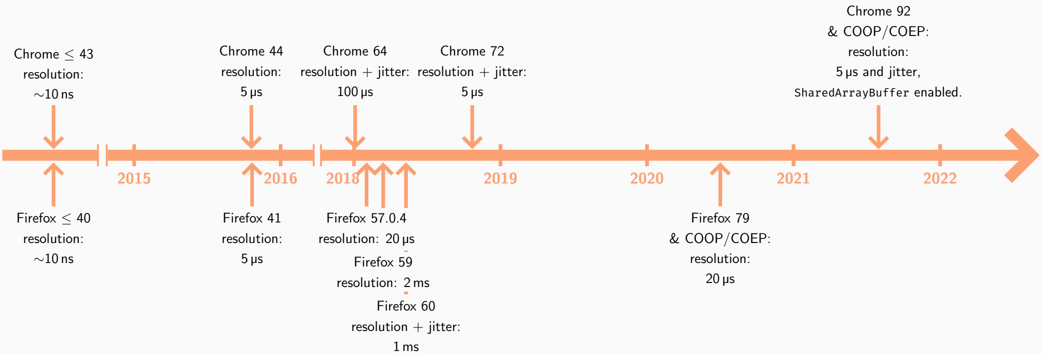
Side Channels in Web Browsers: Applications to Security and Privacy

Thomas Rokicki

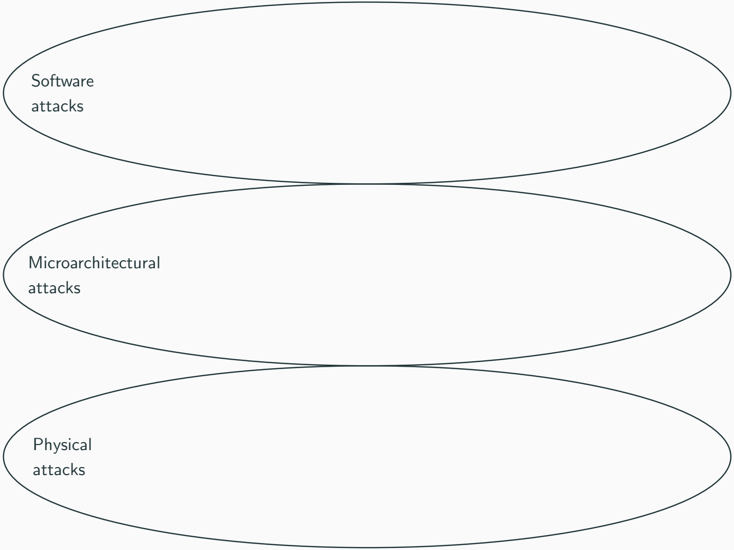
29/11/2022

INSA Rennes, CNRS, IRISA
Supervisors: Clémentine Maurice and Gildas Avoine.

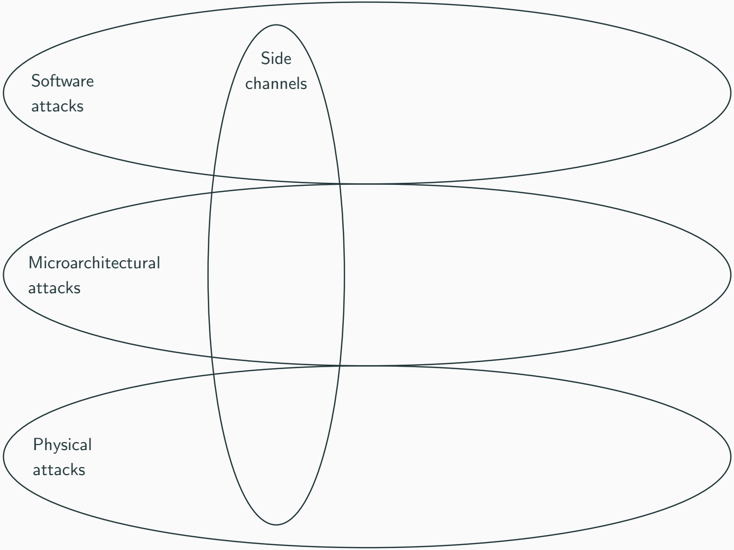
Context


Chrome ≤ 43 resolution: ~10 ns
Chrome 44 resolution: 5 µs
Chrome 64 resolution + jitter: 100 µs
Chrome 72 resolution + jitter: 5 µs
Chrome 92 & COOP/COEP: resolution: 5 µs and jitter, SharedArrayBuffer enabled.
2015
2016
2018
2019
2020
2021
2022
Firefox ≤ 40 resolution: ~10 ns
Firefox 41 resolution: 5 µs
Firefox 57.0.4 resolution: 20 µs
Firefox 59 resolution: 2 ms
Firefox 60 resolution + jitter: 1 ms
Firefox 79 & COOP/COEP: resolution: 20 µs

