



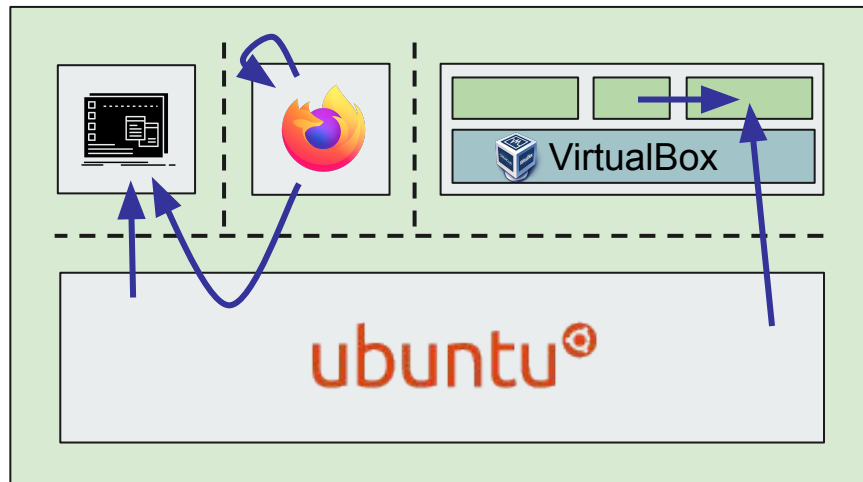
# Performance Evolution of Mitigating Transient Execution Attacks

Jonathan Behrens, Adam Belay, and M. Frans Kaashoek

MIT CSAIL

# Transient execution attacks break isolation

- Enable programs to access information they shouldn't be able to.
- Different cases:
  - a. Program reading information from the OS
  - b. Program accessing information from another program
  - c. Leaking information between websites visited in the same web browser



# There are many different transient execution attacks

## FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution

Jo Van Bulck<sup>1</sup>, Marina Minkin<sup>2</sup>, Ofir Weiss<sup>3</sup>, Daniel Genkin<sup>1</sup>, Baris Kasikci<sup>3</sup>, Frank Piessens<sup>1</sup>, Mark Silberstein<sup>2</sup>, Thomas F. Wenisch<sup>3</sup>, Yuval Yarom<sup>1</sup>, and Raoul Strackx<sup>1</sup>  
<sup>1</sup>imec-DistriNet, KU Leuven, <sup>2</sup>Technion, <sup>3</sup>University of Michigan, <sup>4</sup>University of Adelaide

Trusting enclaves with a minimal Trusted Computing Base (TCB) that includes only the processor and memory. Enclave-private CPU and memory are inaccessible to the code running in the enclave, accessible to the code running in the host.

## ZombieLoad: Cross-Privilege-Boundary Data Sampling

Michael Schwarz<sup>1</sup>, Moritz Lipp<sup>1</sup>, Graz University of Technology, michael.schwarz@iaik.tugraz.at  
Jo Van Bulck<sup>1</sup>, Julian Steinhilber<sup>1</sup>, imec-DistriNet, KU Leuven, jo.vanbulck@cs.kuleuven.be

## ABSTRACT

In early 2018, Meltdown first showed how to read arbitrary kernel memory from user space by exploiting side-channel instructions. While this was a significant breakthrough, it did not

K

## CROSSTALK: Speculative Data Leaks Across Cores Are Real

Yanyan Ragab<sup>1</sup>, Alyssa Milburn<sup>1</sup>, Kaveh Razavi<sup>1</sup>, Herbert Bos<sup>2</sup>, and Cristiano Giuffrida<sup>1</sup>

<sup>1</sup>Department of Computer Science  
Vrije Universiteit Amsterdam, The Netherlands  
y.ragab@vu.nl, a.milburn@vu.nl

## Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution

Revision 1.0 (August 14, 2018)

Jo Van Bulck<sup>1</sup>, Marina Minkin<sup>2</sup>, Daniel Genkin<sup>1</sup>, Baris Kasikci<sup>3</sup>, Mark Silberstein<sup>2</sup>, Raoul Strackx<sup>1</sup>, Thomas F. Wenisch<sup>3</sup>, and Ofir Weiss<sup>3</sup>  
<sup>1</sup>imec-DistriNet, KU Leuven, <sup>2</sup>Technion, <sup>3</sup>University of Michigan, <sup>4</sup>University of Adelaide

