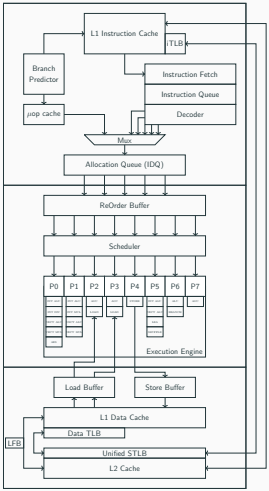
CPU Port Contention Without SMT

Thomas Rokicki - Univ Rennes, CNRS, IRISA
Clémentine Maurice - Univ Lille, CNRS, Inria
Michael Schwarz - CISPA Helmholtz Center for Information Security
ESORICS 2022 - 24/09/2022

Background: Microarchitectural attacks

Exploit subtle timing differences caused by the microarchitecture.

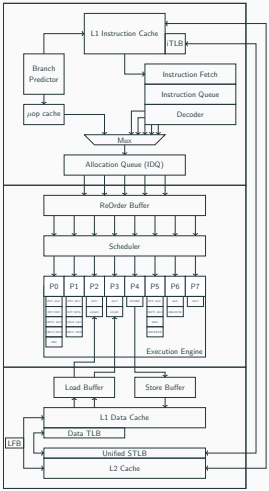
Cache attacks are the most famous, but most microarchitectural optimizations are targeted.



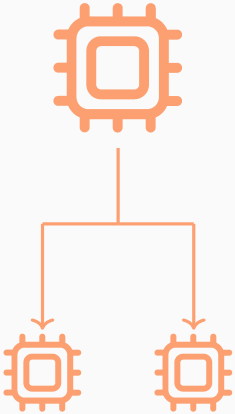
Background: Microarchitectural attacks

Exploit subtle timing differences caused by the microarchitecture.
Cache attacks are the most famous, but most microarchitectural optimizations are targeted.

Here: CPU Ports



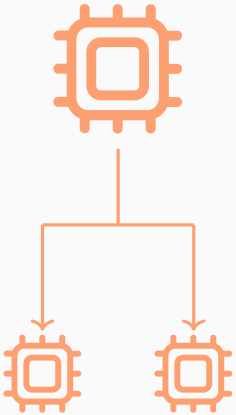
Background: SMT



Simultaneous computation (Also called Hyper-Threading).

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

Background: SMT

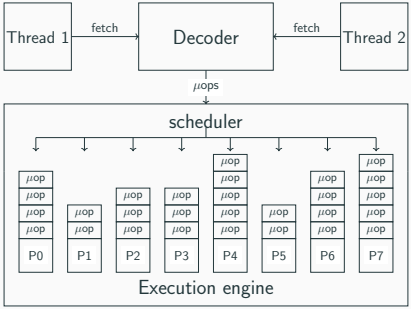


Simultaneous computation (Also called Hyper-Threading).

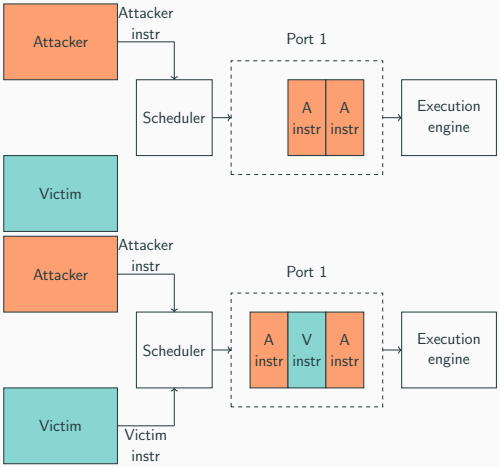
- Physical cores are shared in several (often 2) logical cores.
- Abstraction at the OS level.
- **Hardware resources are shared between logical cores.**

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]



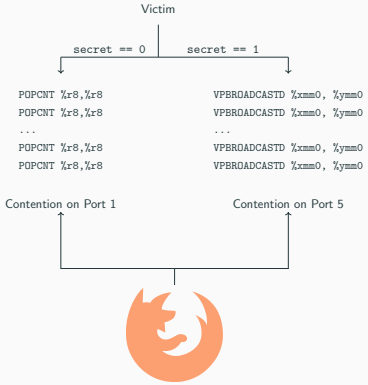
No Contention All the attacker instructions are executed in a row, **fast execution time**.

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

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

Background: Port contention in web browsers[2]

- Port contention is also implementable in browsers with WebAssembly.
- Huge threat surface but restricted environment.
- We proposed a framework to determine the port usage of WebAssembly instructions.
- Spatial resolution on par with Prime+Probe.



[2] Rokicki et al. , Port contention goes portable, AsiaCCS, 2021

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 (RedHat)

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 (RedHat) Dynamic Sharing [3].