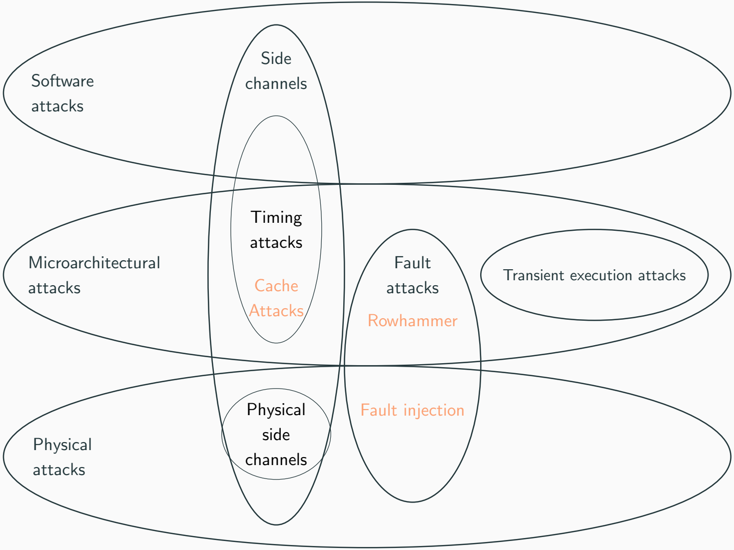
Context of the contributions



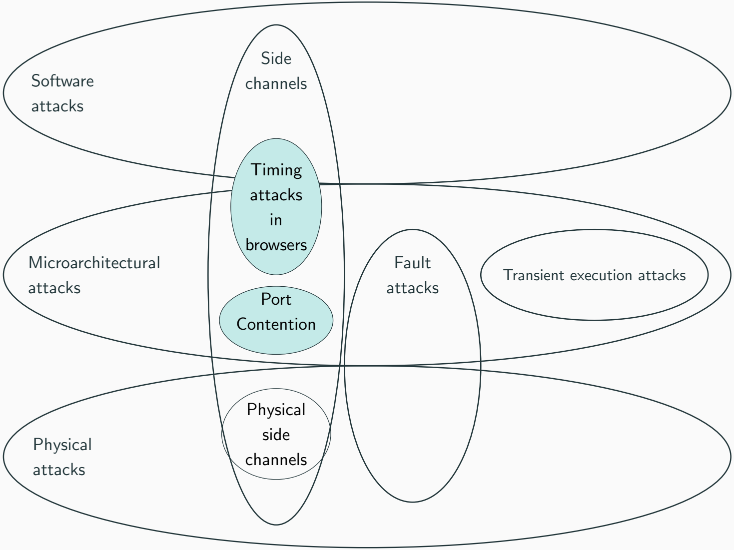
Context of the contributions



Context of the contributions



Context of the contributions



Microarchitectural Side Channels

- Hardware optimizations are designed for performance and not security.
- Attackers can exploit timing differences caused by microarchitectural optimizations.
- Cache attacks are probably the most common.
- Applications to cryptography, covert channels, breaking isolation.

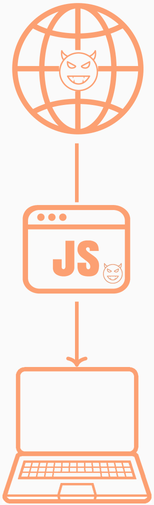
Microarchitectural Side Channels

- Hardware optimizations are designed for performance and not security.
- Attackers can exploit timing differences caused by microarchitectural optimizations.
- Cache attacks are probably the most common.
- Applications to cryptography, covert channels, breaking isolation.

Common prerequisite:
Execute code on shared hardware

Microarchitectural attacks in the browsers: Threat model

- JavaScript or WebAssembly code are client-side languages.

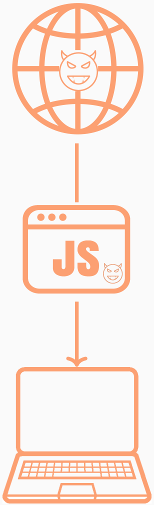


Microarchitectural attacks in the browsers: Threat model

- JavaScript or WebAssembly code are client-side languages.
- The user visits a malicious website and downloads the code.

Microarchitectural attacks in the browsers: Threat model

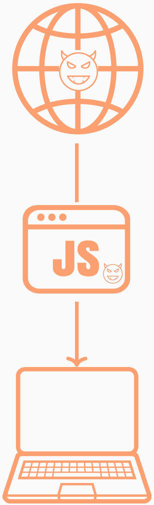
- JavaScript or WebAssembly code are client-side languages.
- The user visits a malicious website and downloads the code.
- They execute it on their machine.



Microarchitectural attacks in the browsers: Threat model

- JavaScript or WebAssembly code are client-side languages.
- The user visits a malicious website and downloads the code.
- They execute it on their machine.

Prerequisite matched!



Microarchitectural attacks in the browsers: Common Challenges

- Client-side languages are sandboxed:
  - No native instructions.
  - Oblivious to memory addresses.
  - No access to the filesystem.
- Timers are restricted.
- High-level interpreted languages:
  - JavaScript especially high level.
  - WebAssembly offers more atomic operations.

Research Questions

- With everchanging browsers and microarchitecture, how can we evaluate the threat posed by side channels?
- What side channels can we implement in the browser?
- What information can we extract from these side channels?

Talk Outline

This defense is composed of three major sections:

- **EuroS&P 2021** Systematic analysis of JavaScript timers.
- **AsiaCCS 2022** Port contention in the browser.
- **ESORICS 2022** Port contention without SMT.

In Search of Lost Time: A Survey of JavaScript Timers

Contributions of this Section

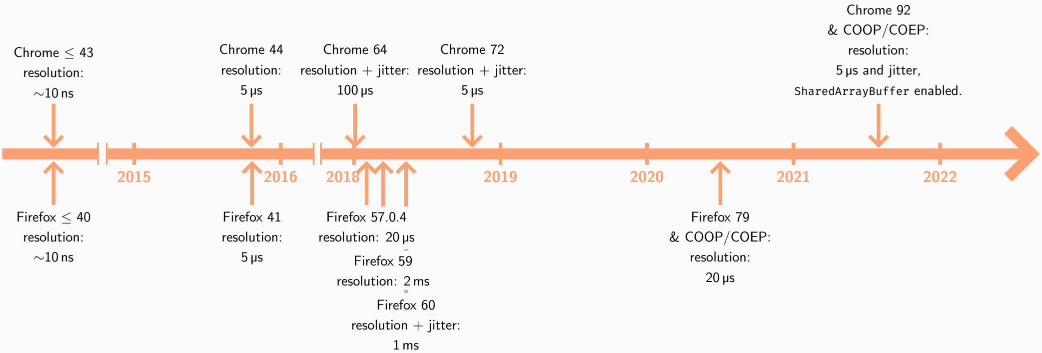
- Classification of browser-based timing attacks.
- Framework to automatically evaluate JavaScript timers.
- Longitudinal study of browsers' timing-based security.