## LazyFP: Leaking FPU Register State using Microarchitectural Side-Channels

## Spectre Attacks: Exploiting Speculative Execution

Paul Kocher<sup>1</sup>, Jann Horn<sup>2</sup>, Anders Fogh<sup>3</sup>, Daniel Genkin<sup>1</sup>, Werner Haas<sup>4</sup>, Mike Hamburg<sup>1</sup>, Moritz Lipp<sup>5</sup>, Prescher<sup>6</sup>, Michael Schwarz<sup>7</sup>, Yuval Yarom<sup>8</sup>  
<sup>1</sup>Google Project Zero, <sup>2</sup>University of Pennsylvania and University of Maryland, <sup>3</sup>University of Technology, <sup>4</sup>Cyberus Technology, <sup>5</sup>Arch Division, <sup>6</sup>University of Adelaide and Data61

leverage hardware vulnerabilities to leak sensitive information and sample, if the attacks of the last

## CacheOut: Leaking Data on Intel CPUs via Cache Evictions

Stephan van Schaik<sup>1</sup>  
University of Michigan  
stephvs@umich.edu

Marina Minkin<sup>2</sup>  
University of Michigan  
minkin@umich.edu

Andrew Kwong<sup>3</sup>  
University of Michigan  
ankwong@umich.edu

Yuval Yarom<sup>4</sup>  
University of Adelaide and Data61  
y.yarom@adelaide.edu.au

## Meltdown: Reading Kernel Memory from User Space

Michael Schwarz<sup>1</sup>, Daniel Gruss<sup>1</sup>, Thomas Prescher<sup>2</sup>, Werner Haas<sup>3</sup>, Anders Fogh<sup>4</sup>, Jann Horn<sup>5</sup>, Stefan Mangard<sup>1</sup>, Daniel Genkin<sup>6</sup>, Yuval Yarom<sup>7</sup>, Mike Hamburg<sup>8</sup>, Data Advanced Analytics, <sup>2</sup>Cyberus Technology GmbH, <sup>3</sup>Google Project Zero, <sup>4</sup>University of Michigan, <sup>5</sup>University of Adelaide and Data61, <sup>6</sup>Rambus, <sup>7</sup>Cryptography Research Division

## Hijacking Transient Execution to Perform Microarchitectural Load Value Injection

Daniel Genkin<sup>1</sup>, Michael Schwarz<sup>2</sup>, Moritz Lipp<sup>3</sup>, Marina Minkin<sup>4</sup>, Yuval Yarom<sup>5</sup>, Berk Sunar<sup>6</sup>, Daniel Gruss<sup>1</sup>, and Frank Piessens<sup>7</sup>  
<sup>1</sup>Leuven, <sup>2</sup>Worcester Polytechnic Institute, <sup>3</sup>Graz University of Technology, <sup>4</sup>University of Michigan, <sup>5</sup>University of Adelaide and Data61



## Mitigations restore security guarantees

- Involve either software changes or hardware fixes.
- Some have a large performance overhead, while others don't.

Attack	Mitigation
Meltdown	Page Table Isolation
L1TF	PTE Inversion Flush L1 Cache
LazyFP	Always save FPU
Spectre V1	Index Masking lfence after swaps
Spectre V2	Generic Retpoline
	AMD Retpoline
	IBRS
	Enhanced IBRS
Spec. Store Bypass	RSB Stuffing
	IBPB
Spec. Store Bypass	SSBD
MDS	Flush CPU Buffers
	Disable SMT



## Contributions: Understanding the performance evolution of mitigations

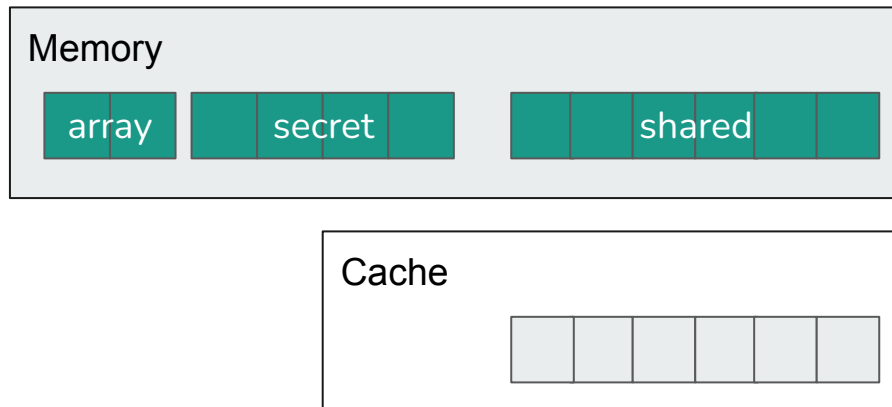
- End-to-end performance study over generations of processors
- Detailed microbenchmarking of individual mitigations
- New technique to measure speculation (not covered in this talk)

We evaluate the performance of mitigations not their security.

