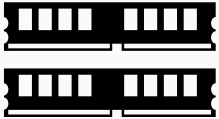
- Language does not provide addresses to programmer
- Closest to virtual address: array indices

Memory Addresses



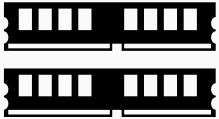
- Language does not provide addresses to programmer
- Closest to virtual address: array indices
- `ArrayBuffer` is page aligned, leaks 12 bits of address

Memory Addresses



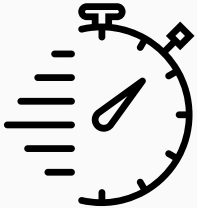
- Language does not provide addresses to programmer
- Closest to virtual address: array indices
- `ArrayBuffer` is page aligned, leaks 12 bits of address
- If 2 MB backing pages are used, 21 bits of address known

Memory Addresses



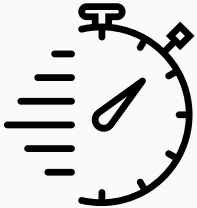
- Language does not provide addresses to programmer
- Closest to virtual address: array indices
- `ArrayBuffer` is page aligned, leaks 12 bits of address
- If 2 MB backing pages are used, 21 bits of address known
- If not page aligned: detect page faults through timing

Accurate Timing



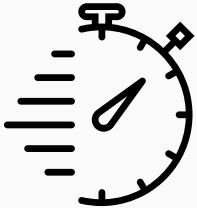
- Nearly all attacks require accurate timing

Accurate Timing



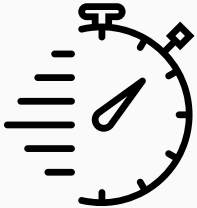
- Nearly all attacks require accurate timing
- No absolute timestamps required, only time differences

Accurate Timing



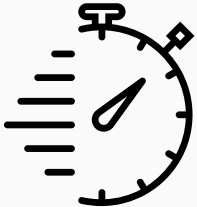
- Nearly all attacks require accurate timing
- No absolute timestamps required, only time differences
- Required accuracy varies between milliseconds and nanoseconds

Accurate Timing



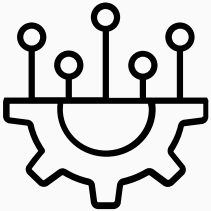
- Nearly all attacks require accurate timing
- No absolute timestamps required, only time differences
- Required accuracy varies between milliseconds and nanoseconds
- Such timers can be built if not available (e.g., message passing)

Accurate Timing



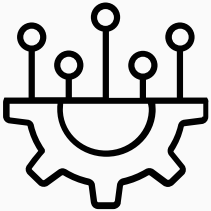
- Nearly all attacks require accurate timing
- No absolute timestamps required, only time differences
- Required accuracy varies between milliseconds and nanoseconds
- Such timers can be built if not available (e.g., message passing)
- If attack is repeatable, less accurate timing can be sufficient

Multithreading



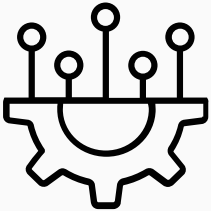
- JavaScript introduced multi threading with web workers

Multithreading



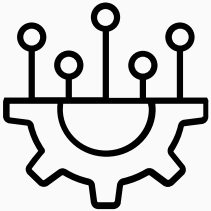
- JavaScript introduced multi threading with web workers
- Enables new side-channel attacks

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

Shared Data

- Usually no shared data between threads due to synchronization issues

Shared Data

- Usually no shared data between threads due to synchronization issues
- Exception: `SharedArrayBuffer`

Shared Data

- Usually no shared data between threads due to synchronization issues
- Exception: `SharedArrayBuffer`
- Only useful in combination with web workers

Shared Data

- 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)

Shared Data

- 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

Sensor API

- Some side-channel attacks only require access to sensors

Sensor API

- Some side-channel attacks only require access to sensors
- Several sensors are available in JavaScript

Sensor API

- Some side-channel attacks only require access to sensors
- Several sensors are available in JavaScript
- Some require user consent, e.g., microphone

Sensor API

- 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

Sensor API

- 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

- Countermeasures have to address all categories

Countermeasures

- Countermeasures have to address all categories
- Should not be visible to the programmer