Classification of JavaScript timing attacks

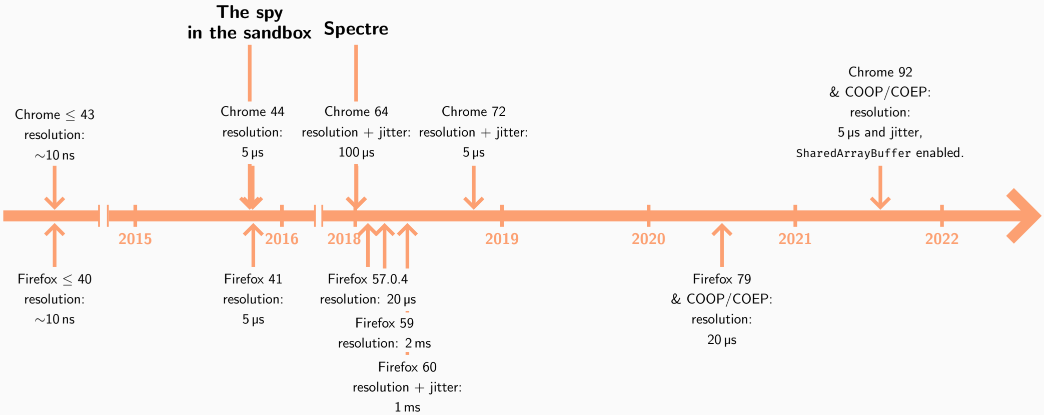
- Hardware-contention-based attacks;
- Transient execution attacks;
- Attacks based on system resources;
- Attacks based on browser resources.

Common prerequisite: Timers.

JavaScript and Timers: A Complicated History



JavaScript and Timers: A Complicated History



JavaScript Timers

Built-in timers have a resolution ranging from 5-100 µs.

We have to create our auxiliary timers[1]:

- Clock Interpolation.

[1] Schwarz et al., Financial Crypto 2017.

JavaScript Timers

Built-in timers have a resolution ranging from 5-100 μs.

We have to create our auxiliary timers[1]:

- Clock Interpolation.
- `SharedArrayBuffer`.

[1] Schwarz et al., Financial Crypto 2017.

JavaScript Timers

Built-in timers have a resolution ranging from 5-100 µs.

We have to create our auxiliary timers[1]:

- Clock Interpolation. **Patch:** Add jitter.
- `SharedArrayBuffer`.

[1]Schwarz et al., Financial Crypto 2017.

JavaScript Timers

Built-in timers have a resolution ranging from 5-100 µs.

We have to create our auxiliary timers[1]:

- Clock Interpolation. **Patch:** Add jitter.
- `SharedArrayBuffer`. **Patch:** Disable `SharedArrayBuffer`.

[1] Schwarz et al., Financial Crypto 2017.

Security vs. Practicality

- These features are needed for development.

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures:

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures: **Isolation-based**.

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures: **Isolation-based**.
- Site isolation and COOP/COEP:

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures: **Isolation-based**.
- Site isolation and COOP/COEP:
  - Each tab/origin runs in a different process.

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures: **Isolation-based**.
- Site isolation and COOP/COEP:
  - Each tab/origin runs in a different process.
  - Different processes mean different address spaces.

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures: **Isolation-based**.
- Site isolation and COOP/COEP:
  - Each tab/origin runs in a different process.
  - Different processes mean different address spaces.
    - Spectre v1 is **mitigated**!

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures: **Isolation-based**.
- Site isolation and COOP/COEP:
  - Each tab/origin runs in a different process.
  - Different processes mean different address spaces.
    - Spectre v1 is **mitigated**!
    - Other attacks are **not impacted**.

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures: **Isolation-based**.
- Site isolation and COOP/COEP:
  - Each tab/origin runs in a different process.
  - Different processes mean different address spaces.
    - Spectre v1 is **mitigated**!
    - Other attacks are **not impacted**.
- Timing-based countermeasures are obsolete:
  - Grant higher resolution and less jitter.
  - Reactivate `SharedArrayBuffer`.

Security vs. Practicality

- These features are needed for development.
- Browser vendors want less penalizing countermeasures: **Isolation-based**.
- Site isolation and COOP/COEP:
  - Each tab/origin runs in a different process.
  - Different processes mean different address spaces.
    - Spectre v1 is **mitigated**!
    - Other attacks are **not impacted**.
- Timing-based countermeasures are obsolete:
  - Grant higher resolution and less jitter.
  - Reactivate `SharedArrayBuffer`.

What are the security implications of reintroducing high-resolution timers?

Automated Framework

Automated framework to evaluate JavaScript timers using Selenium.

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

Automated Framework

Automated framework to evaluate JavaScript timers using Selenium.
Works on Chrome and Firefox, including past and future versions.

Our goal is that this analysis can be helpful at this point and in the future.

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

Framework behaviour

For each timer, we evaluate:

Resolution	Smallest operation a timer can measure.
Performance overhead	Time it takes to make the measurement.

Framework behaviour

For each timer, we evaluate:

Resolution	Smallest operation a timer can measure.
Performance overhead	Time it takes to make the measurement.

You can find more in-depth details of the experiments and results in the full paper.