## Example attack: Spectre V1

```
// vulnerable if index >= SIZE
if (index < SIZE) {
    y = array[index];
    z = shared[y * CACHE_LINE];
}
```

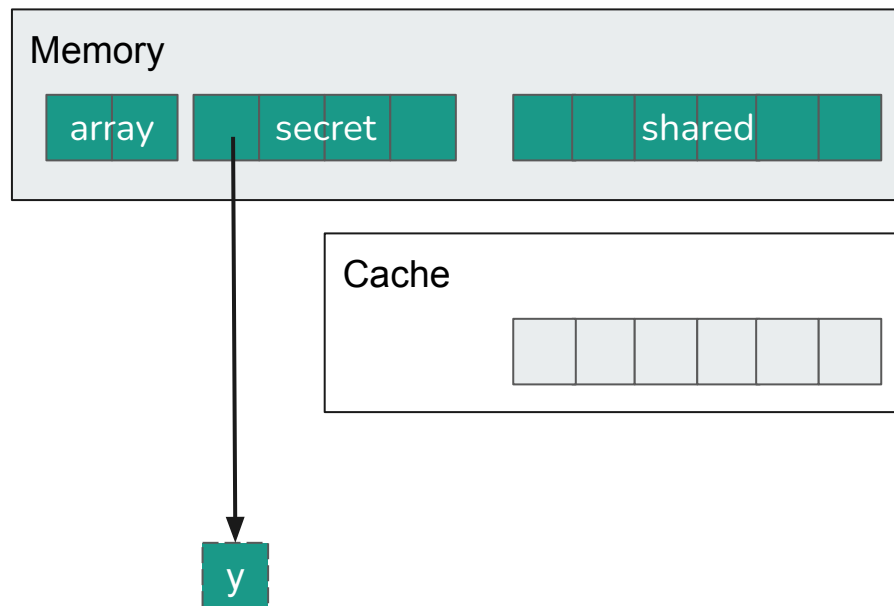
```
// userspace attacker code
secret = is_in_cache(&shared[0]);
```



## Example attack: Spectre V1

```
// vulnerable if index >= SIZE
if (index < SIZE) {
    y = array[index];
    z = shared[y * CACHE_LINE];
}

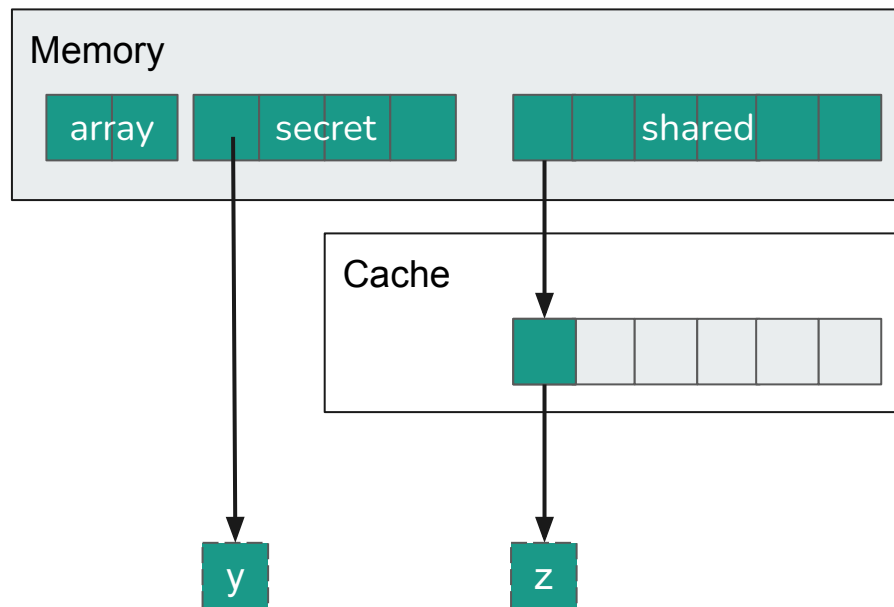
// userspace attacker code
secret = is_in_cache(&shared[0]);
```



## Example attack: Spectre V1

```
// vulnerable if index >= SIZE
if (index < SIZE) {
    y = array[index];
    z = shared[y * CACHE_LINE];
}

// userspace attacker code
secret = is_in_cache(&shared[0]);
```