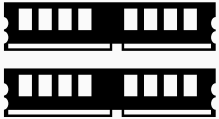
Countermeasures

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

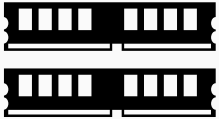
Countermeasures

- 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



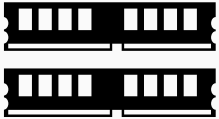
Memory Addresses



#1: Buffer ASLR

- Ensure arrays are not page aligned

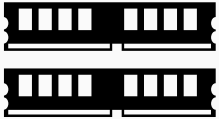
Memory Addresses



#1: Buffer ASLR

- Ensure arrays are not page aligned
- Attacker cannot assume that least significant 12 bits are ‘0’

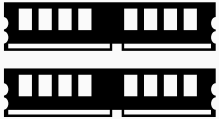
Memory Addresses



#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

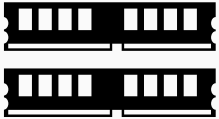
Memory Addresses



#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

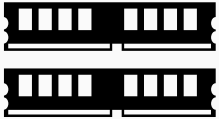
Memory Addresses



#2: Preloading

- Instead of lazy initialization for arrays, ensure that they are always memory backed

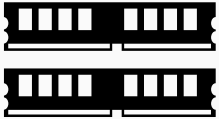
Memory Addresses



#2: Preloading

- Instead of lazy initialization for arrays, ensure that they are always memory backed
- Attacker cannot detect page borders through page faults anymore

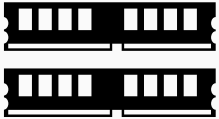
Memory Addresses



#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

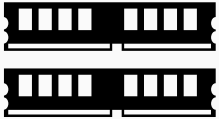
Memory Addresses



#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

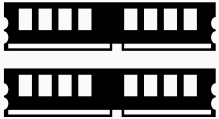
Memory Addresses



#3: Non-determinism

- For every array access, add another random access

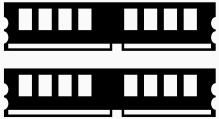
Memory Addresses



#3: Non-determinism

- For every array access, add another random access
- Makes page border detection infeasible without requiring significantly more memory

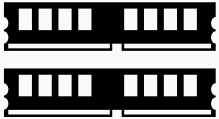
Memory Addresses



#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

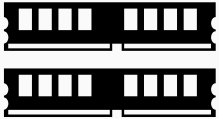
Memory Addresses



#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

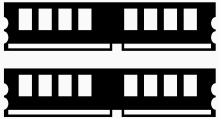
Memory Addresses



#4: Array Index Randomization

- Ensures arrays are not linear

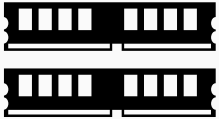
Memory Addresses



#4: Array Index Randomization

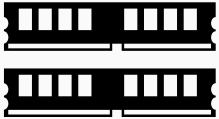
- Ensures arrays are not linear
- Use a random linear function to map array index to underlying buffer

Memory Addresses



#4: Array Index Randomization

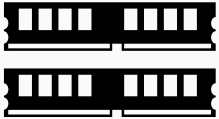
- Ensures arrays are not linear
- Use a random linear function to map array index to underlying buffer
- Index x maps to $f(x) = ax + b \mod n$, where n is array length and a and b are randomly chosen



#4: Array Index Randomization

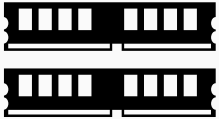
- Ensures arrays are not linear
- Use a random linear function to map array index to underlying buffer
- Index x maps to $f(x) = ax + b \mod n$, where n is array length and a and b are randomly chosen
- Has to be combined with Buffer ASLR and either Preloading or Non-determinism

Memory Addresses



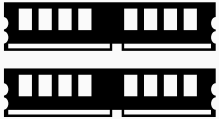
- The four defenses prevent attackers from getting virtual and physical addresses

Memory Addresses



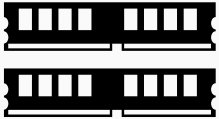
- The four defenses prevent attackers from getting virtual and physical addresses
- Prevents many microarchitectural attacks

Memory Addresses



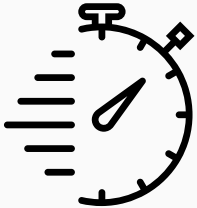
- The four defenses prevent attackers from getting virtual and physical addresses
- Prevents many microarchitectural attacks
- Have to be combined for maximum security

