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

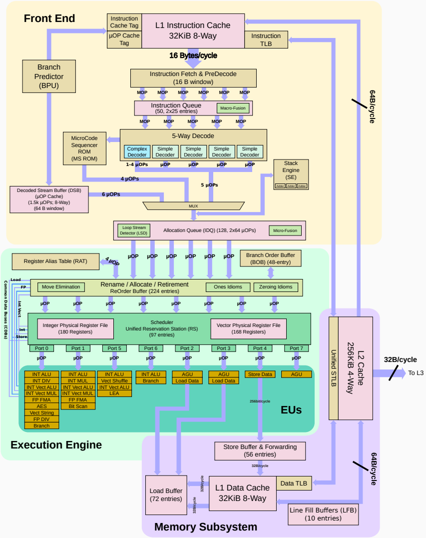
Thomas Rokicki - Univ Rennes, CNRS, IRISA
Clémentine Maurice - Univ Lille, CNRS, Inria
Marina Botvinnik - Ben-Gurion University of the Negev
Yossi Oren - Ben-Gurion University of the Negev

SOSYSEC - 13/05/2022

Background: Microarchitectural attacks

Exploit subtle timing differences caused by the microarchitecture.

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

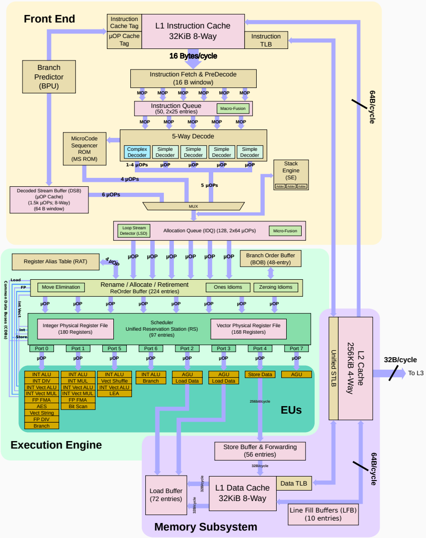


Background: Microarchitectural attacks

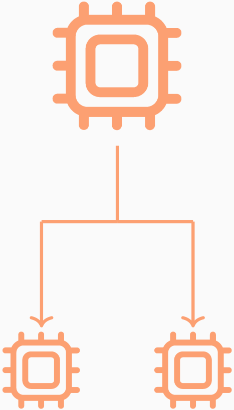
Exploit subtle timing differences caused by the microarchitecture.

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

Here: CPU Ports



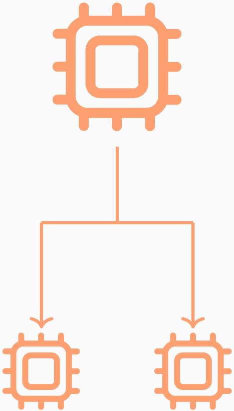
Background: Hyperthreading



Simultaneous computation technology of Intel.

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

Background: Hyperthreading

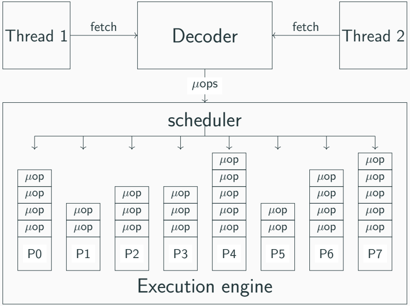


Simultaneous computation technology of Intel.

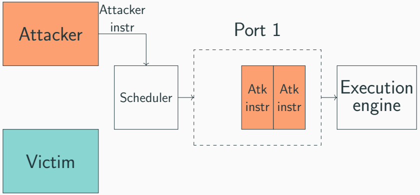
- Physical cores are shared in several (often 2) logical cores
- Abstraction at the OS level
- **Hardware resources are shared between logical cores**

Background: Execution pipeline

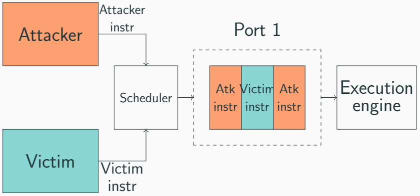
- Instructions are decomposed in micro-operations (μops) to optimize Out-of-Order computation
- The decomposition of instructions into μops is deterministic
- μops are dispatched to specialized execution units through **CPU ports**



Background: Port contention



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



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

State of the Art: Port contention attacks

Aldaya et al. [1] introduced the first attack with port contention, natively attacking OpenSSL's TLS and stealing private keys

Port contention was also used as a side-channel to mount speculative execution attacks[2].