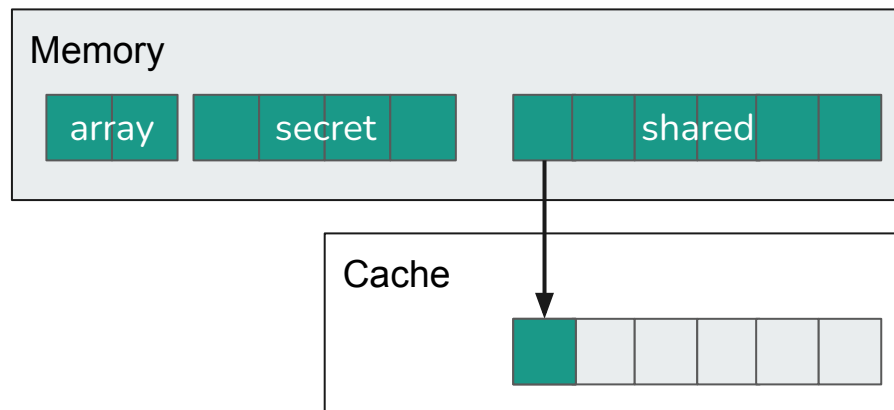




## Example attack: Spectre V1

```
// vulnerable if index >= SIZE
if (index < SIZE) {
    y = array[index];
    z = shared[y * CACHE_LINE];
}

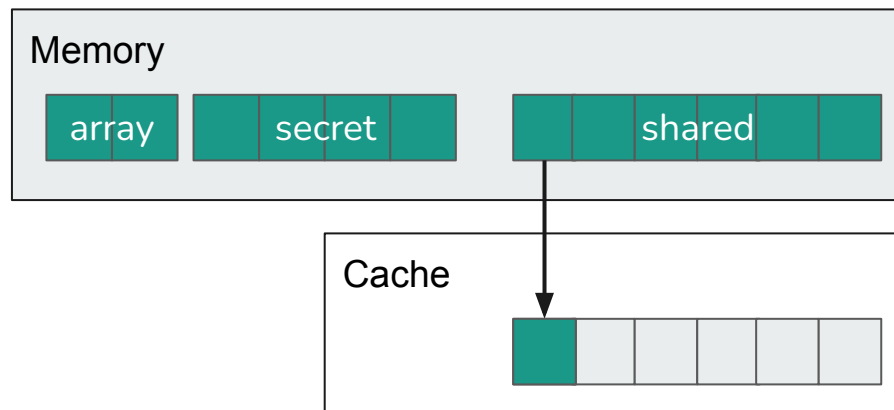
// userspace attacker code
secret = is_in_cache(&shared[0]);
```



## Example attack: Spectre V1

```
// vulnerable if index >= SIZE
if (index < SIZE) {
    y = array[index];
    z = shared[y * CACHE_LINE];
}

// userspace attacker code
secret = is_in_cache(&shared[0]);
```



## Example Mitigation

```
// vulnerable if index >= SIZE
if (index < SIZE) {
    index &= ~(long)(index|(SIZE-1-index))>>63;
    y = array[index];
    z = shared[y * CACHE_LINE];
}
```

```
// userspace attacker code
secret = is_in_cache(&shared[0]);
```