[3] Taram et al. , SecSMT: Securing SMT Processors against Contention-Based Covert Channels, USENIX 22

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 (RedHat) Dynamic Sharing [3].

Can we create port contention without SMT?

[3] Taram et al. , SecSMT: Securing SMT Processors against Contention-Based Covert Channels, USENIX 22

Sequential Port Contention

We introduce **Sequential Port Contention**.

Sequential Port Contention

We introduce **Sequential Port Contention**.

Exploit parallelism at instruction level.

Sequential Port Contention

We introduce **Sequential Port Contention**.

Exploit parallelism at instruction level.

Creates contention on ports and exploits it without SMT.

Port Contention Without SMT - Concept

(a) Grouped

(b) Interleaved

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

Instructions use the same ports

Instructions use different ports

Cycle 0

Port Contention Without SMT - Concept

Instructions use the same ports

Instructions use different ports

Cycle 1

Port Contention Without SMT - Concept

Instructions use the same ports

Instructions use different ports

Cycle 2

Port Contention Without SMT - Concept

Instructions use the same ports

Instructions use different ports

Cycle 3

Port Contention Without SMT - Concept

Instructions use the same ports

The instructions create contention at the port level:

slower execution

Instructions use different ports

The execution is paralellized at port level:

faster execution

Results - Execution time

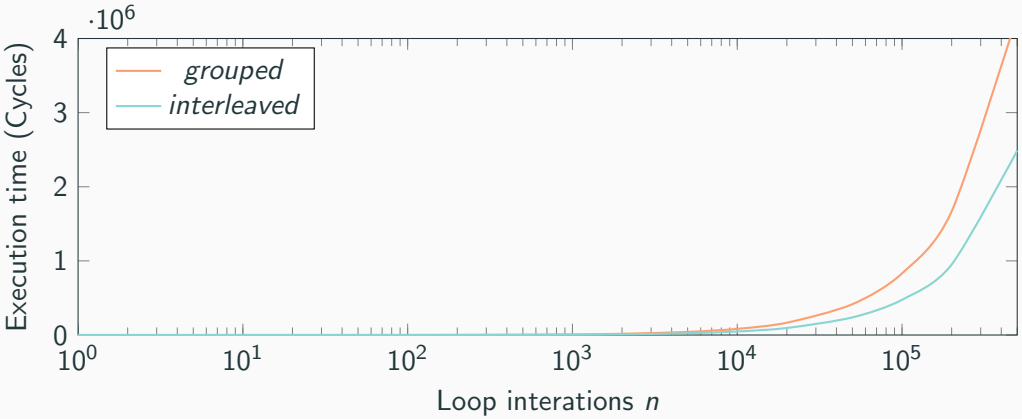


Figure 8: Execution time for grouped vs. interleaved with different ports.

Results - Ratio of Execution time

We compute the ratio $\rho_{grouped\,/\,interleaved}$.

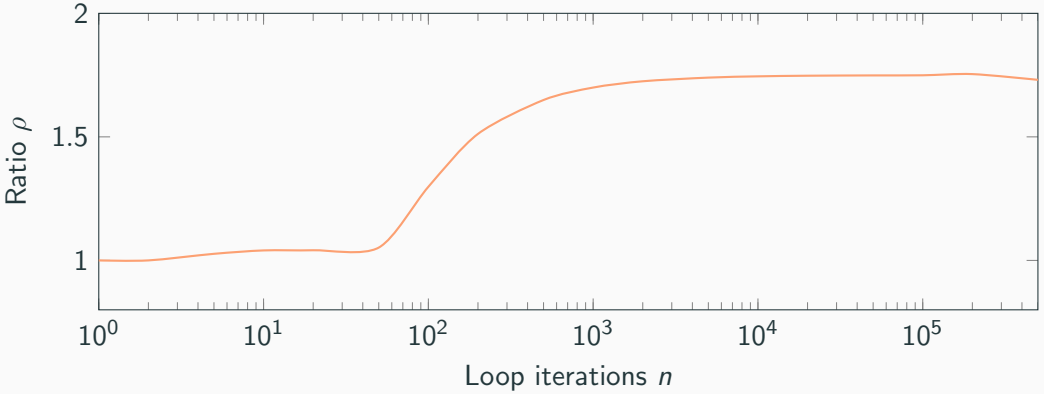


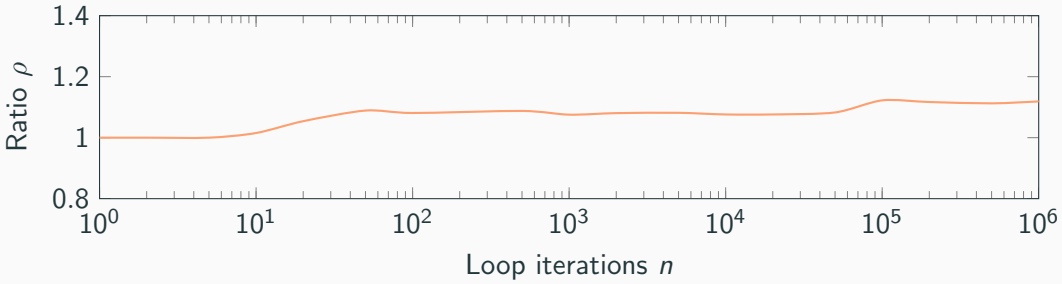
Figure 9: $\rho_{grouped\,/\,interleaved}$.

