# JavaScript Zero

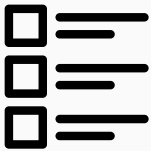
## Real JavaScript and Zero Side-Channel Attacks

**Michael Schwarz**, Moritz Lipp, Daniel Gruss

20.02.2018

www.iaik.tugraz.at

# Outline



- Analysis of current microarchitectural and side-channel attacks
- Identifying building blocks for attacks
- Countermeasures for preventing attacks
- Implementation of countermeasures
- Evaluation of countermeasures

# Landscape of Microarchitectural Attacks

- Currently 11 microarchitectural and side-channel attacks in JavaScript

# Landscape of Microarchitectural Attacks

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

# Landscape of Microarchitectural Attacks

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

# Landscape of Microarchitectural Attacks

- 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

# Landscape of Microarchitectural Attacks

- 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

# Memory Addresses

- Language does not provide addresses to programmer

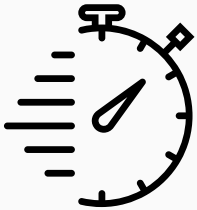
# Memory Addresses

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

# Memory Addresses

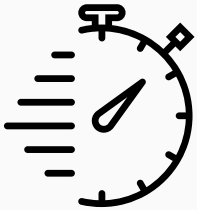
- Language does not provide addresses to programmer
- Closest to virtual address: array indices
- Detect beginning of physical pages through high timing on page faults

# Accurate Timing



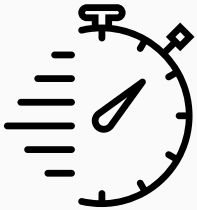
- Nearly all attacks require accurate timing

# Accurate Timing



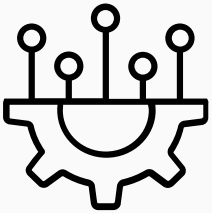
- Nearly all attacks require accurate timing
- No absolute timestamps required, only time differences

# Accurate Timing



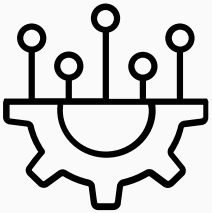
- Nearly all attacks require accurate timing
- No absolute timestamps required, only time differences
- Required accuracy varies between milliseconds and nanoseconds

# Multithreading



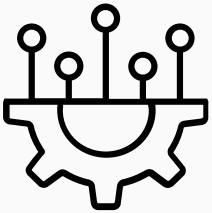
- JavaScript introduced multi threading with web workers

# Multithreading



- JavaScript introduced multi threading with web workers
- Real concurrency in applications

# Multithreading



- JavaScript introduced multi threading with web workers
- Real concurrency in applications
- Enables new side-channel attacks

# 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
- 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
- Some sensors can be used without user consent, e.g., ambient light

# Sensor API

- Some side-channel attacks only require access to sensors
- Some sensors can be used without user consent, e.g., ambient light
- Every sensor is exploitable

# 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

- Ensure arrays are memory backed and not linear

- Ensure arrays are memory backed and not linear
- Additionally, add random dummy accesses

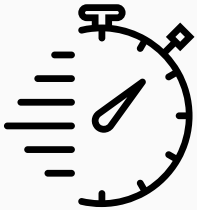
# Memory Addresses

- Ensure arrays are memory backed and not linear
- Additionally, add random dummy accesses
- Prevents many microarchitectural attacks

# Memory Addresses

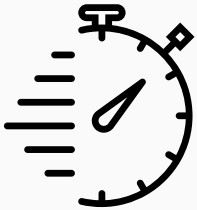
- Ensure arrays are memory backed and not linear
- Additionally, add random dummy accesses
- Prevents many microarchitectural attacks
- Side effect: make exploits harder where addresses are required

# Accurate Timing



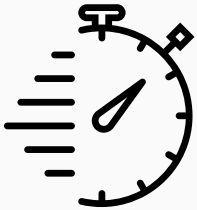
- Reducing the resolution of `performance.now()` is a first step

# Accurate Timing



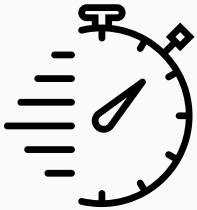
- Reducing the resolution of `performance.now()` is a first step
- Only rounding the timestamps is not sufficient