**Block out-of-bounds  
memory access**

Memory

array

secret

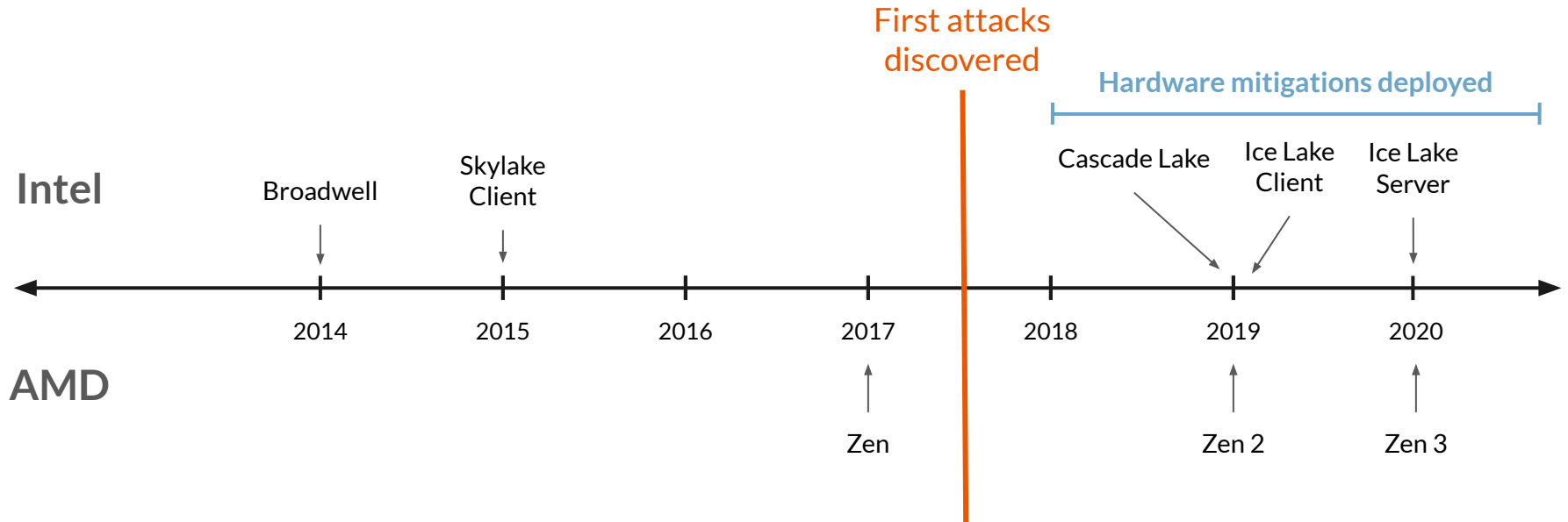
shared

Cache

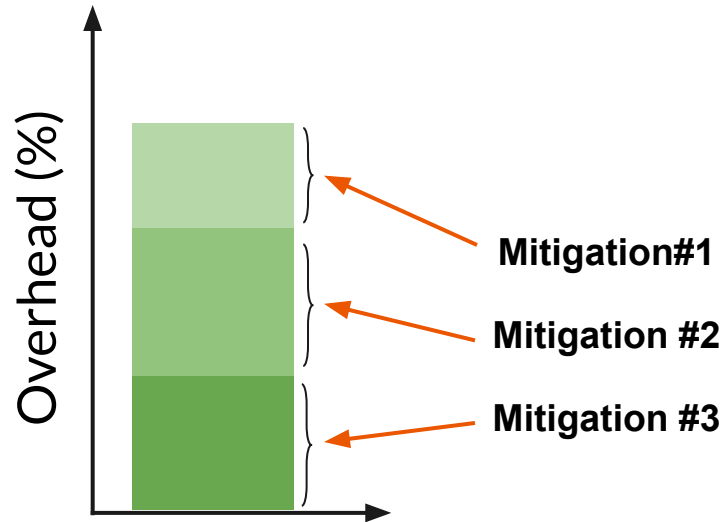
---

# Understanding Performance Impact

## Approach: Evaluate a range of CPU generations

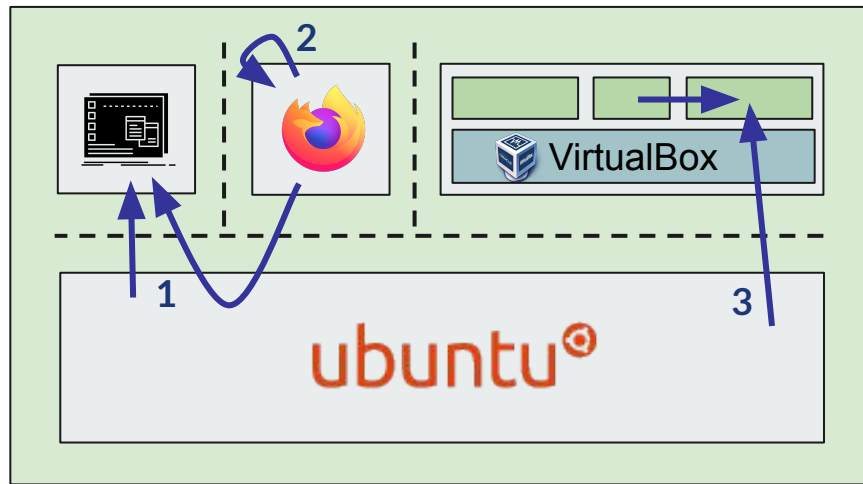



## Goal: Attribute overhead to mitigations



# Workloads: focus on security boundaries

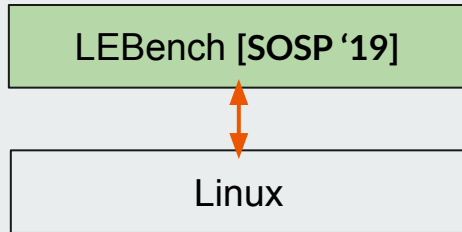
1. Operating system boundary
2. JavaScript sandbox
3. Virtual machines
  - Had minimal overhead; see paper for details





