Rage Against the Machine Clear: A Systematic Analysis of Machine Clears and Their Implications for Transient Execution Attacks

Hany Ragab[*], **Enrico Barberis**[*], **Herbert Bos** and **Cristiano Giuffrida**

[*]Equal contribution joint first authors

Vrije Universiteit Amsterdam

Speculative Execution

```
if (x < array_size ) {
    y = array[x]
}
```

Data cache

Not cached

Speculative Execution

```
if (x < array_size ) {
    y = array[x]
}
```

Data cache

Not cached

Speculative Execution

```
if (x < array_size ) {
    y = array[x]
}
```

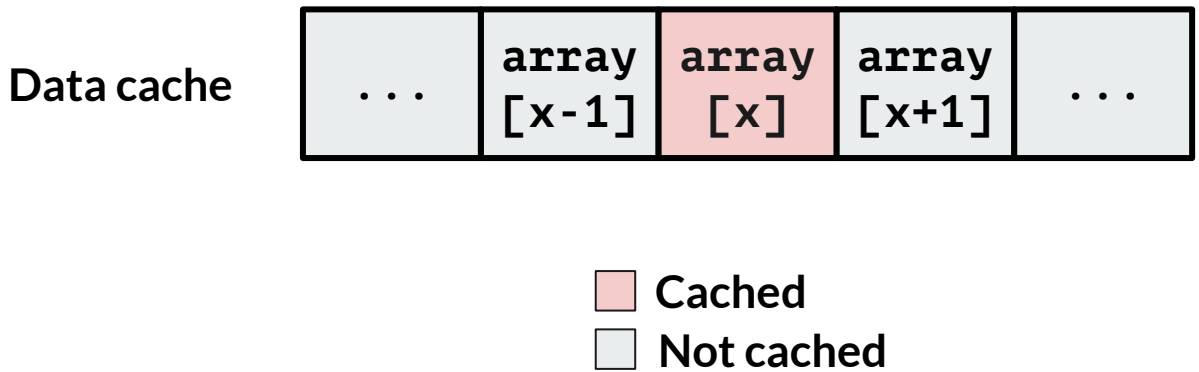
Data cache

Cached

Not cached

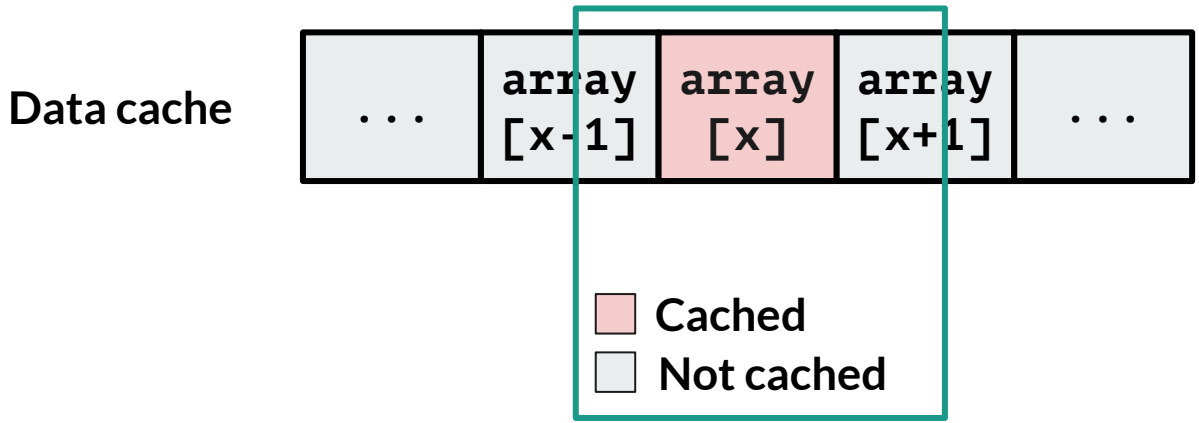
Speculative Execution

```
if (x < array_size) {
    y = array[x]
}
```