# Accurate Timing

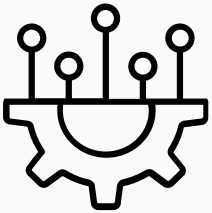


- Reducing the resolution of `performance.now()` is a first step
- Only rounding the timestamps is not sufficient
- Fuzzy time (Vattikonda et al.) adds random jitter

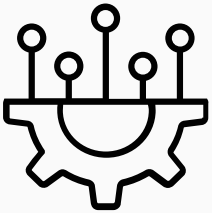
# Accurate Timing



- 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

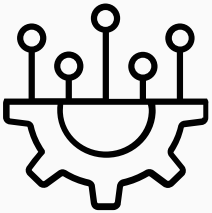


- Only real solution is to prevent multithreading



- Only real solution is to prevent multithreading
- Some attacks can be prevented by adding random delays to `postMessage`

# Multithreading



- Only real solution is to prevent multithreading
- 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 exploit fast concurrent access to `SharedArrayBuffer`

# Shared Data

- Best countermeasures: do not allow shared data
- Many attacks exploit fast concurrent access to `SharedArrayBuffer`
- Alternative: delay access to buffer

# Shared Data

- Best countermeasures: do not allow shared data
- Many attacks exploit fast concurrent access to `SharedArrayBuffer`
- Alternative: delay access to buffer
- Degrades resolution of timing primitive to microseconds

- 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

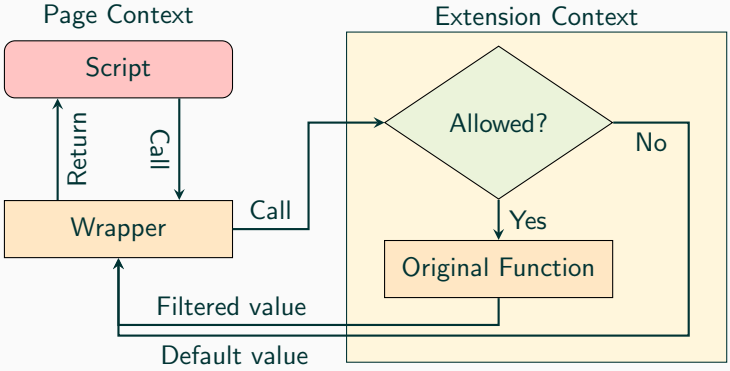
# Designing the Countermeasure

- 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

- Functions and properties are replaced by wrappers



- Functions can be re-defined in JavaScript

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

- 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)
```

- Functions can be re-defined in JavaScript

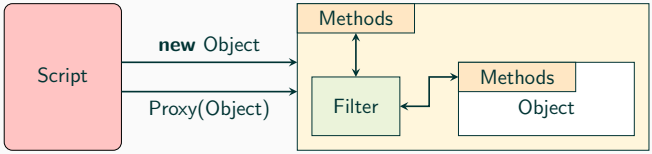
```
var original_reference = window.performance.now;
window.performance.now = function() { return 0; };

// 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));
```

- Functions can be re-defined in JavaScript

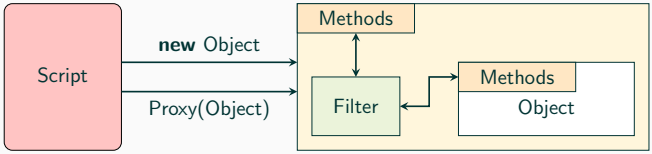
```
var original_reference = window.performance.now;
window.performance.now = function() { return 0; };

// 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

- Objects are proxied

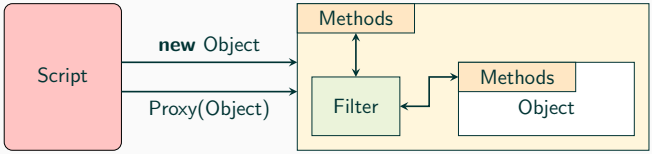


- Objects are proxied



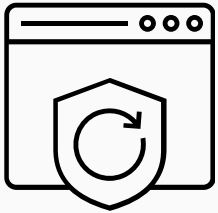

- All properties and functions are handled by the original object

- Objects are proxied

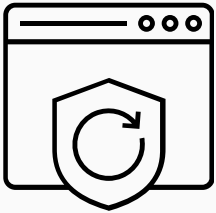



- All properties and functions are handled by the original object
- Functions and properties can be overwritten in the proxy object