Some perspective

On Firefox 88 (2021) vs. Firefox 78 (2018), an attacker can:

Some perspective

On Firefox 88 (2021) vs. Firefox 78 (2018), an attacker can:

- Create a cache covert channel **800,000** faster.

Some perspective

On Firefox 88 (2021) vs. Firefox 78 (2018), an attacker can:

- Create a cache covert channel **800,000** faster.

- Mount cache attacks in **a matter of seconds** *vs* tens of minutes

Some perspective

On Firefox 88 (2021) vs. Firefox 78 (2018), an attacker can:

- Create a cache covert channel **800,000** faster.

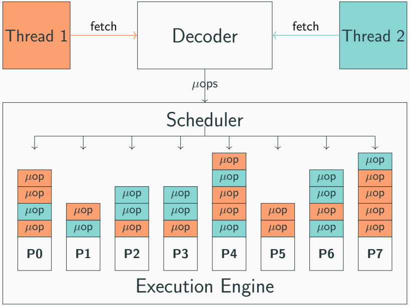
- Mount cache attacks in **a matter of seconds** *vs* tens of minutes

Timing attacks are more of a threat than 4 years ago.

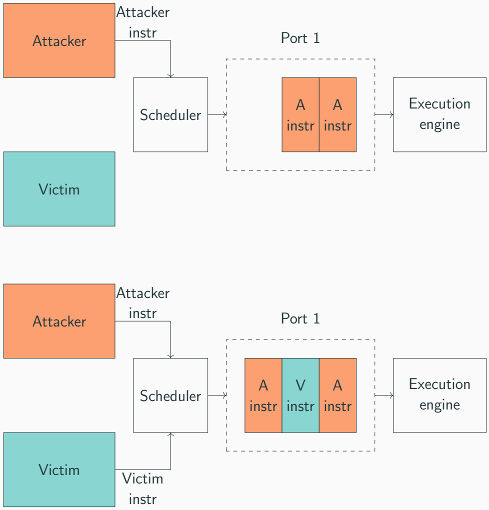
Port Contention Goes Portable

Background: Execution pipeline

- Shared by all threads on the physical core.
- Instructions are decomposed in **micro-operations** (μops).
- The decomposition is **deterministic**.
- μops are dispatched to specialized execution units through **CPU ports**.



Background: Port contention[2]



No Contention All the attacker instructions are executed in a row, **fast execution time**.

Contention Attacker instructions are delayed, **slow execution time**.

[2] Aldaya et al., S&P 2019

C1 - Core control

JavaScript does not have core control.

C1 - Core control

JavaScript does not have core control.

The scheduler tries to balance the workload of **physical** cores.

C1 - Core control

JavaScript does not have core control.

The scheduler tries to balance the workload of **physical** cores.

Solution: Exploit JavaScript multithreading and work with the scheduler.

C2 - High-resolution timers

Baseline JavaScript timers are not sufficient to mount our attacks.

C2 - High-resolution timers

Baseline JavaScript timers are not sufficient to mount our attacks.

We use auxiliary timers based on `SharedArrayBuffer`.

C2 - High-resolution timers

Baseline JavaScript timers are not sufficient to mount our attacks.

We use auxiliary timers based on `SharedArrayBuffer`.

Puts more constraints on core control.

C3 - Port usage of WebAssembly

WebAssembly is a high-level language.

We need to find out the port usage of WebAssembly instructions.

So we built **PC-Detector**

Test the contention of 244 WebAssembly instructions with our knowledge of native port usage.

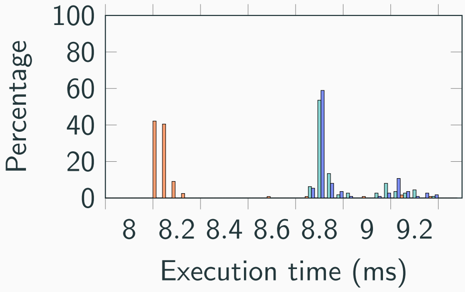
PC-Detector - Description

Control	The web script runs alone in the browser.
Contention on Port x	The web script runs while we create Px contention.

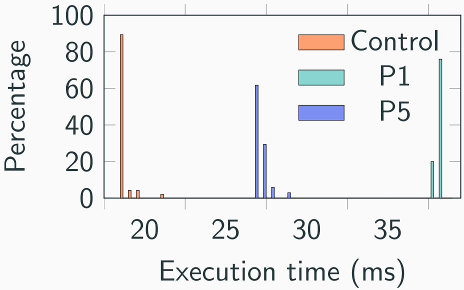
PC-Detector - Description

Control The web script runs alone in the browser.

Contention on Port x The web script runs while we create Px contention.



(a) Result for instruction `f64.floor`



(b) Result for instruction `i64.rem_u`

PC-Detector - Results

We tested over 200 different instructions.

- 80 instructions creating contention.
  - Some create more timing difference.
  - `i64.rem_u` seems to cause the most difference in timing.
- Contention on 4 ports: 0, 1, 5, and 6.
  - More threat surface!
  - Ports 2 and 3 have the exact same usage, so execution is always parallelized.

Side-Channel Artificial Example - Description

Generic example of a side channel attack. Web user attacks a native victim and extracts a secret.

Victim

`secret == 0` →

```
POPCNT %r8,%r8
POPCNT %r8,%r8
...
POPCNT %r8,%r8
POPCNT %r8,%r8
```

Contention on Port 1

`secret == 1` →

```
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
...
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 5

Monitors port usage

Side-Channel Artificial Example - Description

Generic example of a side channel attack. Web user attacks a native victim and extracts a secret.

Victim

`secret == 0`

```
POPCNT %r8,%r8
POPCNT %r8,%r8
...
POPCNT %r8,%r8
POPCNT %r8,%r8
```

`secret == 1`

```
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
...
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 1

Secret is 0!