Speculative Execution

```
if (x < array_size) {
    y = array[x]
}
```



Bad Speculation

The root cause of discarding issued µOps on x86 processors

Bad Speculation

The root cause of discarding issued µOps on x86 processors

Branch Misprediction

Bad Speculation

The root cause of discarding issued µOps on x86 processors

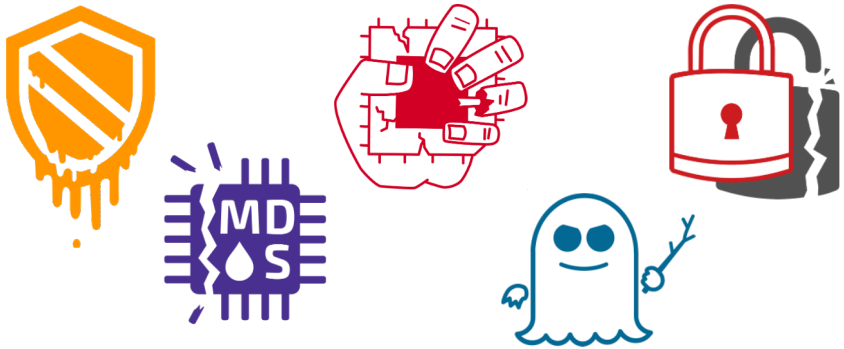
Branch Misprediction

Machine Clear

Bad Speculation

The root cause of discarding issued µOps on x86 processors

Branch Misprediction & Intel TSX

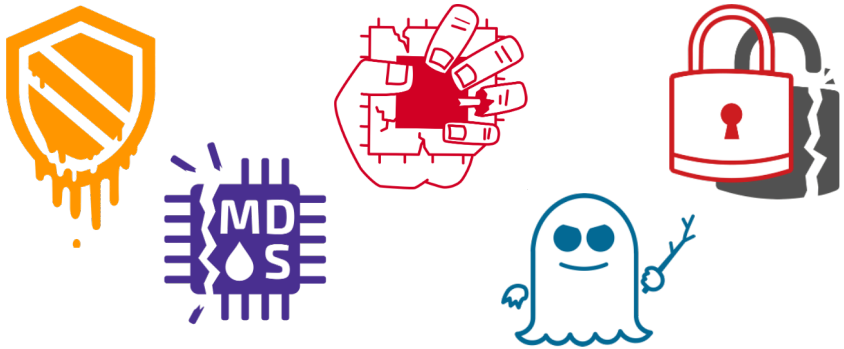


Machine Clear

Bad Speculation

The root cause of discarding issued µOps on x86 processors

Branch Misprediction & Intel TSX



Machine Clear

?

Rage Against The Machine Clear

Self-Modifying Code Machine Clear

Floating-Point Machine Clear

Memory Ordering Machine Clear

Memory Disambiguation Machine Clear

Rage Against The Machine Clear

Self-Modifying Code Machine Clear

Floating-Point Machine Clear

Rage Against The Machine Clear

Self-Modifying Code Machine Clear

Speculative Code Store Bypass (SCSB)

Negligible mitigation overhead

Floating-Point Machine Clear

Rage Against The Machine Clear

Self-Modifying Code Machine Clear

Speculative Code Store Bypass (SCSB)

Negligible mitigation overhead

Floating-Point Machine Clear

Floating-Point Value Injection (FPVI)

53% Mitigation overhead

Rage Against The Machine Clear

Self-Modifying Code Machine Clear

Floating-Point Machine Clear

End-to-end exploit leaking arbitrary memory in Firefox

With a leakage rate of **13 KB/s**

Security Analysis of Machine Clear

Security Analysis of Machine Clear

1. Architectural Invariant

Security Analysis of Machine Clear

1. Architectural Invariant
2. Invariant Violation

Security Analysis of Machine Clear

1. Architectural Invariant
2. Invariant Violation
3. Security Implications

Security Analysis of Machine Clear

1. Architectural Invariant
2. Invariant Violation
3. Security Implications
4. Exploitation

Self-Modifying Code Machine Clear

Self-Modifying Code Machine Clear

Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed

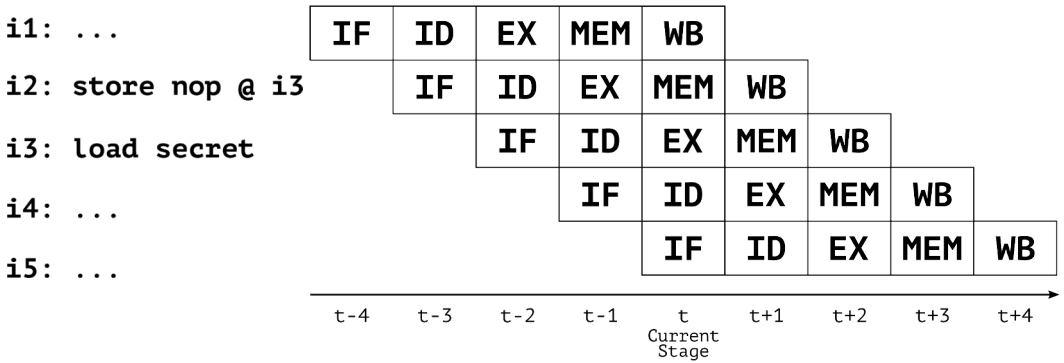
Self-Modifying Code Machine Clear

Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed

```
i1: ...
i2: store nop @ i3
i3: load secret
i4: ...
i5: ...
```