- Attacker tries to circumvent JavaScript Zero

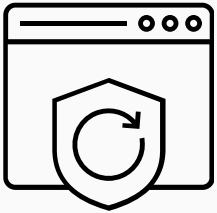


# Self Protection



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

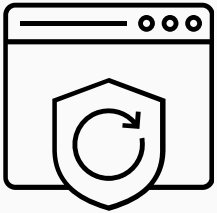
# Self Protection



- 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

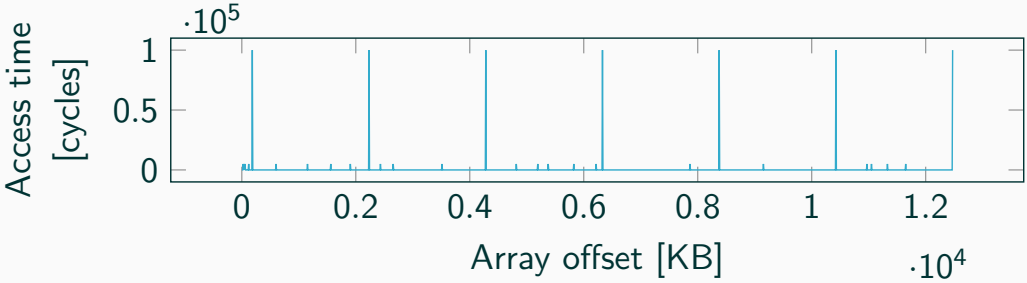
# Page Border Detection

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

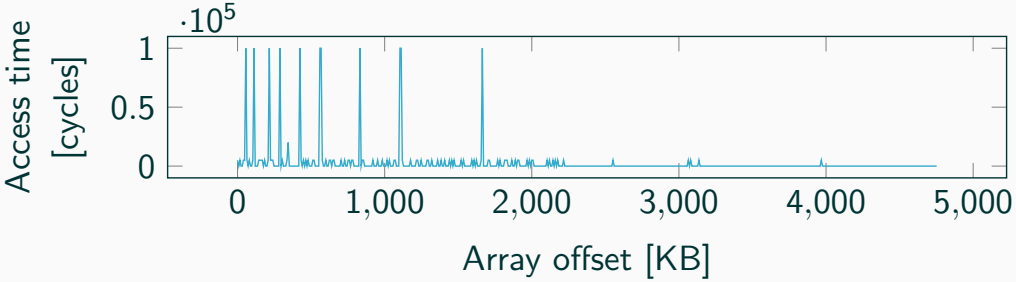
# Page Border Detection

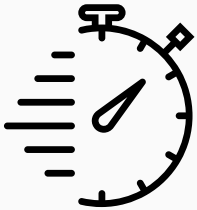
- Border of pages leak 12 or 21 bits (depending on page size)
- Create huge array
- Iterate over array, measure access time
- Page border raise pagefault, taking significantly longer to access

# Page Border Detection

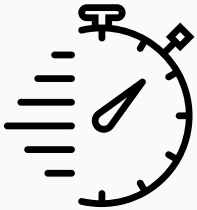


# Page Border Detection with Random Access



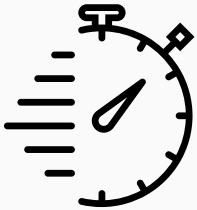


- Multithreading allows to detect interrupts



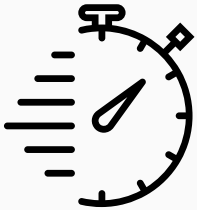
- Multithreading allows to detect interrupts
- Endless loop which counts number of increments in time window

# Interrupt Detection



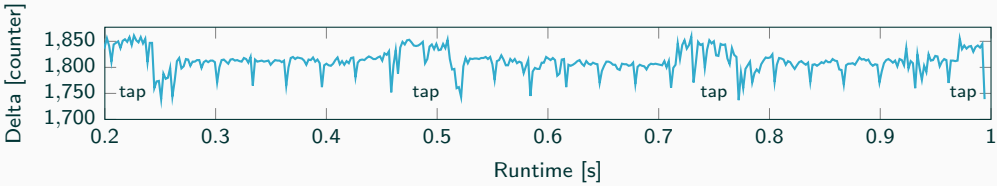
- Multithreading allows to detect interrupts
- Endless loop which counts number of increments in time window
- Different number of increments indicate interrupt

