Port Contention Goes Portable: Port Contention Side-Channels in Web Browsers

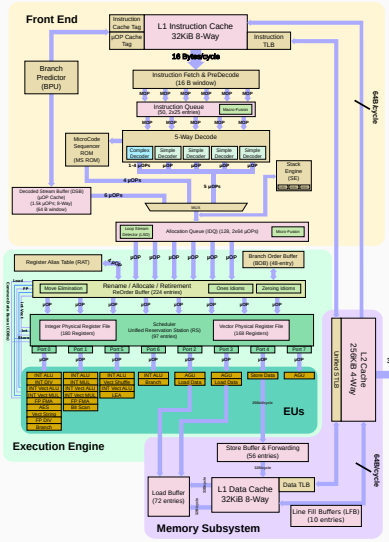
Thomas Rokicki - Univ Rennes, CNRS, IRISA
Clémentine Maurice - Univ Lille, CNRS, Inria
Marina Botvinnik - Ben-Gurion University of the Negev
Yossi Oren - Ben-Gurion University of the Negev

AsiaCCS 2022 - June 2nd 2022

Background: Microarchitectural attacks

Exploit subtle timing differences caused by the microarchitecture.

Cache attacks are the most famous, but most microarchitectural optimizations are targeted.

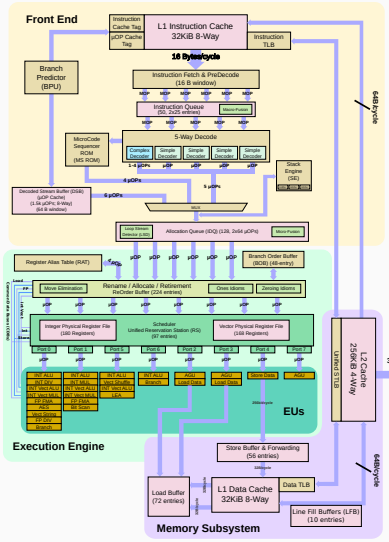


Background: Microarchitectural attacks

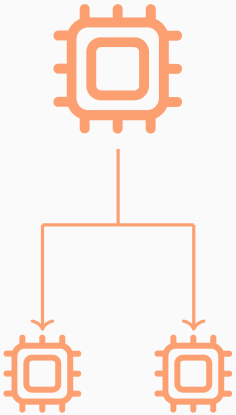
Exploit subtle timing differences caused by the microarchitecture.

Cache attacks are the most famous, but most microarchitectural optimizations are targeted.

Here: CPU Ports



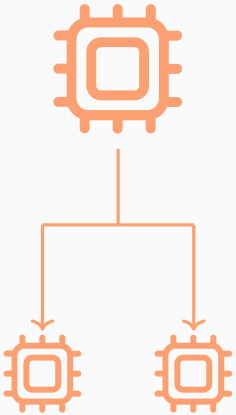
Background: Hyperthreading



Simultaneous computation technology of Intel.

- Physical cores are shared in several (often 2) logical cores
- Abstraction at the OS level

Background: Hyperthreading

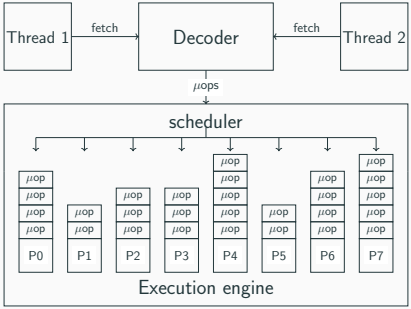


Simultaneous computation technology of Intel.

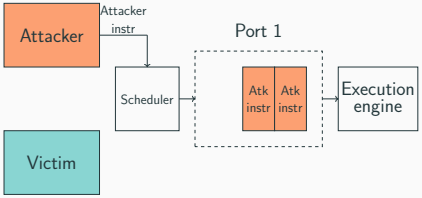
- Physical cores are shared in several (often 2) logical cores
- Abstraction at the OS level
- **Hardware resources are shared between logical cores**

Background: Execution pipeline

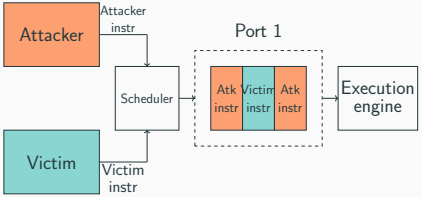
- Instructions are decomposed in micro-operations (μops) to optimize Out-of-Order computation
- The decomposition of instructions into μops is deterministic
- μops are dispatched to specialized execution units through **CPU ports**



Background: Port contention[1]



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



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

[1] Aldaya et al. , Port Contention for Fun and Profit, S&P, 2019

Port contention prerequisites

- Attacker code must run on the victim's hardware
- Attacker and victim must be on the **same physical core**
- Attacker must have access to high-resolution timers

Background: JavaScript and WebAssembly

- Runs code on the **client's hardware**.
- JIT compilation.
- Sandboxed

- Runs code on the **client's hardware**
- Compiled from another language
- Sandboxed
- Smaller, more atomic instructions

Threat model

Client side languages run on the client's hardware.

We can run port contention attacks on the victim's hardware

Threat model

Client side languages run on the client's hardware.

We can run port contention attacks on the victim's hardware

Malicious website or advertisement

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

To prevent timing attacks, browsers removed access to JavaScript high-resolution timers, and added jitter to the measurements.

C2 - high-resolution timers

To prevent timing attacks, browsers removed access to JavaScript high-resolution timers, and added jitter to the measurements.

Build auxiliary timers with a resolution of several nanoseconds[2].