Memory Addresses



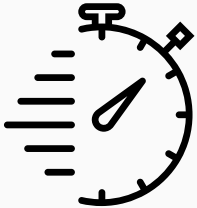
- The four defenses prevent attackers from getting virtual and physical addresses
- Prevents many microarchitectural attacks
- Have to be combined for maximum security
- Side effect: make exploits harder where addresses are required

Accurate Timing



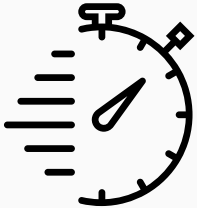
- Reducing the resolution of `performance.now()` is a first step

Accurate Timing



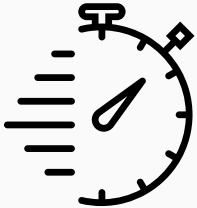
- Reducing the resolution of `performance.now()` is a first step
- Only rounding the timestamps is not sufficient

Accurate Timing



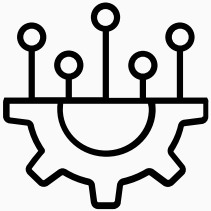
- Reducing the resolution of `performance.now()` is a first step
- Only rounding the timestamps is not sufficient
- Fuzzy time (Vattikonda et al.) adds random jitter

Accurate Timing



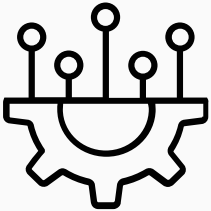
- Reducing the resolution of `performance.now()` is a first step
- Only rounding the timestamps is not sufficient
- Fuzzy time (Vattikonda et al.) adds random jitter
- Timestamps are still monotonic, but clock edges are randomized

Multithreading



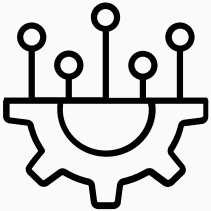
- Only real solution is to prevent multithreading

Multithreading



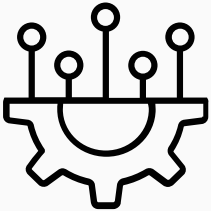
- Only real solution is to prevent multithreading
- We used a polyfill to not completely break websites

Multithreading



- 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`

Multithreading



- 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

Shared Data

- Best countermeasures: do not allow shared data

Shared Data

- Best countermeasures: do not allow shared data
- Many attacks are impossible without `SharedArrayBuffer`

Shared Data

- Best countermeasures: do not allow shared data
- Many attacks are impossible without `SharedArrayBuffer`
- Alternative: delay access to buffer

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

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

Sensor API

- Reduce resolution and update frequency of sensors

Sensor API

- Reduce resolution and update frequency of sensors
- Sensor APIs should always ask user for permission

Sensor API

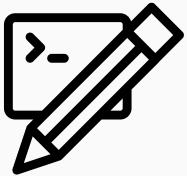
- Reduce resolution and update frequency of sensors
- Sensor APIs should always ask user for permission
- Every sensor is usable for attacks, even ambient light sensor

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
- To not break existing applications, sensors return constant value

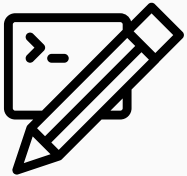
Implementation

Designing the Countermeasure



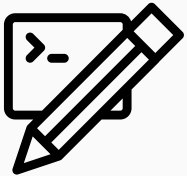
- Best solution is to implement defenses in the browser core

Designing the Countermeasure



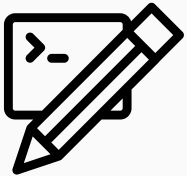
- Best solution is to implement defenses in the browser core
- Maintaining a browser fork is hard work

Designing the Countermeasure



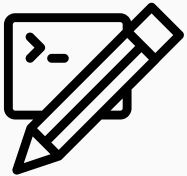
- Best solution is to implement defenses in the browser core
- Maintaining a browser fork is hard work
- We want a generic solution for multiple browsers

Designing the Countermeasure



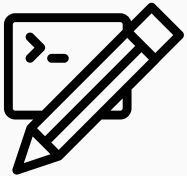
- Best solution is to implement defenses in the browser core
- Maintaining a browser fork is hard work
- We want a generic solution for multiple browsers
- Parsing JavaScript is hard