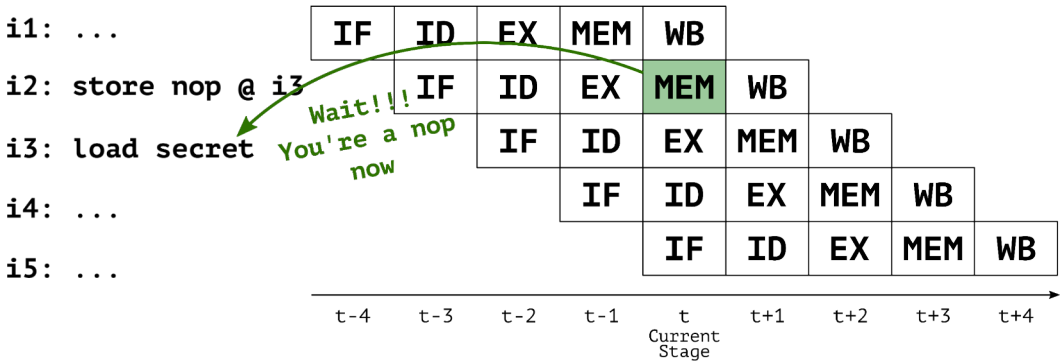
Self-Modifying Code Machine Clear

Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed



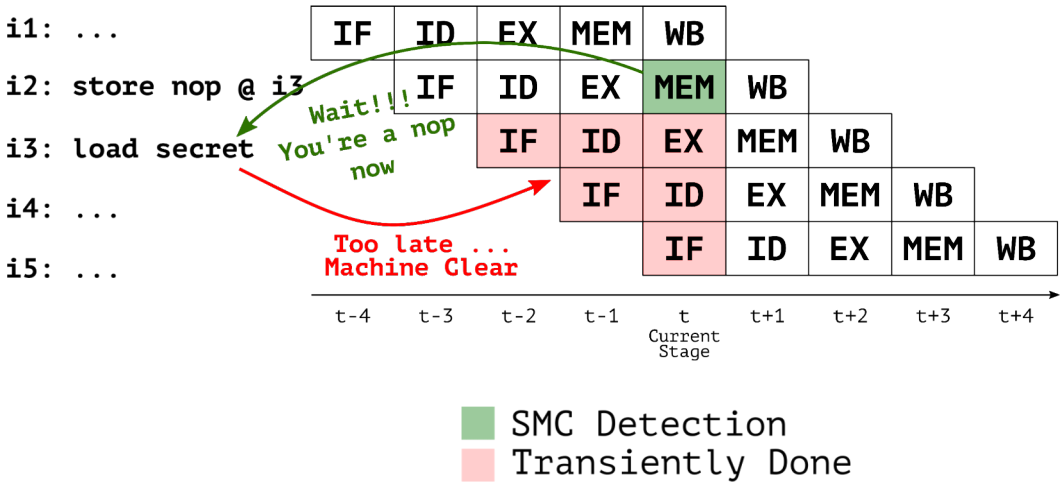
Self-Modifying Code Machine Clear

Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed



Self-Modifying Code Machine Clear

Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed

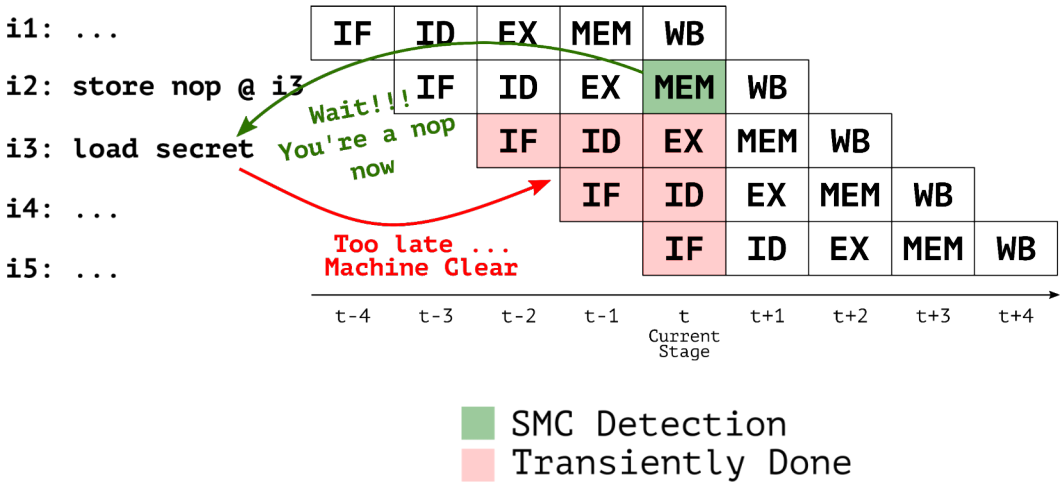


Self-Modifying Code Machine Clear

Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed

Architectural Invariant

Stores always target data



Self-Modifying Code Machine Clear