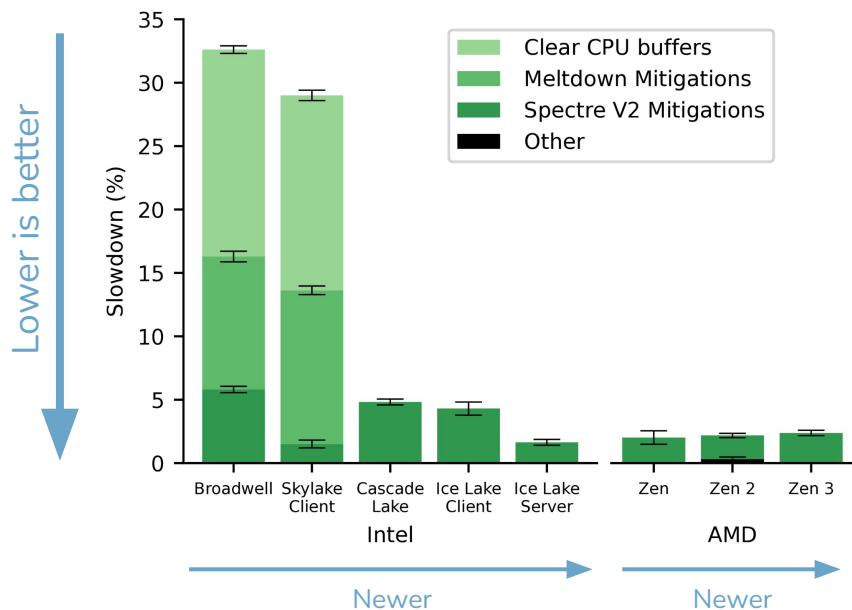
## Study 1:

# Operating System Boundary





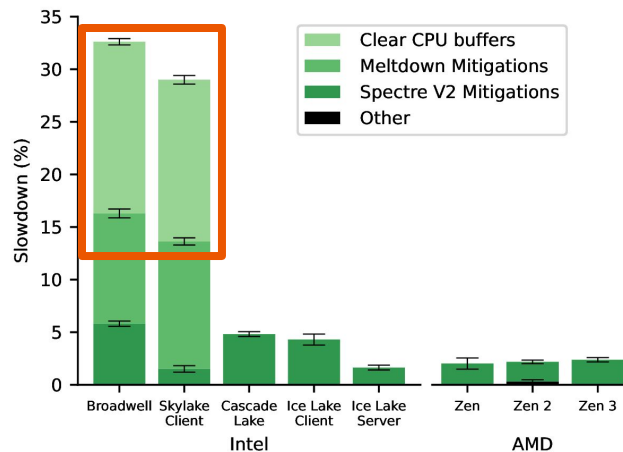
# Operating system boundary overhead has decreased



- Declined from 30% → 3%.
- Only a few attacks impact performance.

# Microarchitectural Data Sampling (MDS)

- Performance impact on Broadwell and Skylake Client
- Adds **15% overhead** to LEBench.



OS-level mitigation overhead



## MDS: Hardware fix avoids costly mitigation

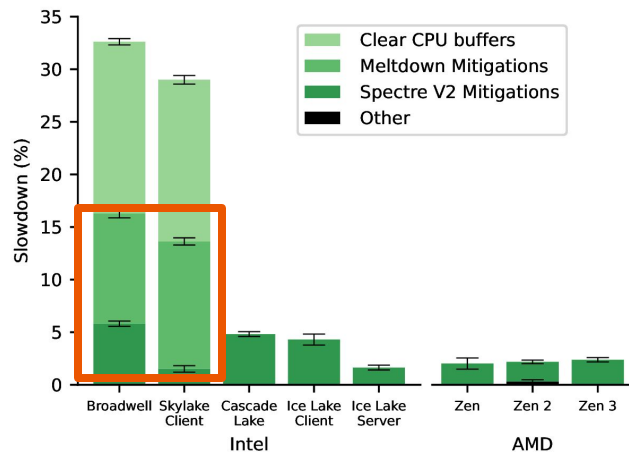
- Mitigated by executing a **verw** instruction after every system call.
- **Adds 500+ cycles overhead** to every system call.
- Only older Intel processors are vulnerable.

Vendor	CPU	Clear Cycles
Intel	Broadwell	610
	Skylake Client	518
	Cascade Lake	N/A
	Ice Lake Client	N/A
	Ice Lake Server	N/A
AMD	Zen	N/A
	Zen 2	N/A
	Zen 3	N/A

Cycles required to perform the MDS mitigation

## Meltdown: Also expensive to mitigate

- Same two processors are affected.
- Causes **10% overhead** on LEBench.