Side-Channel Artificial Example - Description

Generic example of a side channel attack. Web user attacks a native victim and extracts a secret.

Victim

`secret == 0`

```
POPCNT %r8,%r8
POPCNT %r8,%r8
...
POPCNT %r8,%r8
POPCNT %r8,%r8
```

`secret == 1`

```
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
...
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 5

Secret is 1!

Side-Channel Artificial Example - Results

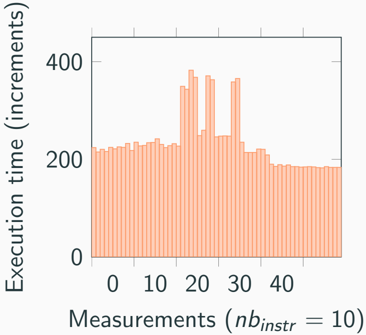


Figure 3: Secret key: 1101001.

- Able to detect 1024 native instructions in a single trace.

Side-Channel Artificial Example - Results

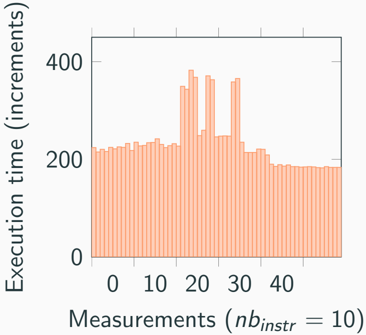


Figure 3: Secret key: 1101001.

- Able to detect 1024 native instructions in a single trace.
- Spatial resolution similar to web-based cache attacks (Prime+Probe).

Side-Channel Artificial Example - Results

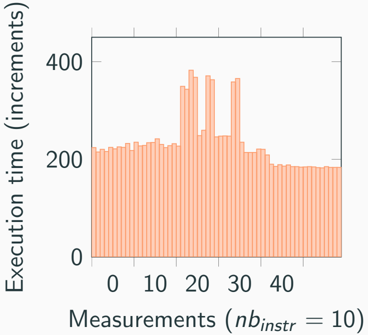


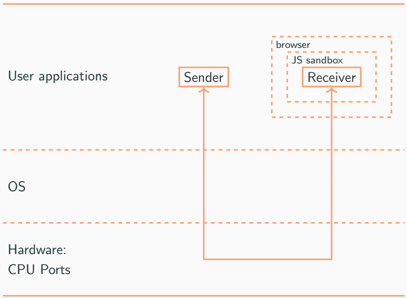
Figure 3: Secret key: 1101001.

- Able to detect 1024 native instructions in a single trace.
- Spatial resolution similar to web-based cache attacks (Prime+Probe).
- Timers are the main bottleneck.

Covert Channel

Composed of two components:

- **Native:** C/x86 sender.
- **Web:** JavaScript/WebAssembly receiver.



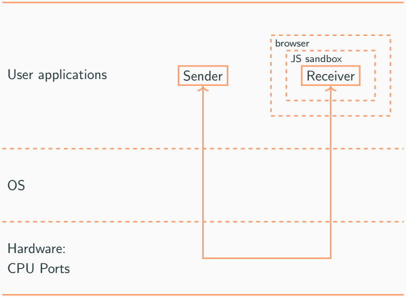
Covert Channel

Composed of two components:

- **Native:** C/x86 sender.
- **Web:** JavaScript/WebAssembly receiver.

Applications:

- Exchanging cookies/tracking information.
- Extracting native data.



Covert Channel - Physical layer

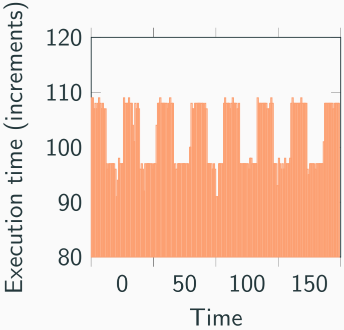


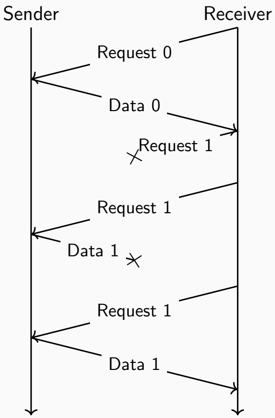
Figure 4: Transmitted square signal

- Sending a 1-bit by creating contention on Port 1
- Receiving bits by measuring execution times of Port 1 instructions
- Fixed bit duration of t_{bit}

Covert Channel - Data-Link layer

Data is separated in frames:

- Sequence number to handle synchronization
- Error-detecting code for bit insertion/deletion

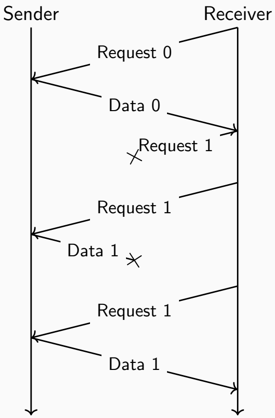


Covert Channel - Data-Link layer

Data is separated in frames:

- Sequence number to handle synchronization
- Error-detecting code for bit insertion/deletion

Simple request-to-send protocol to handle lost frames



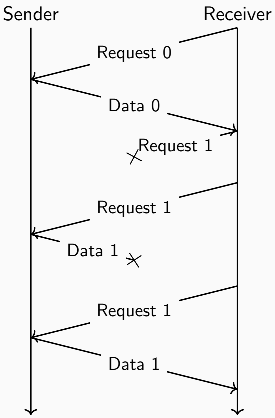
Covert Channel - Data-Link layer

Data is separated in frames:

- Sequence number to handle synchronization
- Error-detecting code for bit insertion/deletion

Simple request-to-send protocol to handle lost frames

Frame starts are detected using a density clustering algorithm.



Covert Channel - Evaluation

We found $t_{bit} = 1\,\text{ms}$ to be best.

On a quiet system, we obtain the following results:

- 200 bit/s of effective data (Best bandwidth for a web-based covert channel!)
- 6% of frame loss

Covert Channel - Evaluation

We found $t_{bit} = 1$ ms to be best.

On a quiet system, we obtain the following results:

- 200 bit/s of effective data (Best bandwidth for a web-based covert channel!)
- 6% of frame loss

We evaluated the covert channel with noise:

- `stress -m 2`: 170 bit/s
- `stress -m 3`: 25 bit/s

Covert Channel - Evaluation

We found $t_{bit} = 1\,\text{ms}$ to be best.

On a quiet system, we obtain the following results:

- 200 bit/s of effective data (Best bandwidth for a web-based covert channel!)
- 6% of frame loss