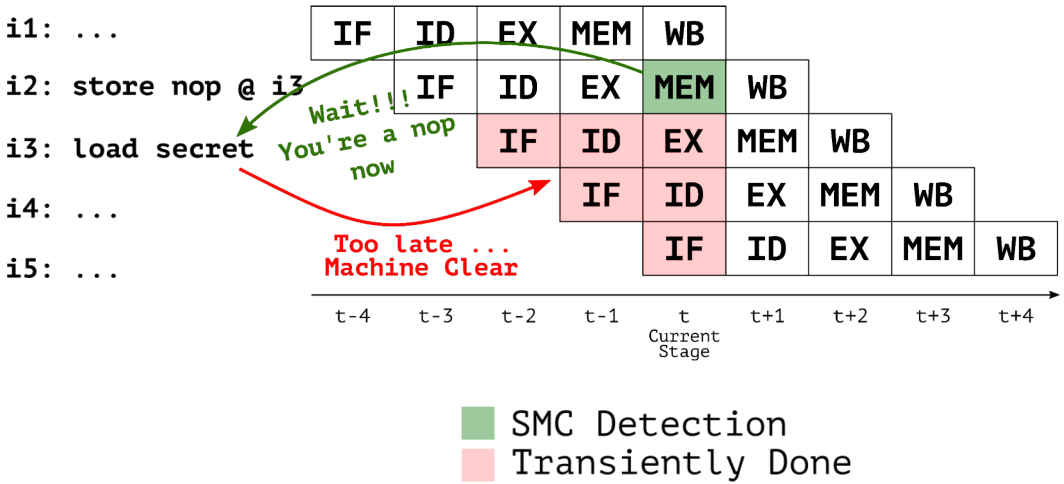
Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed

Architectural Invariant

Stores always target data

Invariant Violation

Self-Modifying Code



Self-Modifying Code Machine Clear

Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed

Architectural Invariant

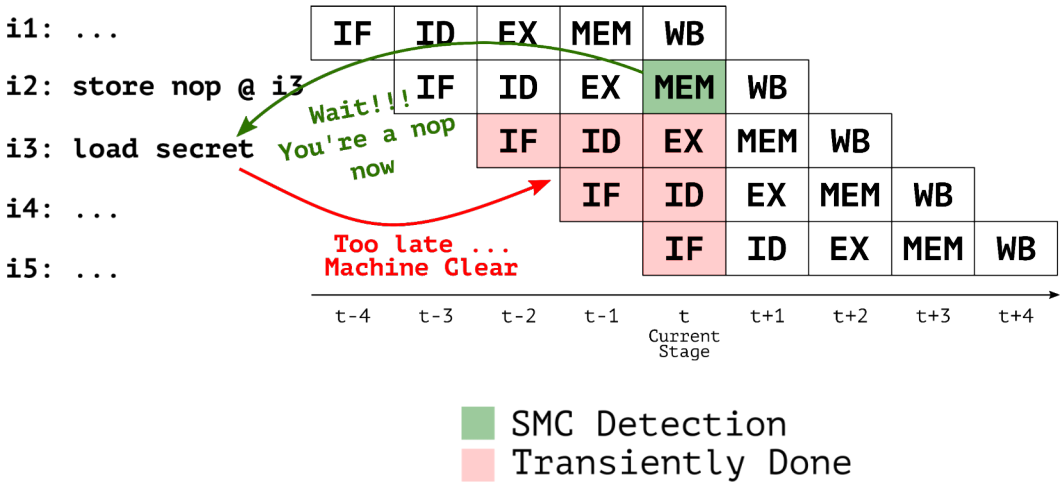
Stores always target data

Invariant Violation

Self-Modifying Code

Security Implications

Transiently execute stale code



Self-Modifying Code Machine Clear

Self-Modifying Code is a program storing instructions as data, modifying its own code as it is being executed