[2]Schwarz et al. , Fantastic timers and where to find them, Financial Cryptography, 2017

Rokicki et al. , Sok: In search of lost time: A review of javascript timers in browsers, EuroS&P, 2021

C2 - high-resolution timers

To prevent timing attacks, browsers removed access to JavaScript high-resolution timers, and added jitter to the measurements.

Build auxiliary timers with a resolution of several nanoseconds[2].

For most experiments in this paper, we use a timer based on `SharedArrayBuffer`.

[2] Schwarz et al. , Fantastic timers and where to find them, Financial Cryptography, 2017

Rokicki et al. , Sok: In search of lost time: A review of javascript timers in browsers, EuroS&P, 2021

PC-Detector

We don't know 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

PC-Detector is also composed of a native *spammer* and a web *tester*.

For each WebAssembly instruction, we run the following experiments:

Control : The web script runs alone in the browser

Contention on Port x : The web script runs while the native component repeatedly calls an instruction creating contention on Port x

We test all instructions with ports 0,1,(2,3),5 and 6.

PC-Detector - Results

We tested over 200 different instructions.

- 80 instructions creating contention
- 4 ports: 0, 1, 5 and 6
- Best instruction is `i64.rem_u`

Side-Channel Artificial Example - Description

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

Side-Channel Artificial Example - Description

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

Secret is 0!

Side-Channel Artificial Example - Description

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

Secret is 1!

Side-Channel Artificial Example - Results

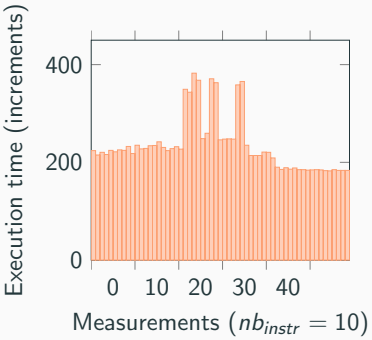


Figure 1: Secret key: 1101001.

- Able to detect 1024 native instructions in a single trace

Side-Channel Artificial Example - Results

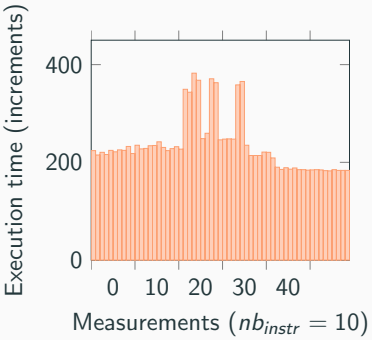


Figure 1: Secret key: 1101001.

- 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

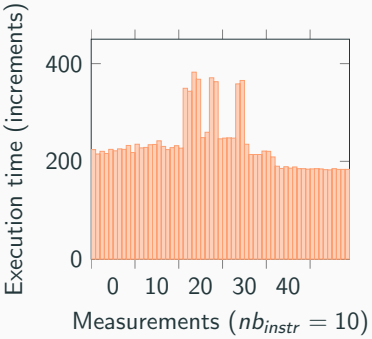


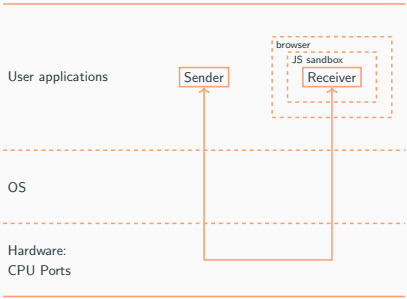
Figure 1: Secret key: 1101001.

- 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



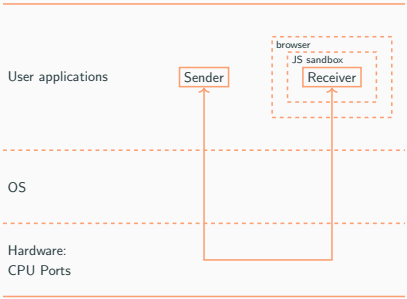
Covert Channel

Composed of two components:

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

Results:

- 200 bit/s
- 6% frame loss



Covert Channel

Composed of two components:

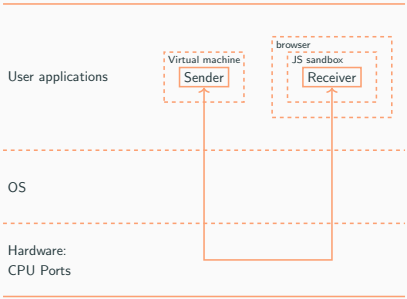
- **Native:** C/x86 sender
- **Web:** JavaScript/WebAssembly receiver

Results:

- 200 bit/s
- 6% frame loss

Other settings:

- Host-to-VM



Covert Channel

Composed of two components:

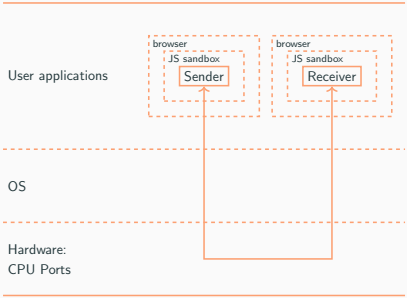
- **Native:** C/x86 sender
- **Web:** JavaScript/WebAssembly receiver

Results:

- 200 bit/s
- 6% frame loss

Other settings:

- Host-to-VM
- Cross Browser



Countermeasures

Hardware: Disable SMT, dynamic SMT

OS: Port-independent code, port-aware scheduler

Browser: Removing high-resolution timers, process isolation.

Conclusion

- 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

Questions?

Contact me here: thomas.rokicki@irisa.fr

Feel free to read the paper for more technical details!

Find the code here: https://github.com/MIAOUS-group/web-port-contention