Web

Same idea but in WebAssembly.

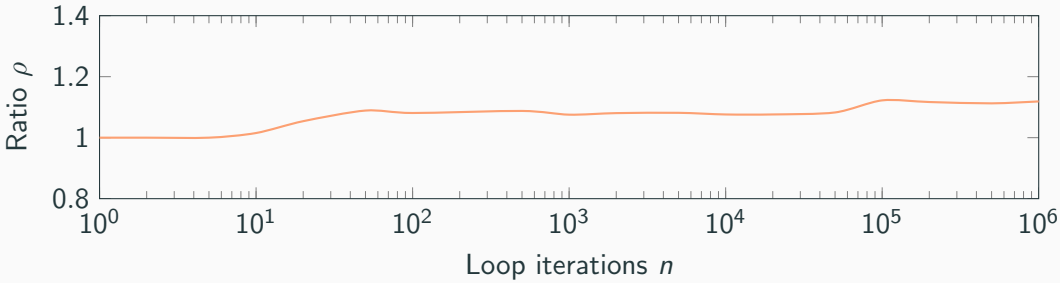
Web

Same idea but in WebAssembly.



Web

Same idea but in WebAssembly.



Ratio is lower due to lowest-resolution timers / less control on port usage.

Allows for way more portability!

Application to fingerprint - Idea

- CPU generations bring changes to the microarchitecture.

Application to fingerprint - Idea

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

Application to fingerprint - 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 fingerprint - 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[4].

[4] Trampert et al. , Browser-based CPU Fingerprinting, Esorics 2022

Application to fingerprint - 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 fingerprint - 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 fingerprint - Framework

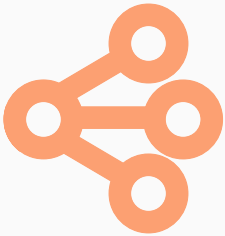
We need to find **distinguishers**, *i.e.*, pairs of instructions that:

- 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 built a **framework**, testing 458 pairs of instructions for distinguishers and found **30**.

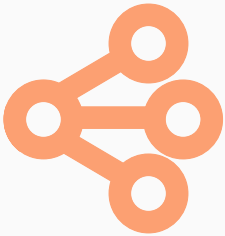
Application to fingerprint - Classifier, Training and Evaluation

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



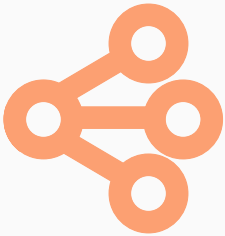
Application to fingerprint - Classifier, Training and Evaluation

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

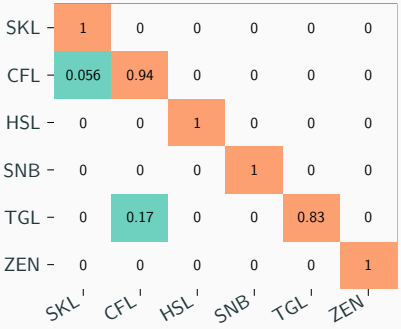


Application to fingerprint - Classifier, Training and Evaluation

- Once we have these distinguishers, we create *generation fingerprints*, ie 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 fingerprint - Results

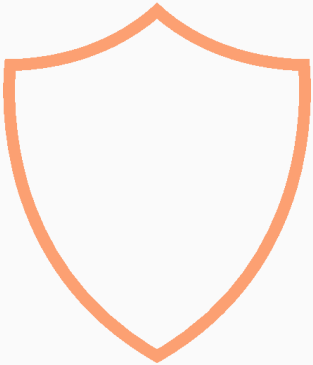


- Evaluation on 50 different CPUs, spanning 13 generations.
- Includes Intel CPUs and AMD.
- 92% accuracy.
- Highly stable and resistant to noise.

Countermeasures

Software Diversification : JavaScript engine can reorder code while keeping functionality or add fences to the output.

Performance Counters Detection : We can detect when sequential port contention occurs by measuring backend-bound execution.



Conclusion

- Threat surface extension for port contention.

Conclusion

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

Conclusion

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

Conclusion

- 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?

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/port-contention-without-smt

Credits

Credits for images:

- Vecteezy.com
- Veryicon.com