[1] Aldaya et al. , Port Contention for Fun and Profit, S&P, 2019
[2] Bhattacharyya et al. , Smotherspectre: Exploiting speculative execution through port contention, CCS, 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

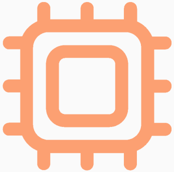
Client side languages run on the client's hardware.

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

Malicious website or advertisement

C1 - Core control

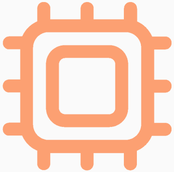
JavaScript does not have core control



C1 - Core control

JavaScript does not have core control

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

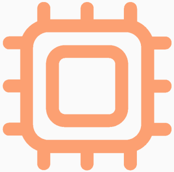


C1 - Core control

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.

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[3].

[3] 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[3].

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

[3] 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

Proof of Concept

The proof of concept is composed of two components:

Native : A C script that runs `TZCNT` x86 instructions (P1 μop) on all physical cores

Web : A WebAssembly/JavaScript script repeatedly calling the `i64.ctz` instruction and timing the execution

We run two experiments:

Control : The web script runs alone in the browser

Contention : Both web and script components are executed together

PoC - Results

Figure 1: Port 1 contention experiment on `i64.ctz` for 1 000 000 instructions.

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 - Metrics

Some instructions create "better" contention than others, *i.e.*, the two distributions are more distinguishable. We need metrics to evaluate them.

PC-Detector - Metrics

Some instructions create "better" contention than others, *i.e.*, the two distributions are more distinguishable. We need metrics to evaluate them.

Error rate : Given a threshold, ratio of control values computed as contention values and vice versa

PC-Detector - Metrics

Some instructions create "better" contention than others, *i.e.*, the two distributions are more distinguishable. We need metrics to evaluate them.

Error rate : Given a threshold, ratio of control values computed as contention values and vice versa

Cohen's Distance : Distance between the two distributions divided by their spread.

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.

Victim

`secret == 0` | `secret == 1`

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

Contention on Port 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 sender 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 sender 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

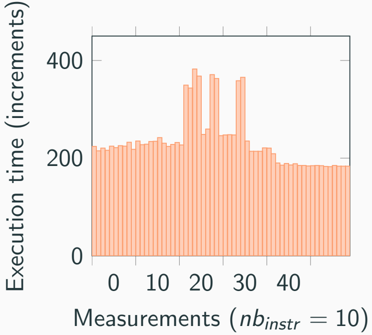


Figure 2: Secret key: 1101001.

- Able to detect 1024 native instructions in a single trace

Side-Channel Artificial Example - Results

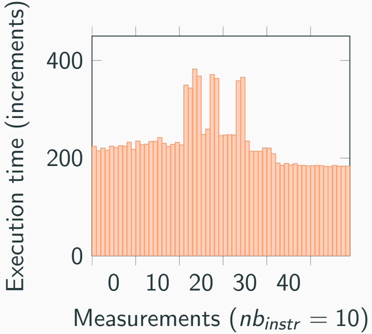


Figure 2: Secret key: 1101001.

- 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

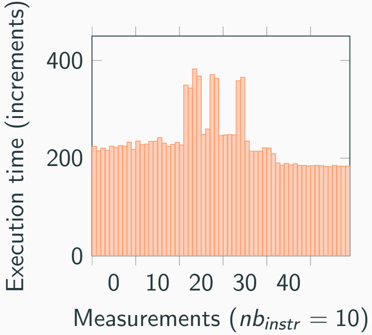


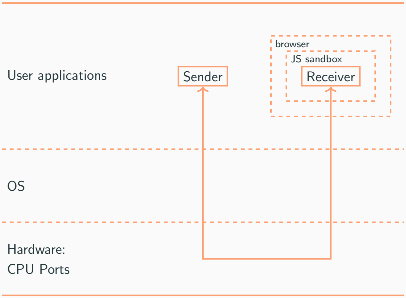
Figure 2: Secret key: 1101001.

- 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 - Threat Model

Composed of two components:

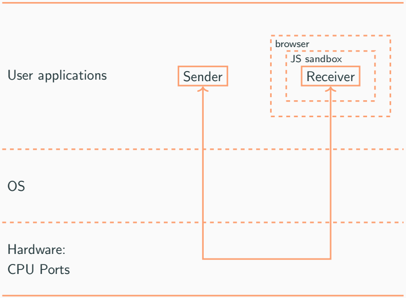
- **Native:** C/x86 sender
- **Web:** JavaScript/WebAssembly receiver



Covert Channel - Threat Model

Composed of two components:

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



Example threats:

- Extracting sensible data
- Exchanging cookies and tracking
- Monitoring

Covert Channel - Physical layer

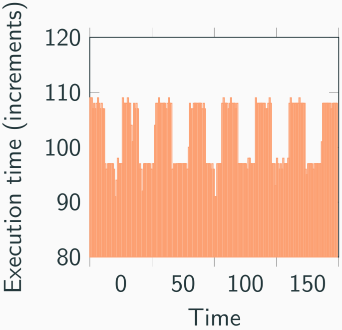


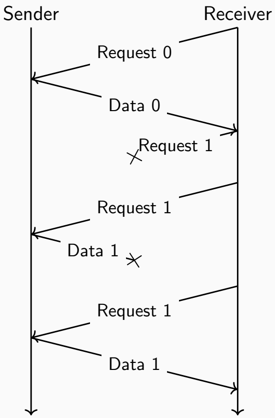
Figure 3: Transmitted square signal

- Sending a 1-bit by creating contention on Port 1
- Receiving bits by measuring execution time 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

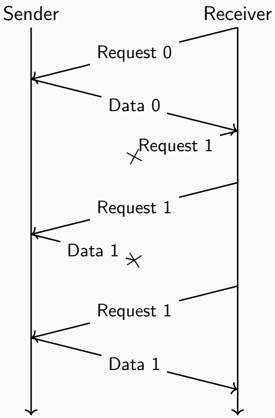


Covert Channel - Data-Link layer

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



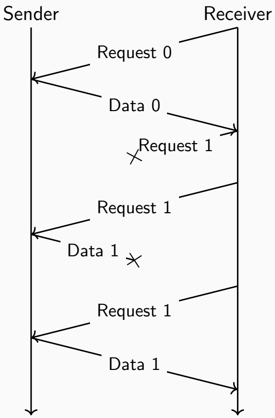
Covert Channel - Data-Link layer

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

Frames start 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 $t_{bit} = 1\,\text{ms}$ 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

Covert Channel - Evaluation

We found $t_{bit} = 1\,\mathrm{ms}$ 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

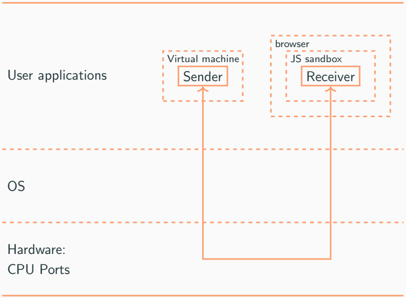
Due to the same-core nature of port contention.

Covert Channel - VM-to-host

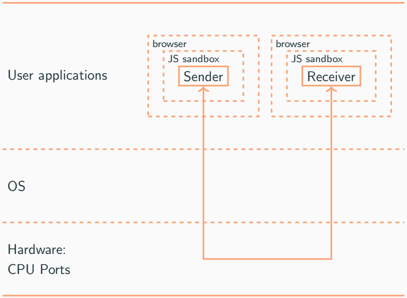
VM-to-host scenario

No control of real OS cores on the native sender.

80 bit/s bandwidth.



Covert Channel - Cross-Browser



Browser-to-browser scenario.

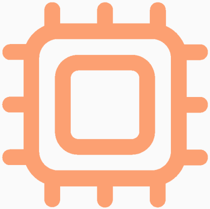
No control of cores, everything is handled by multithreading.

200 bit/s bandwidth at physical layer.

Even works with different browsers!

Countermeasures - Hardware Level

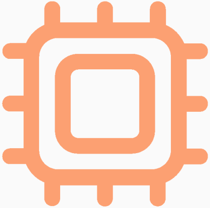
Disabling SMT: High performance cost (15%)



Disabling SMT: High performance cost (15%)

Dynamic sharing of resources:

- Temporal sharing: At a given time, a resource is available to only one thread[4]
- Adaptative sharing: When computing critical information, resources are not shared[5]



[4] Townley and Ponomarev, SMT-COP: defeating side-channel attacks on execution units in SMT processors, PACT, 2019.

[5] Mohammadkazem et al. , Secsmt: Securing SMT processors against contention-based covert channel, Usenix, 2022

Countermeasures - OS and software level

- Static / dynamic analysis

Countermeasures - OS and software level

- Static / dynamic analysis
- Port-independent code

Countermeasures - OS and software level

- Static / dynamic analysis
- Port-independent code
- Port-aware scheduler

Countermeasures - Browser level

- Remove high-resolution timers
- Grant more isolation to processes

Countermeasures - Browser level

- Remove high-resolution timers
- Grant more isolation to processes

Countermeasures are not really suited for browsers.

Future work

- Implement a cryptographic side-channel attack
- Study in more details the translation of web-to-native code
- Find other vectors of contention, automatically or across cores

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`