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

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
- Proof-of-concept is implemented as browser extension

User Experience

- Some defenses might impair user experience, e.g., disable multithreading

User Experience

- Some defenses might impair user experience, e.g., disable multithreading
- The user can choose one of several pre-defined protection levels

User Experience

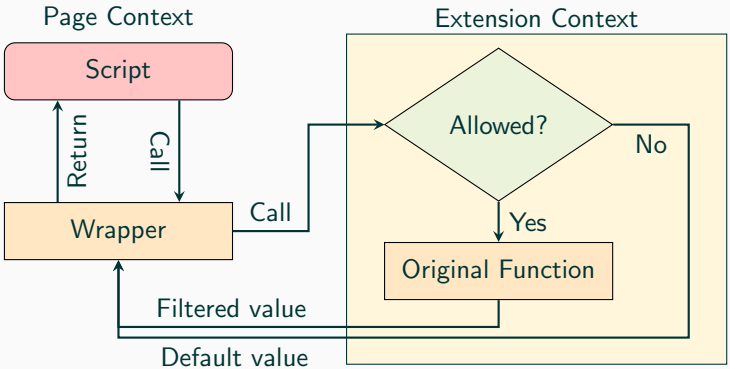
- Some defenses might impair user experience, e.g., disable multithreading
- The user can choose one of several pre-defined protection levels
- Protection levels apply different combinations of defenses

User Experience

- 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

Virtual Machine Layering

- Functions and properties are replaced by wrappers



Virtual Machine Layering

- Functions can be re-defined in JavaScript

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

Virtual Machine Layering

- Functions can be re-defined in 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)
```

Virtual Machine Layering

- Functions can be re-defined in 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));
```