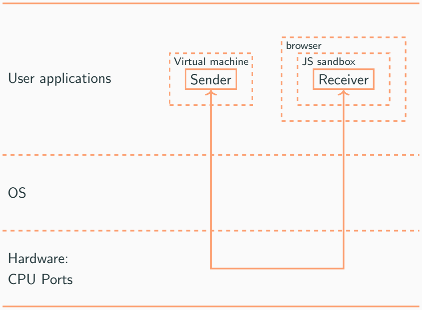
We evaluated the covert channel with noise:

- `stress -m 2`: 170 bit/s
- `stress -m 3`: 25 bit/s

Due to the same-core nature of port contention.

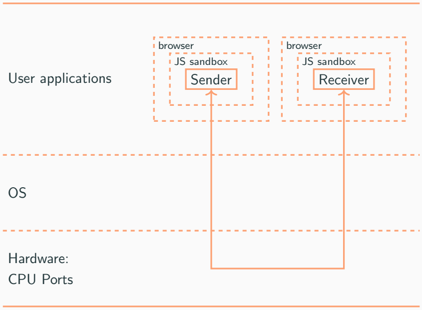
More Covert Channels

VM-to-host



80 bit/s bandwidth.

Cross-browser



200 bit/s bandwidth, across browsers!

Perspective on Web Port Contention

- First implementation of port contention in the browser.
- Fastest covert channel existing in the browser.
- High spatial resolution.
- Breaks the isolation of browser: cross-origin communication is possible, even through virtualized environments.

Sequential Port Contention

Port Contention and SMT

Port contention attacks rely on the attacker and victim sharing a hardware component. They are highly dependent on SMT.

Countermeasures to SMT attacks are starting to appear:

Port Contention and SMT

Port contention attacks rely on the attacker and victim sharing a hardware component.
They are highly dependent on SMT.

Countermeasures to SMT attacks are starting to appear:

- Disable SMT.

Port Contention and SMT

Port contention attacks rely on the attacker and victim sharing a hardware component. They are highly dependent on SMT.

Countermeasures to SMT attacks are starting to appear:

- Disable SMT.
- Dynamic Sharing.

Port Contention and SMT

Port contention attacks rely on the attacker and victim sharing a hardware component. They are highly dependent on SMT.

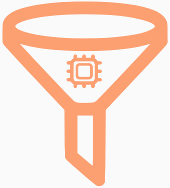
Countermeasures to SMT attacks are starting to appear:

- Disable SMT.
- Dynamic Sharing.

Can we create port contention without SMT?

Sequential Port Contention

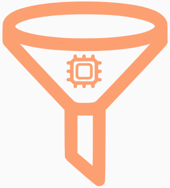
We introduce **Sequential Port Contention**.



Sequential Port Contention

We introduce **Sequential Port Contention**.

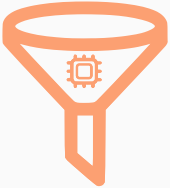
Exploit parallelism at the instruction level.



We introduce **Sequential Port Contention**.

Exploit parallelism at the instruction level.

Creates contention on ports and exploits it without SMT.



Port Contention Without SMT - Concept

(a) Grouped

(b) Interleaved

- Both experiments have the same number of instructions.
- Will they have a similar execution time?

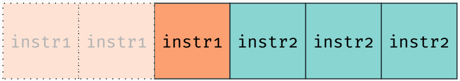
Port Contention Without SMT - Concept

Cycle 0

Port Contention Without SMT - Concept

Cycle 1

Port Contention Without SMT - Concept



Cycle 2

Port Contention Without SMT - Concept

Cycle 3

Port Contention Without SMT - Concept

Cycle 4

Port Contention Without SMT - Concept

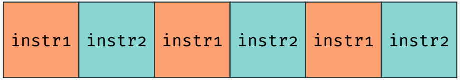
Cycle 5

Execution is never parallelized

Port Contention Without SMT - Concept

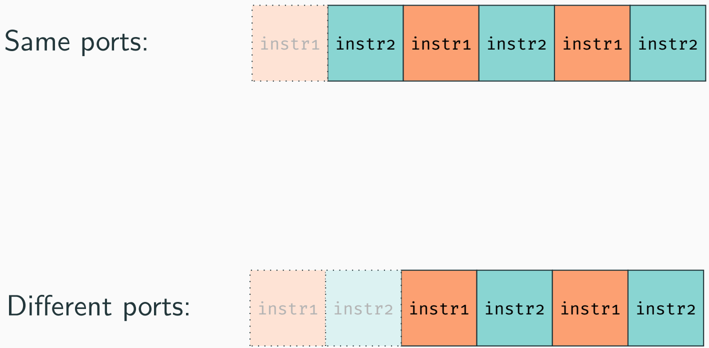
Same ports:

Different ports:



Cycle 0

Port Contention Without SMT - Concept



Cycle 1

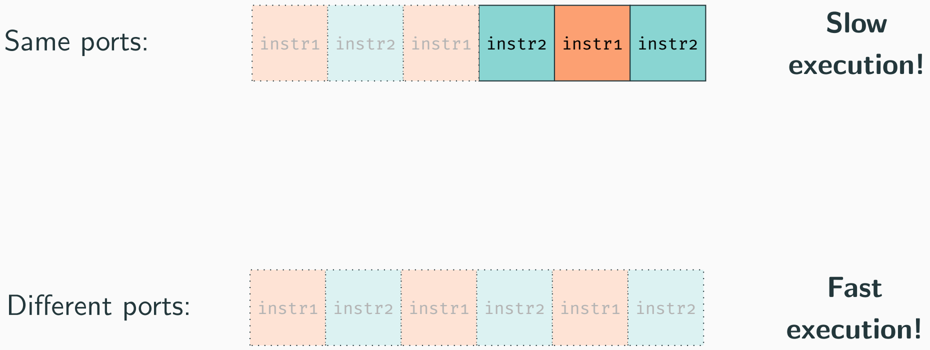
Port Contention Without SMT - Concept

Same ports:

Different ports:

Cycle 2

Port Contention Without SMT - Concept



Cycle 3

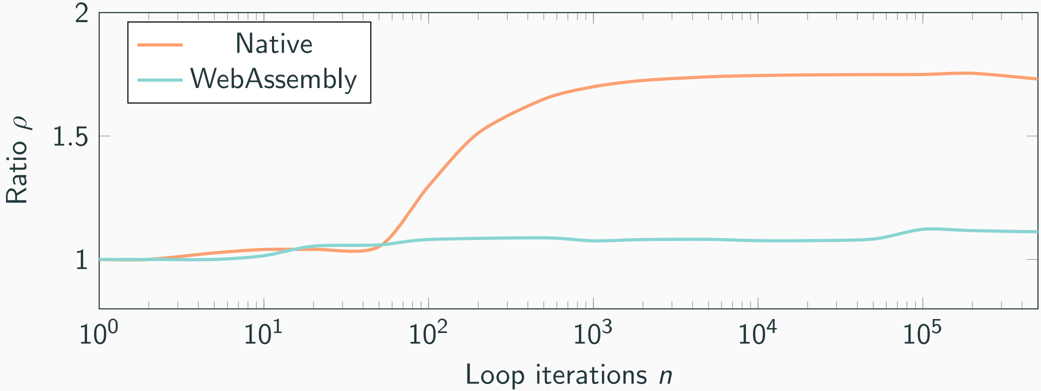


Figure 6: $\rho_{grouped\,/\,interleaved}$.

Application to Fingerprinting - Idea

- CPU generations bring changes to the microarchitecture.

Application to Fingerprinting - Idea

- CPU generations bring changes to the microarchitecture.
- Instructions can have **different port usages** between generations.