Architectural Invariant

Stores always target data

Invariant Violation

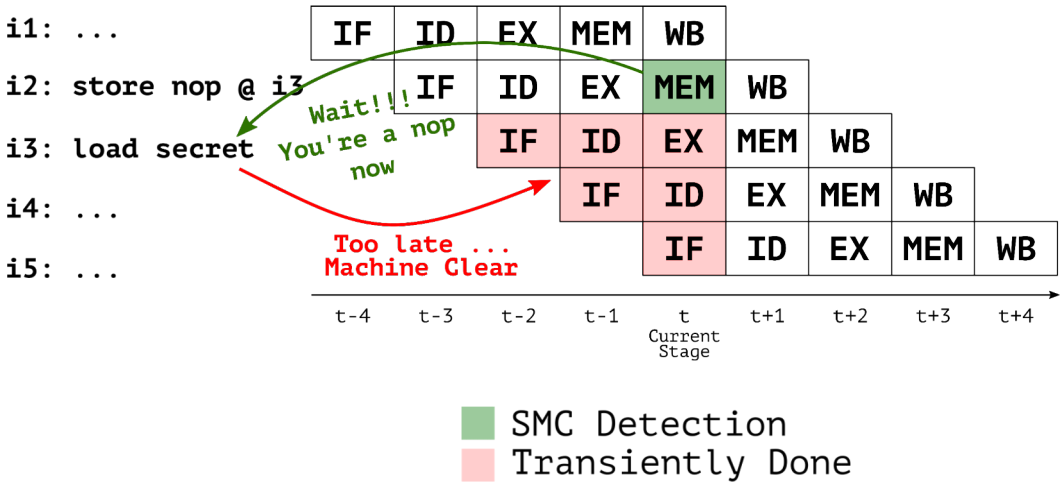
Self-Modifying Code

Security Implications

Transiently execute stale code

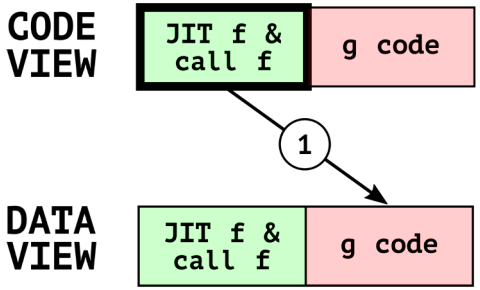
Exploitation

?

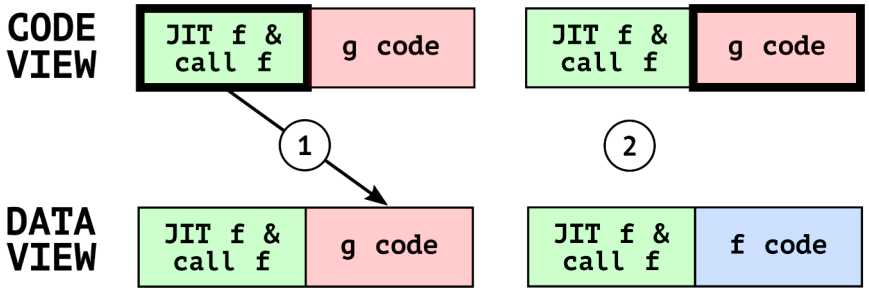


Speculative Code Store Bypass (SCSB)

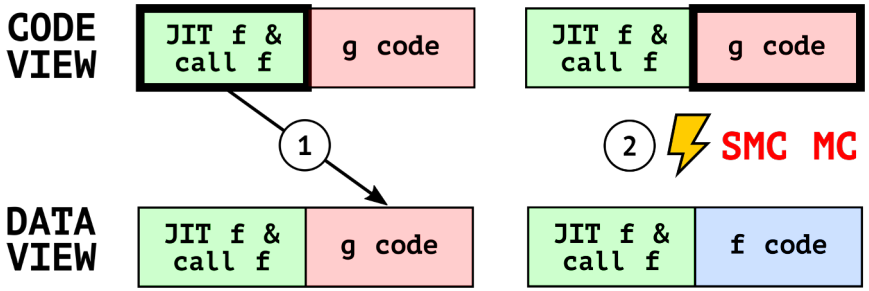
Speculative Code Store Bypass (SCSB)