Virtual Machine Layering

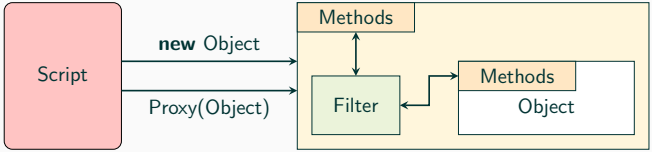
- Functions can be re-defined in JavaScript

```
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

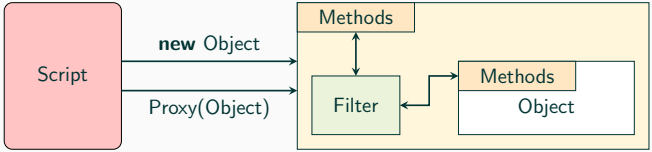
Virtual Machine Layering for Objects

- Objects are proxied



Virtual Machine Layering for Objects

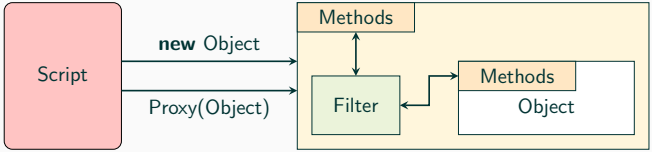
- Objects are proxied



- All properties and functions are handled by the original object

Virtual Machine Layering for Objects

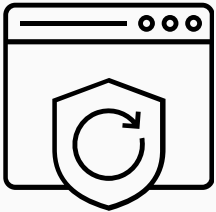
- Objects are proxied



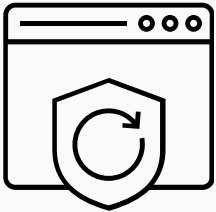
- All properties and functions are handled by the original object
- Functions and properties can be overwritten in the proxy object

Self Protection

- Attacker tries to circumvent JavaScript Zero

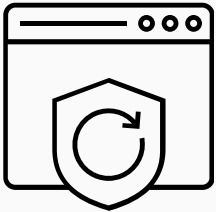


Self Protection



- Attacker tries to circumvent JavaScript Zero
- Self protection is necessary if implemented in JavaScript

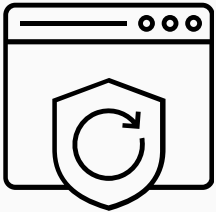
Self Protection



- Attacker tries to circumvent JavaScript Zero
- Self protection is necessary if implemented in JavaScript
- Use closures to hide all references to original functions

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

Self Protection



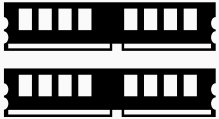
- Attacker tries to circumvent JavaScript Zero
- Self protection is necessary if implemented in JavaScript
- Use closures to hide all references to original functions

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

- Prevent objects from being modified: `Object.freeze`

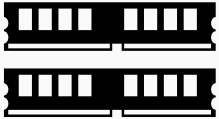
Evaluation

Page Border Detection



- Border of pages leak 12 or 21 bits (depending on page size)

Page Border Detection



- Border of pages leak 12 or 21 bits (depending on page size)
- Create huge array

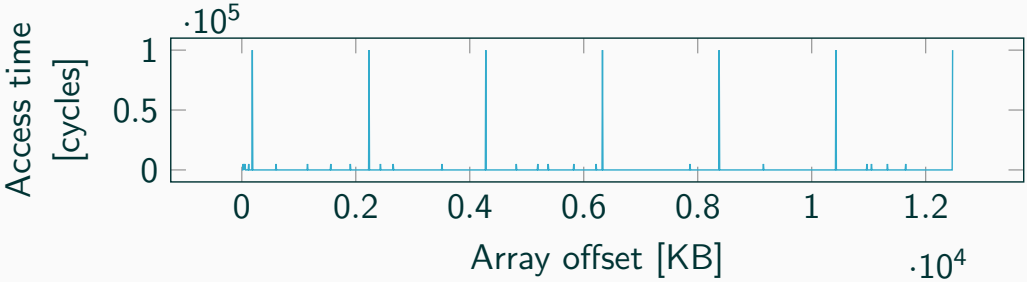