Application to Fingerprinting - Idea

- CPU generations bring changes to the microarchitecture.
- Instructions can have **different port usages** between generations.
- If we can determine the port usage of these instructions **from the web**, we can guess the generation!

Application to Fingerprinting - Idea

- CPU generations bring changes to the microarchitecture.
- Instructions can have **different port usages** between generations.
- If we can determine the port usage of these instructions **from the web**, we can guess the generation!
- Consolidate software attributes for fingerprinting.

Application to Fingerprinting - Framework

We need to find **distinguishers**, *i.e.*, pairs of instructions that:

- Exhibit different contention on different generations;
- Exhibit similar contention on different CPUs of the same generation.

Application to Fingerprinting - Framework

We need to find **distinguishers**, *i.e.*, pairs of instructions that:

- Exhibit different contention on different generations;
- Exhibit similar contention on different CPUs of the same generation.

Problem: We do not know how our WebAssembly instructions are translated.

Application to Fingerprinting - Framework

We need to find **distinguishers**, *i.e.*, pairs of instructions that:

- Exhibit different contention on different generations;
- Exhibit similar contention on different CPUs of the same generation.

Problem: We do not know how our WebAssembly instructions are translated.

We extended **PC-detector** to test 458 pairs of instructions for distinguishers, and found **30**.

Application to Fingerprinting - Classifier, Training, and Evaluation

- Once we have these distinguishers, we create **generation fingerprints**, *i.e.*, the behavior of the distinguishers for a given generation.

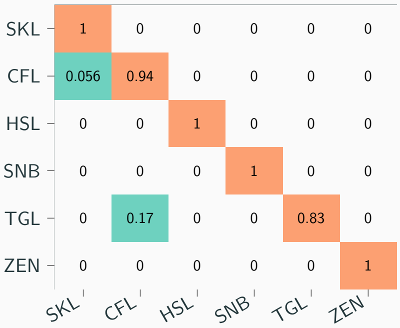
Application to Fingerprinting - Classifier, Training, and Evaluation

- Once we have these distinguishers, we create **generation fingerprints**, *i.e.*, the behavior of the distinguishers for a given generation.
- We use it to train a k-NN model to classify unknown CPUs.

Application to Fingerprinting - Classifier, Training, and Evaluation

- Once we have these distinguishers, we create **generation fingerprints**, *i.e.*, the behavior of the distinguishers for a given generation.
- We use it to train a k-NN model to classify unknown CPUs.
- We created a website to get these fingerprints: `https://fp-cpu-gen.github.io/fp-cpu-gen` Feel free to try and send us results!

Application to Fingerprinting - Results



- Evaluation on **50** different CPUs, spanning **13 generations**.
- Includes Intel CPUs and AMD.
- **92%** accuracy.
- Highly stable and resistant to noise.

Perspective on Sequential Port Contention

- Threat surface extension for port contention.

Perspective on Sequential Port Contention

- Threat surface extension for port contention.
- Applications to browser fingerprinting.

Perspective on Sequential Port Contention

- Threat surface extension for port contention.
- Applications to browser fingerprinting.
- Highly resistant to noise.

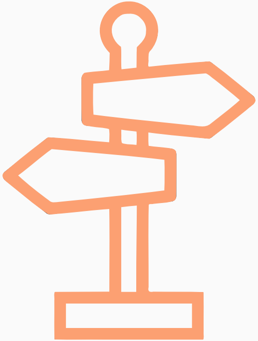
Perspective on Sequential Port Contention

- Threat surface extension for port contention.
- Applications to browser fingerprinting.
- Highly resistant to noise.