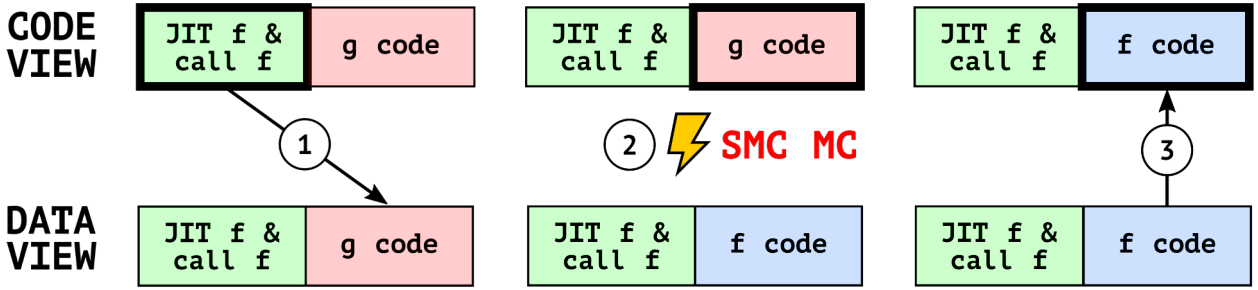
Speculative Code Store Bypass (SCSB)



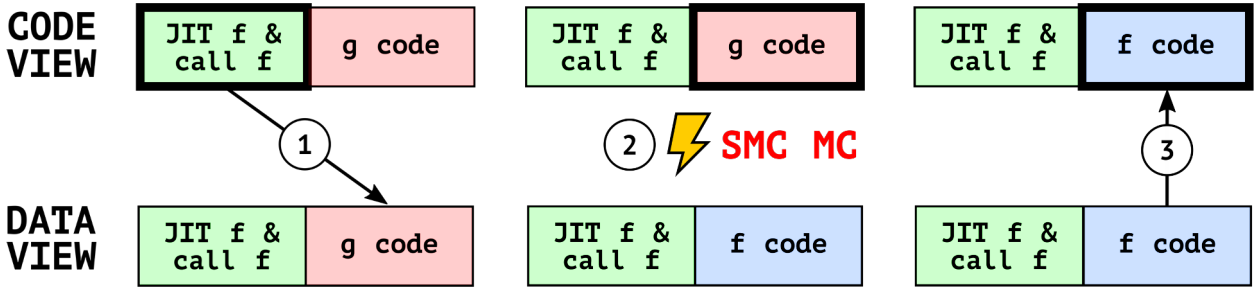
Speculative Code Store Bypass (SCSB)



Speculative Code Store Bypass (SCSB)



Speculative Code Store Bypass (SCSB)

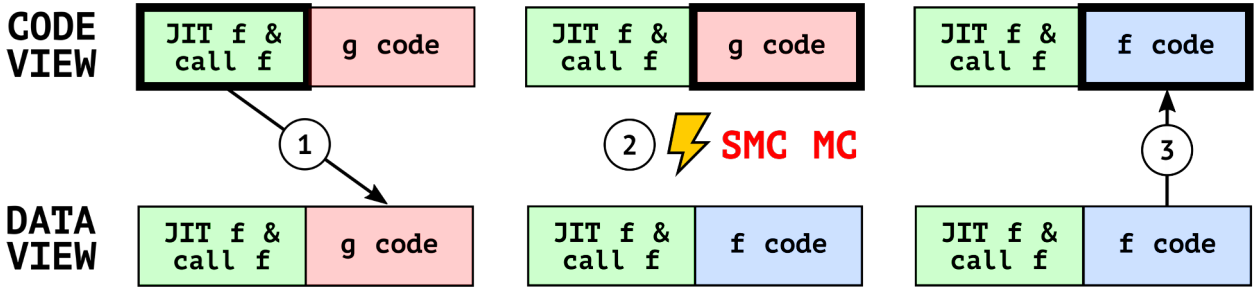


8.1.3 Handling Self- and Cross-Modifying Code

(* OPTION 1 *)
Store modified code (as data) into code segment;
Jump to new code or an intermediate location;
Execute new code;

(* OPTION 2 *)
Store modified code (as data) into code segment;
Execute a serializing instruction; (* For example, CPUID instruction *)
Execute new code;

Speculative Code Store Bypass (SCSB)



8.1.3 Handling Self- and Cross-Modifying Code

(* OPTION 1 *)
Store modified code (as data) into code segment;
Jump to new code or an intermediate location;
Execute new code;

(* OPTION 2 *)
Store modified code (as data) into code segment;
Execute a serializing instruction; (* For example, CPUID instruction *)
Execute new code;

Speculative Code Store Bypass (SCSB)

Architectural Invariant

Stores always target data memory

Invariant Violation

Self-Modifying Code

Security Implications

Transiently execute stale code

Speculative Code Store Bypass (SCSB)

Architectural Invariant

Stores always target data memory

Invariant Violation

Self-Modifying Code

Security Implications

Transiently execute stale code

Exploitation

Speculative Code Store Bypass

Floating-Point Machine Clear

Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)