Page Border Detection

- Border of pages leak 12 or 21 bits (depending on page size)
- Create huge array
- Iterate over array, measure access time

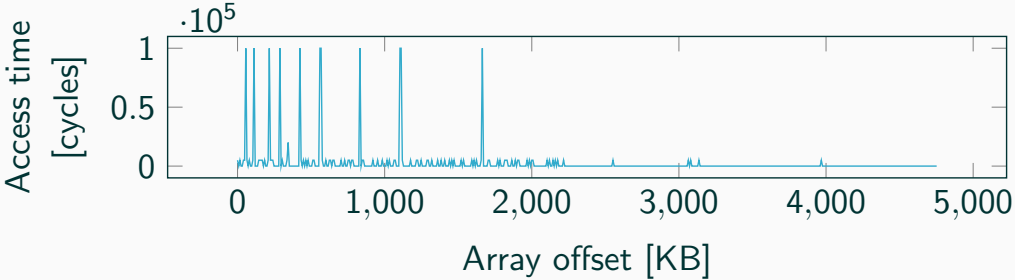
Page Border Detection

- 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

Page Border Detection

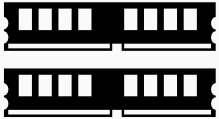


Page Border Detection with Random Access

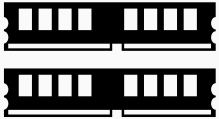


Prime+Probe

- Find addresses (= array indices) that fall into same cache set

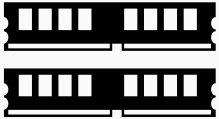


Prime+Probe



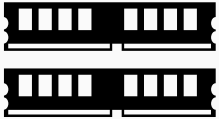
- Find addresses (= array indices) that fall into same cache set
- Physical address defines in which cache set the data is cached

Prime+Probe



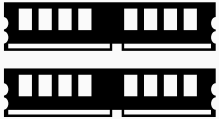
- Find addresses (= array indices) that fall into same cache set
- Physical address defines in which cache set the data is cached
- Enough addresses in one set evicts the set (Prime)

Prime+Probe



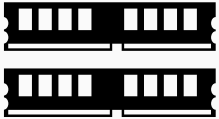
- Find addresses (= array indices) that fall into same cache set
- Physical address defines in which cache set the data is cached
- Enough addresses in one set evicts the set (Prime)
- Iterate again over addresses (Probe)

Prime+Probe



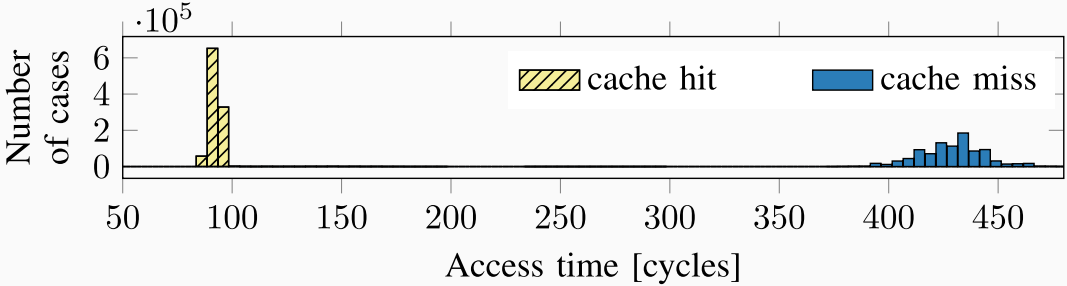
- Find addresses (= array indices) that fall into same cache set
- Physical address defines in which cache set the data is cached
- Enough addresses in one set evicts the set (Prime)
- Iterate again over addresses (Probe)
- If it is fast, they are still cached

Prime+Probe

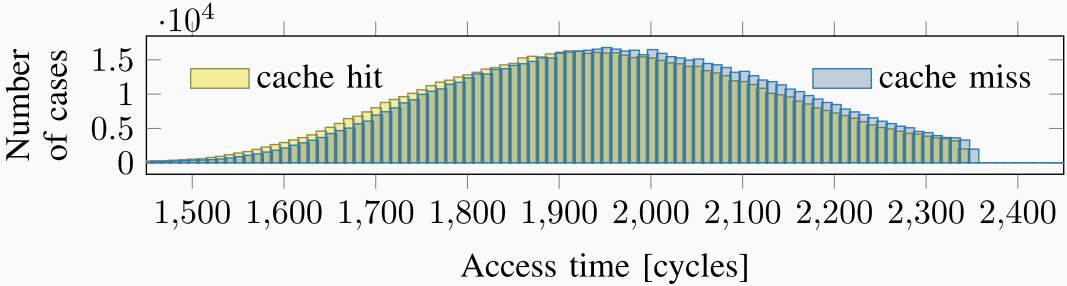


- Find addresses (= array indices) that fall into same cache set
- Physical address defines in which cache set the data is cached
- Enough addresses in one set evicts the set (Prime)
- Iterate again over addresses (Probe)
- If it is fast, they are still cached