OS-level mitigation overhead



## Meltdown: Hardware fix avoids another mitigation

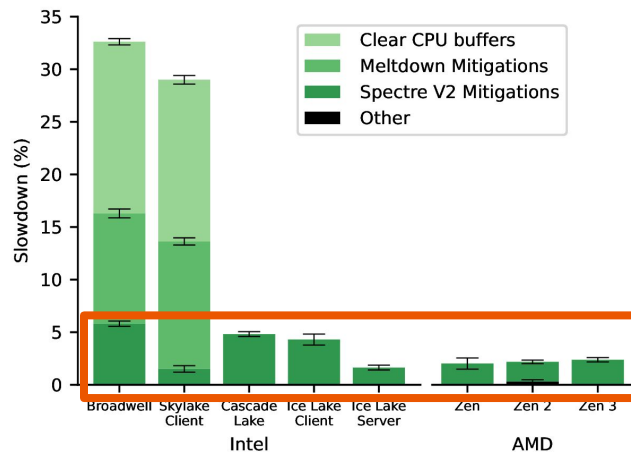
- Mitigated using **Kernel Page Table Isolation**
  - Requires switching page tables on every system call entry *and* exit.
- Cost far exceeds time spent changing privilege modes.
- Again, only older Intel processors are vulnerable

Vendor	CPU	KPTI Cycles
Intel	Broadwell	412
	Skylake Client	382
	Cascade Lake	N/A
	Ice Lake Client	N/A
	Ice Lake Server	N/A
AMD	Zen	N/A
	Zen 2	N/A
	Zen 3	N/A

Cycles spent on the mitigation during a system call

## Spectre V2: Still around but modest cost

- Impacts all our processors.
- Overhead of 3-5% on LEBench.



OS-level mitigation overhead



## Spectre V2: Involves many mitigations

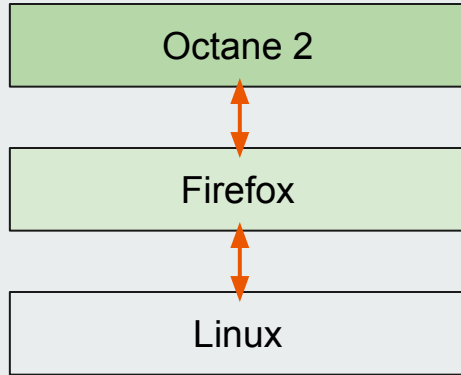
- Many different mitigations, both hardware and software.
  - See paper for a detailed look

Mitigation	<i>Broadwell</i>	<i>Skylake Client</i>	<i>Cascade Lake</i>	<i>Ice Lake Client</i>	<i>Ice Lake Server</i>	<i>Zen</i>	<i>Zen 2</i>	<i>Zen 3</i>
Retpoline	✓	✓				✓	✓	✓
IBRS								
eIBRS			✓	✓	✓			
RSB Stuffing	✓	✓	✓	✓	✓	✓	✓	✓
IBPB	✓	✓	✓	✓	✓	✓	✓	✓

The different mitigations for Spectre V2 and which processors use each.