- Maybe other SMT attacks can be leveraged with instruction-level parallelism?

Conclusion and Future Work

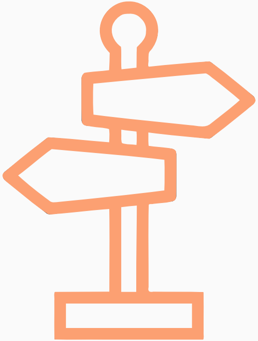
Future Work

- We have many new side channels to discover.



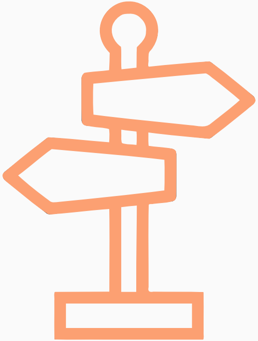
Future Work

- We have many new side channels to discover.
  - Exploit already existing native side channels.
  - Change the attack paradigm to discover new threats.



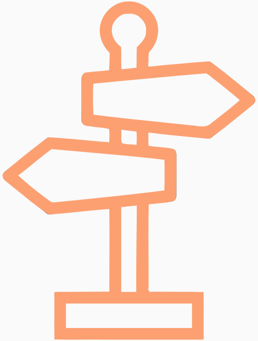
Future Work

- We have many new side channels to discover.
  - Exploit already existing native side channels.
  - Change the attack paradigm to discover new threats.
- Find new directions for browser countermeasures.



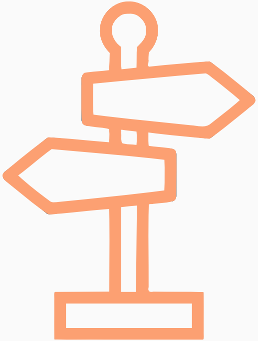
Future Work

- We have many new side channels to discover.
  - Exploit already existing native side channels.
  - Change the attack paradigm to discover new threats.
- Find new directions for browser countermeasures.
  - Countermeasures at the application level.
  - Enforcing detection on top of prevention.



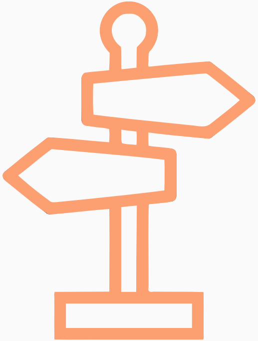
Future Work

- We have many new side channels to discover.
  - Exploit already existing native side channels.
  - Change the attack paradigm to discover new threats.
- Find new directions for browser countermeasures.
  - Countermeasures at the application level.
  - Enforcing detection on top of prevention.
- Bringing automation to the browser.



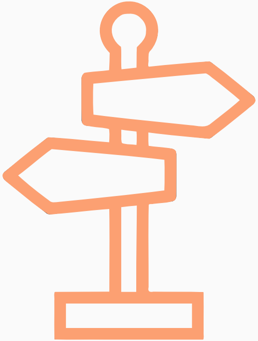
Future Work

- We have many new side channels to discover.
  - Exploit already existing native side channels.
  - Change the attack paradigm to discover new threats.
- Find new directions for browser countermeasures.
  - Countermeasures at the application level.
  - Enforcing detection on top of prevention.
- Bringing automation to the browser.
  - Automated side channel discovery.
  - Systematized understanding of JavaScript engine.



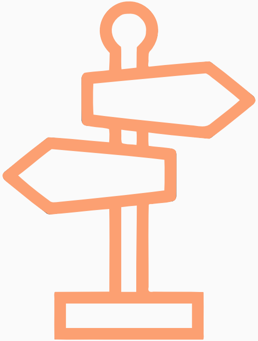
Future Work

- We have many new side channels to discover.
  - Exploit already existing native side channels.
  - Change the attack paradigm to discover new threats.
- Find new directions for browser countermeasures.
  - Countermeasures at the application level.
  - Enforcing detection on top of prevention.
- Bringing automation to the browser.
  - Automated side channel discovery.
  - Systematized understanding of JavaScript engine.
- Hardware browser fingerprinting is promising.



Future Work

- We have many new side channels to discover.
  - Exploit already existing native side channels.
  - Change the attack paradigm to discover new threats.
- Find new directions for browser countermeasures.
  - Countermeasures at the application level.
  - Enforcing detection on top of prevention.
- Bringing automation to the browser.
  - Automated side channel discovery.
  - Systematized understanding of JavaScript engine.
- Hardware browser fingerprinting is promising.
  - As a complement to software fingerprinting.
  - Exploiting imperfection can lead to unique fingerprints.



Contributions

ESORICS 2022 Port Contention Without SMT.
Thomas Rokicki, Clémentine Maurice, Michael Schwarz.

AsiaCCS 2022 Port Contention Goes Portable.
Thomas Rokicki, Clémentine Maurice, Marina Botvinnik, Yossi Oren.

EuroS&P 2021 In Search Of Lost Time.
Thomas Rokicki, Clémentine Maurice, Pierre Laperdrix.

Thank you for your attention!