- If it is slow, someone used this cache set and evicted our addresses

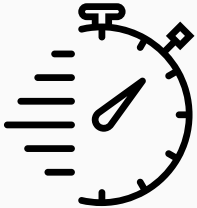
Prime+Probe


$\cdot 10^5$
Number of cases
6
4
2
0
cache hit
cache miss
50
100
150
200
250
300
350
400
450
Access time [cycles]

Prime+Probe with Random Access

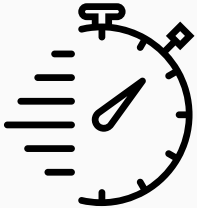


Interrupt Detection



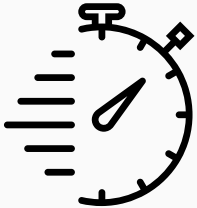
- Multithreading allows to detect interrupts

Interrupt Detection



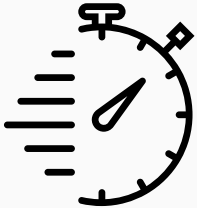
- Multithreading allows to detect interrupts
- Endless loop which counts number of increments in time window

Interrupt Detection



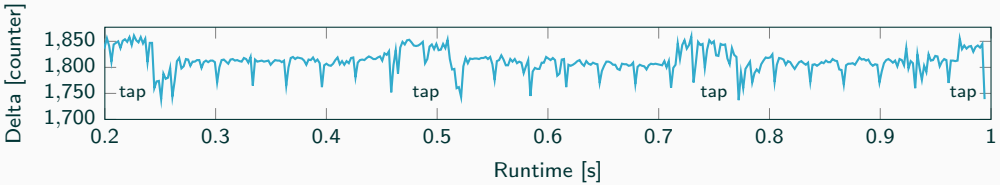
- Multithreading allows to detect interrupts
- Endless loop which counts number of increments in time window
- Different number of increments indicate interrupt

Interrupt Detection

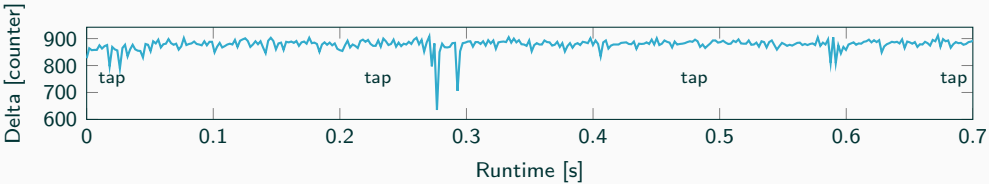


- Multithreading allows to detect interrupts
- Endless loop which counts number of increments in time window
- Different number of increments indicate interrupt
- Fuzzy time prevents deterministic equally-sized time window

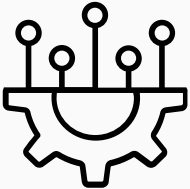
Interrupt Detection



Interrupt Detection with Fuzzy Time

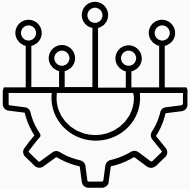


Event Queue Spying



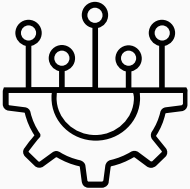
- Messages between web workers are handled in the event queue

Event Queue Spying



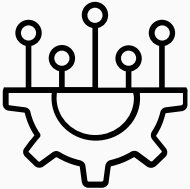
- Messages between web workers are handled in the event queue
- User activity is also handled in the event queue

Event Queue Spying



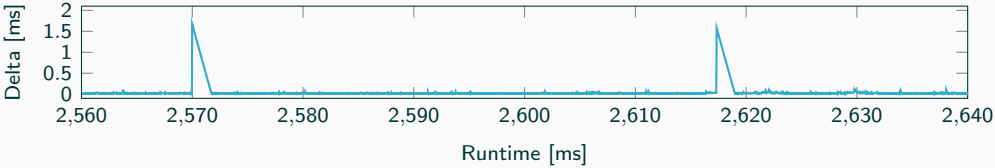
- Messages between web workers are handled in the event queue
- User activity is also handled in the event queue
- Posting many messages allows to measure latency

Event Queue Spying

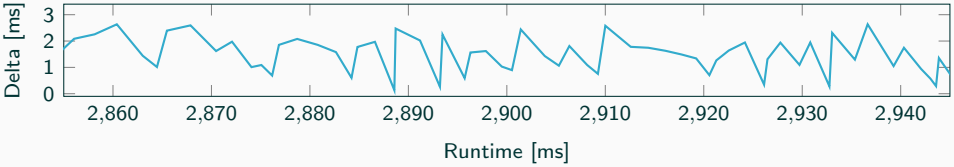