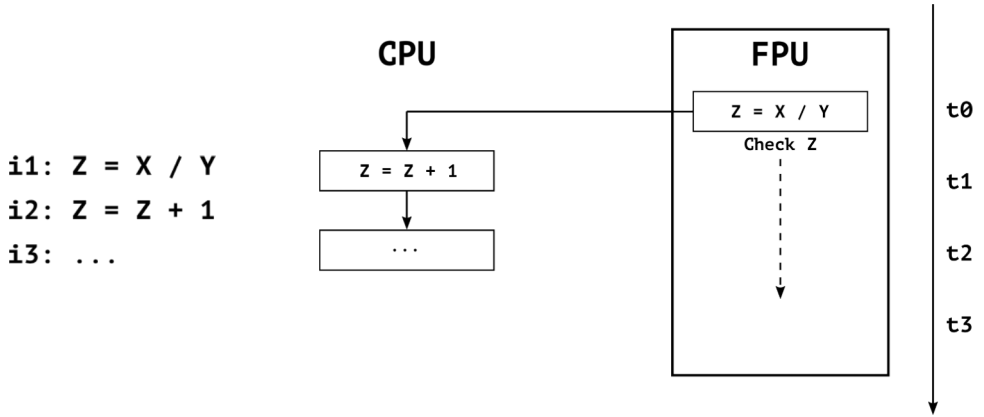
Floating-Point Machine Clear

Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)

```
i1: Z = X / Y
i2: Z = Z + 1
i3: ...
```

Floating-Point Machine Clear

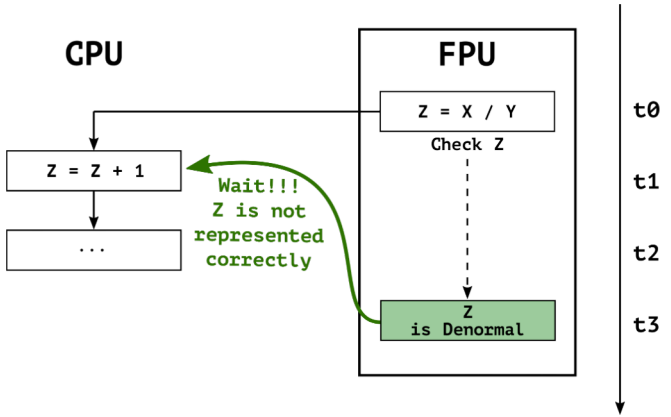
Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)



Floating-Point Machine Clear

Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)

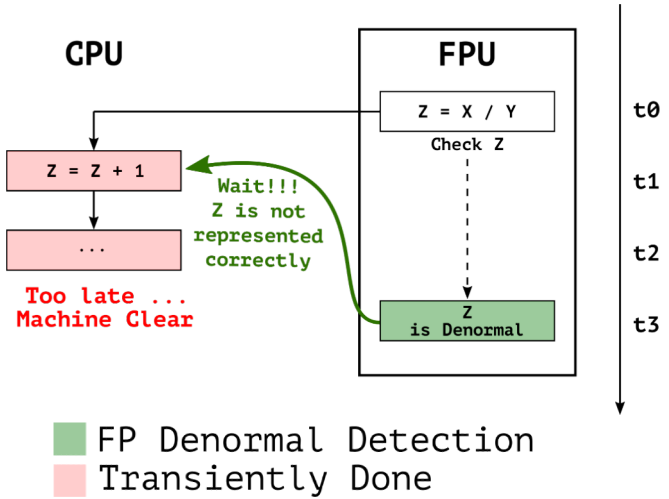
```
i1: Z = X / Y
i2: Z = Z + 1
i3: ...
```



Floating-Point Machine Clear

Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)

```
i1: Z = X / Y
i2: Z = Z + 1
i3: ...
```



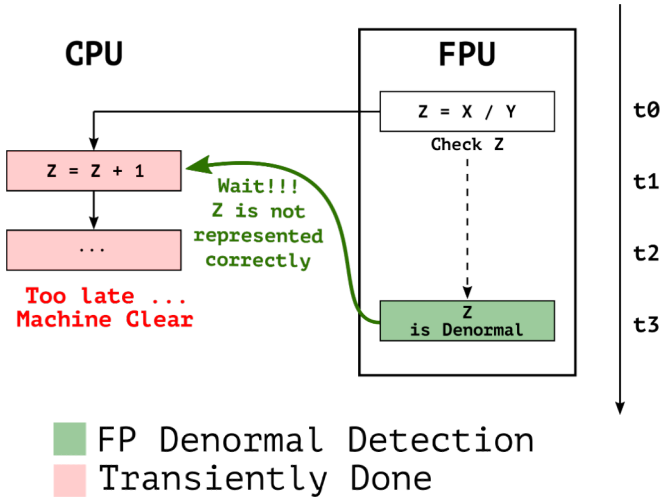
Floating-Point Machine Clear

Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)