# Interrupt Detection

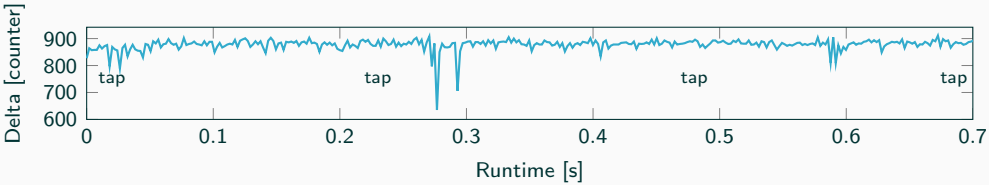


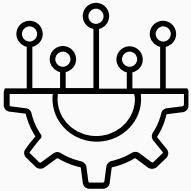
- Multithreading allows to detect interrupts
- Endless loop which counts number of increments in time window
- Different number of increments indicate interrupt
- Fuzzy time prevents deterministic equally-sized time window

# Interrupt Detection



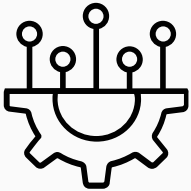
# Interrupt Detection with Fuzzy Time





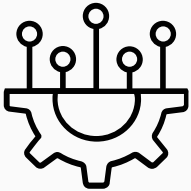
- Messages between web workers are handled in the event queue

# Event Queue Spying



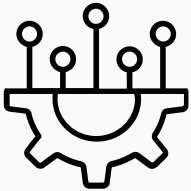
- Messages between web workers are handled in the event queue
- User activity is also handled in the event queue

# Event Queue Spying



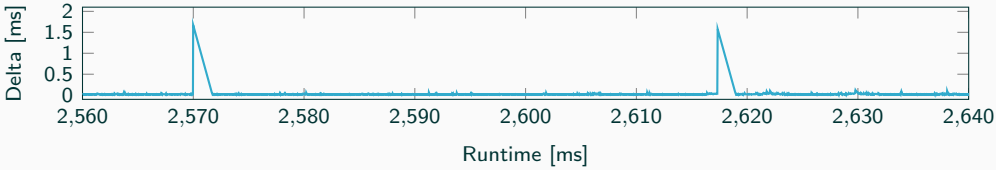
- Messages between web workers are handled in the event queue
- User activity is also handled in the event queue
- Posting many messages allows to measure latency

# Event Queue Spying

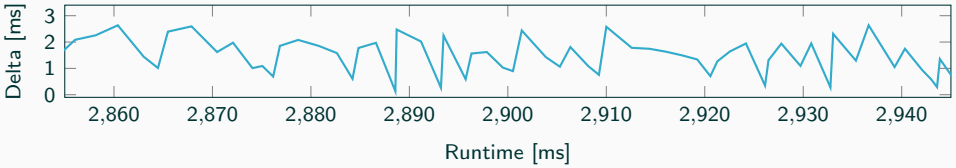


- Messages between web workers are handled in the event queue
- User activity is also handled in the event queue
- Posting many messages allows to measure latency
- Latency indicates user input

# 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

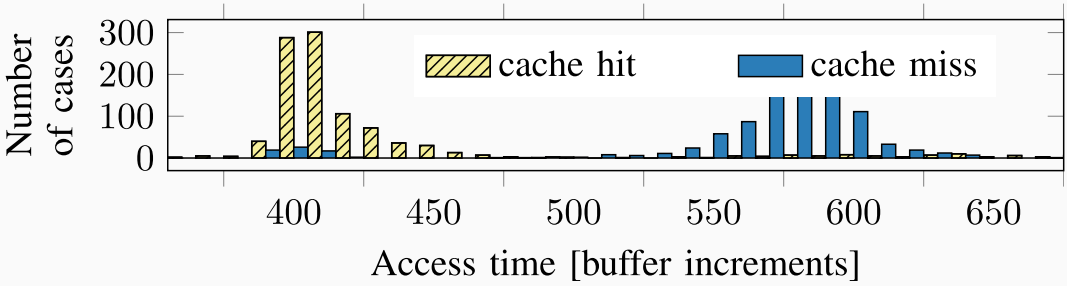
- `SharedArrayBuffer` allows to build a timing primitive with the highest resolution
- One web worker continuously increments variable in the shared array