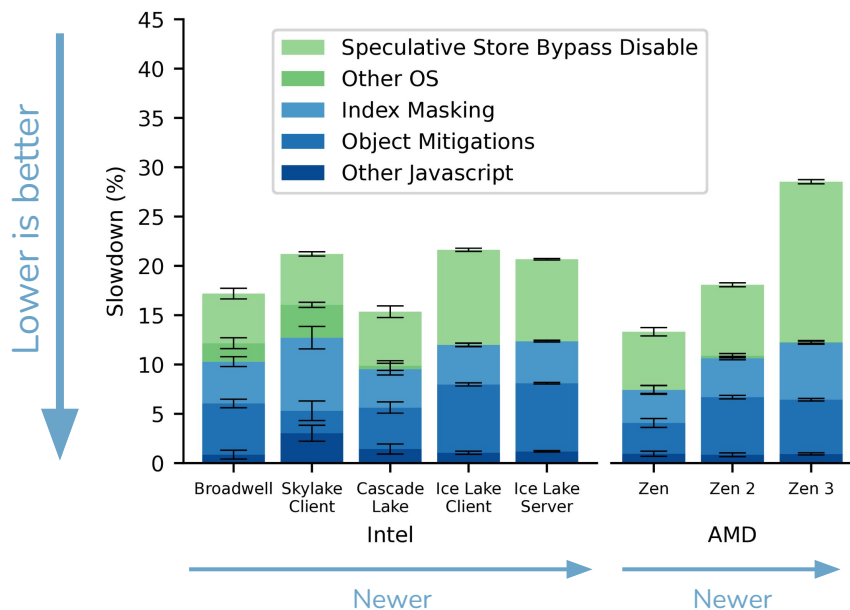


## Study 2: JavaScript Sandbox





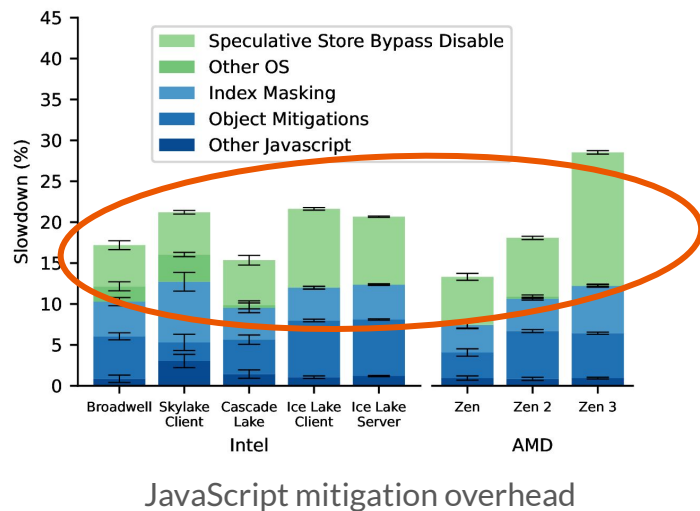
# Javascript sandbox overhead has not improved



- **No improvement** across generations
- Slowdown is in relative terms: Zen 3 is actually far faster than any of the others

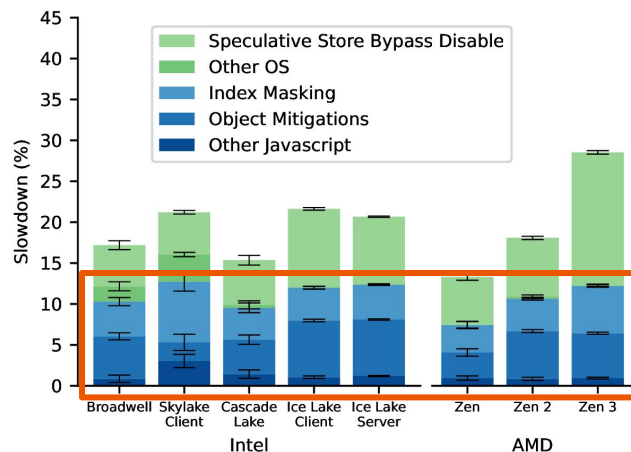
# Speculative Store Bypass: Impacts Firefox if enabled

- All our processors are vulnerable, but protection is opt-in.
- **Most programs do not use the mitigation.**
  - Firefox did when we tested, but future versions seem not to.
- ISA provides a flag to detect if a processor is vulnerable.
  - Suggests that future processors might not be.



# Spectre V1: Only impacts Firefox

- Major slowdown for JavaScript.
- Mitigated using **index masking**, **object mitigations**, and a few others.
  - Automatically by a JIT (JavaScript)
  - Or manually applied by the programmer (C code)



JavaScript mitigation overhead



# Takeaways

- Operating system boundary: new processors eliminate nearly all the overhead
- JavaScript sandboxing: overhead is still high
  - Better Spectre V1 mitigations could have a big impact
- Overhead on new CPUs is only from **Spectre V1**, **Spectre V2**, and **Speculative Store Bypass**
  - All three attacks have been known since 2018.
  - Attacks discovered since don't cause much overhead.



# Limitations

- Workloads may not be representative of all applications
  - To find out how your specific application is impacted, benchmark it!
- Some security boundaries aren't covered (e.g. the eBPF-kernel boundary)
- Future is uncertain:
  - New processor generations might be different
  - Other attacks might be discovered
  - Existing mitigations might actually be flawed



## Related work

- Many attack papers; a couple surveys including **Hill** [MICRO '19], **Canella** [CoRR '18], and **Xiong** [ACS '22].
- Lots of work on hardware and software fixes.
  - **SpecShield** [PACT '19], **Speculative Taint Tracking** [MICRO '19], **NDA** [MICRO '19], **MuonTrap** [ISCA '20], and **Speculative Data-Oblivious Execution** [ISCA '20].
  - **Site Isolation** in Firefox and Chrome and **Swivel** [USENIX Security] all for WASM bytecode.
  - **Retpolines** and **Kernel Page Table Isolation**.
- Top-level benchmarks from **Phoronix** and others.
  - We go further by attributing overheads to specific mitigations, and measuring Javascript mitigations.



## Conclusion

- We benchmarked the effect of mitigations over a range of processor generations and workloads.
- Hardware changes significantly speed up OS-level workloads, while JavaScript overheads remain.
- JavaScript Spectre V1 mitigations are a good direction for further optimization.

[github.com/mit-pdos/spectrebench](https://github.com/mit-pdos/spectrebench)

Contact: [jonathan@fintelia.io](mailto:jonathan@fintelia.io)