- Messages between web workers are handled in the event queue
- User activity is also handled in the event queue
- Posting many messages allows to measure latency
- Latency indicates user input

Event Queue Spying



Event Queue Spying with Message Delay



`SharedArrayBuffer` Timing Primitive

- `SharedArrayBuffer` allows to build a timing primitive with the highest resolution

`SharedArrayBuffer` Timing Primitive

- `SharedArrayBuffer` allows to build a timing primitive with the highest resolution
- One web worker continuously increments variable in the shared array

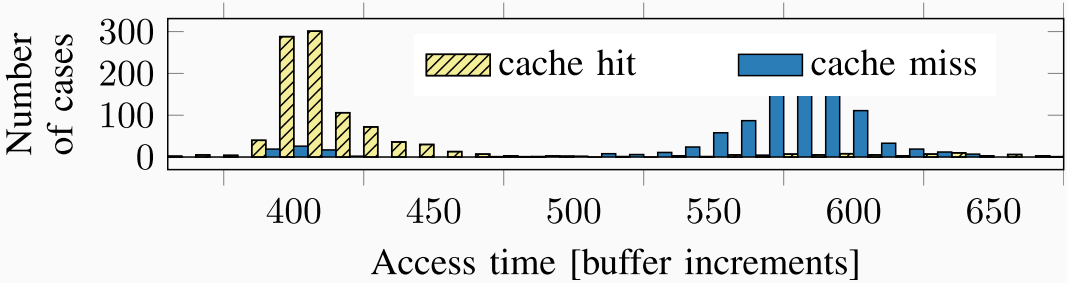
`SharedArrayBuffer` Timing Primitive

- `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

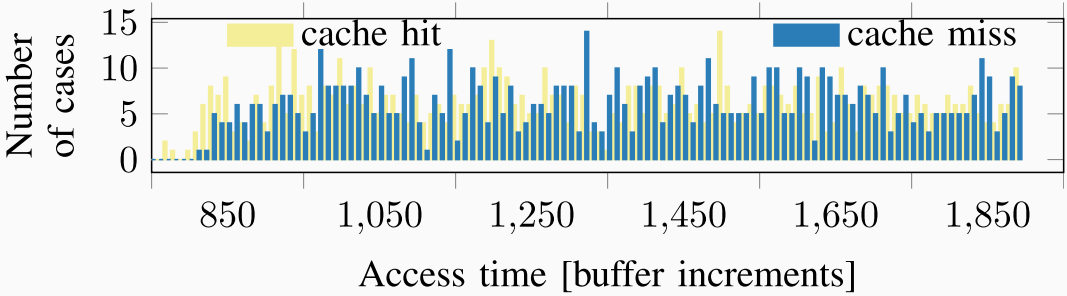
`SharedArrayBuffer` Timing Primitive

- `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



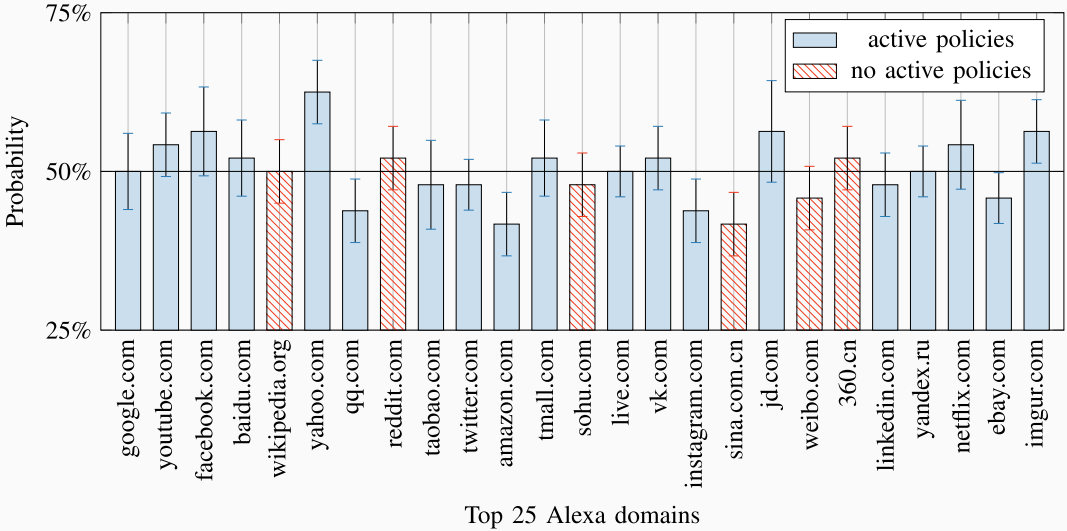
SharedArrayBuffer with Random Delay



Defense \ Prevents	Rowhammer.js	Page Deduplication	DRAM Covert Channel	Anti-ASLR	Cache Eviction	Keystroke Timing	Browser
Buffer ASLR	○	◐	○	●	●	○	○
Array preloading	●	○	●	○	○	○	○
Non-deterministic array	●	◐	◐	●	●	○	○
Array index randomization	○	●	○	●	○	○	○
Low-resolution timestamp	○	◐	○	○	○	◐	◐
Fuzzy time	○	◐*	○	○*	○	●*	●*
`WebWorker` polyfill	○	○	●	●	●	●	○
Message delay	○	○	○	○	○	◐	◐
Slow `SharedArrayBuffer`	○	○	●	◐	●	○	○
No `SharedArrayBuffer`	○	○*	●	●*	●	○*	○*
Summary	●	●	●	●	●	●	●

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



Conclusion

- Just rounding timers is not sufficient

Conclusion

- Just rounding timers is not sufficient
- Multithreading and shared data allow to build new timers

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

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

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
- More microarchitectural attacks in JavaScript will appear