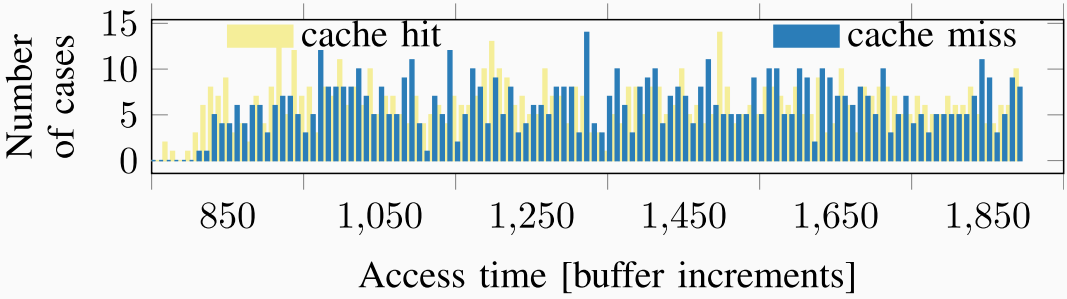
# `SharedArrayBuffer` Timing Primitive

- `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

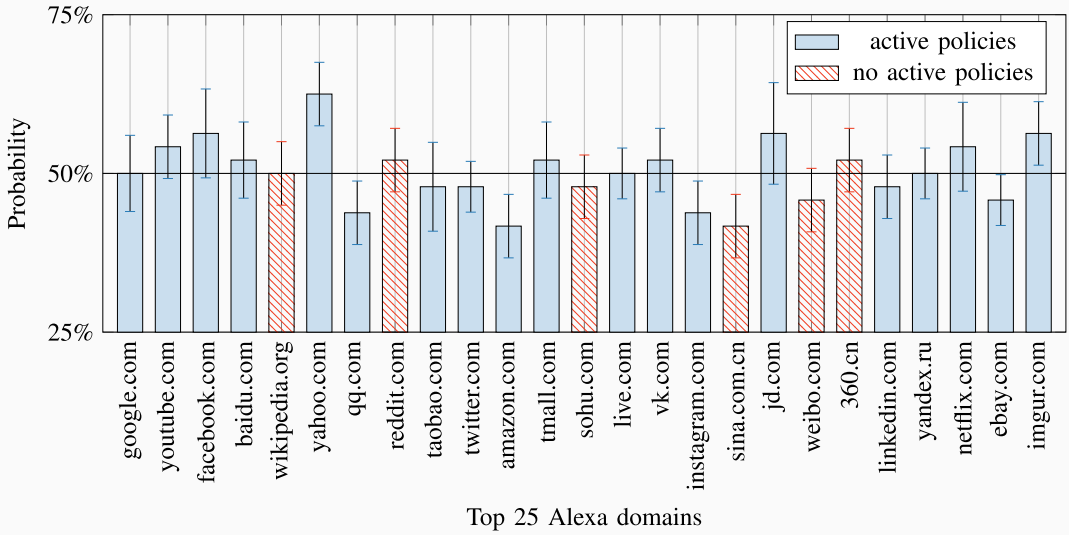

Number of cases
300
200
100
0
cache hit
cache miss
400
450
500
550
600
650
Access time [buffer increments]


Number of cases
15
10
5
0
cache hit
cache miss
850
1,050
1,250
1,450
1,650
1,850
Access time [buffer increments]

| Defense \ Prevents | Rowham-mer.js | Page Dedu-plication | 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.


75%
50%
25%
Probability
active policies
no active policies
google.com
youtube.com
facebook.com
baidu.com
wikipedia.org
yahoo.com
qq.com
reddit.com
taobao.com
twitter.com
amazon.com
tmall.com
sohu.com
live.com
vk.com
instagram.com
sina.com.cn
jd.com
weibo.com
360.cn
linkedin.com
yandex.ru
netflix.com
ebay.com
imgur.com
Top 25 Alexa domains

# 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

# JavaScript zero

## REAL JavaScript AND ZERO SIDE-CHANNEL ATTACKS

Michael Schwarz, Moritz Lipp, Daniel Gruss