Architectural Invariant

FPU always operates on normal numbers

```
i1: Z = X / Y
i2: Z = Z + 1
i3: ...
```



Floating-Point Machine Clear

Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)

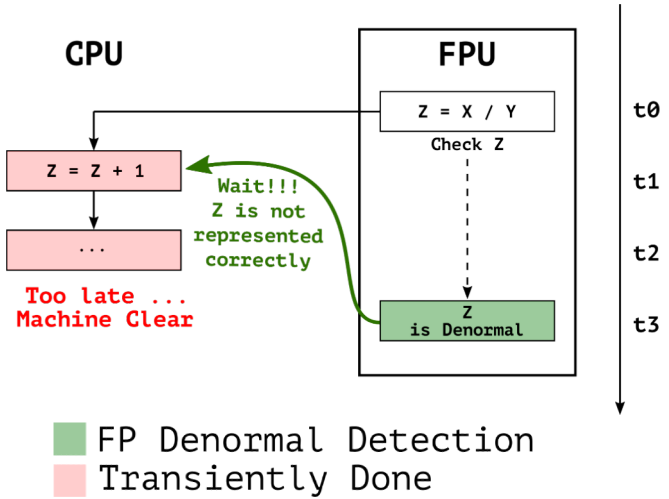
Architectural Invariant

FPU always operates on normal numbers

Invariant Violation

Subnormal FP operations

```
i1: Z = X / Y
i2: Z = Z + 1
i3: ...
```



Floating-Point Machine Clear

Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)

Architectural Invariant

FPU always operates on normal numbers

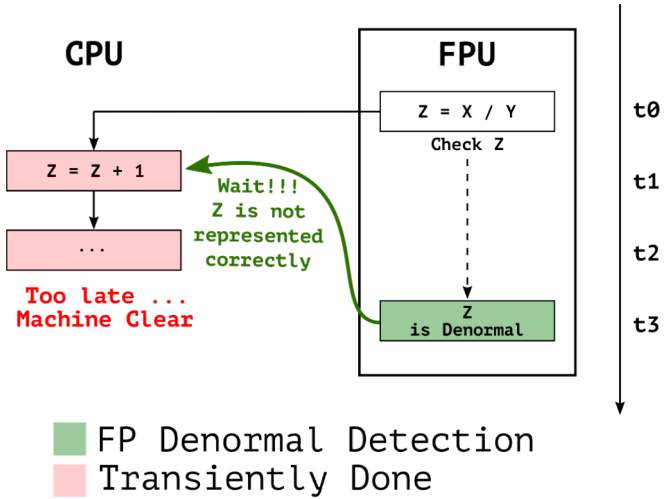
Invariant Violation

Subnormal FP operations

Security Implications

Transiently inject arbitrary FP values

```
i1: Z = X / Y
i2: Z = Z + 1
i3: ...
```



Floating-Point Machine Clear

Subnormal/Denormal numbers are a special range of floating-point numbers with a value smaller than the smallest Normal number (i.e. 2^-1022)

Architectural Invariant

FPU always operates on normal numbers

Invariant Violation

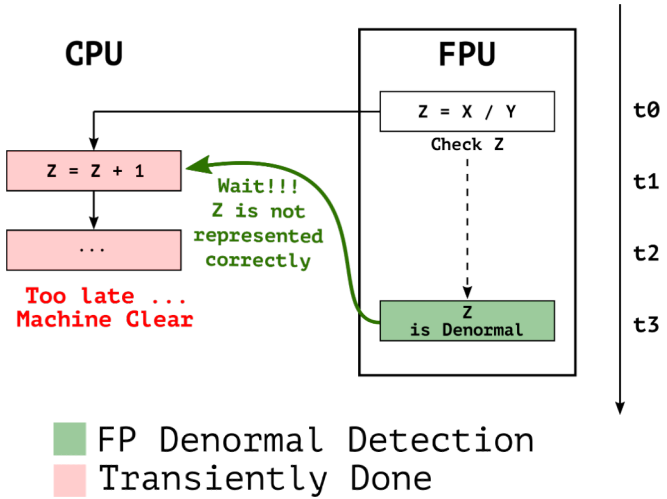
Subnormal FP operations

```
i1: Z = X / Y
i2: Z = Z + 1
i3: ...
```

Security Implications

Transiently inject arbitrary FP values

Exploitation

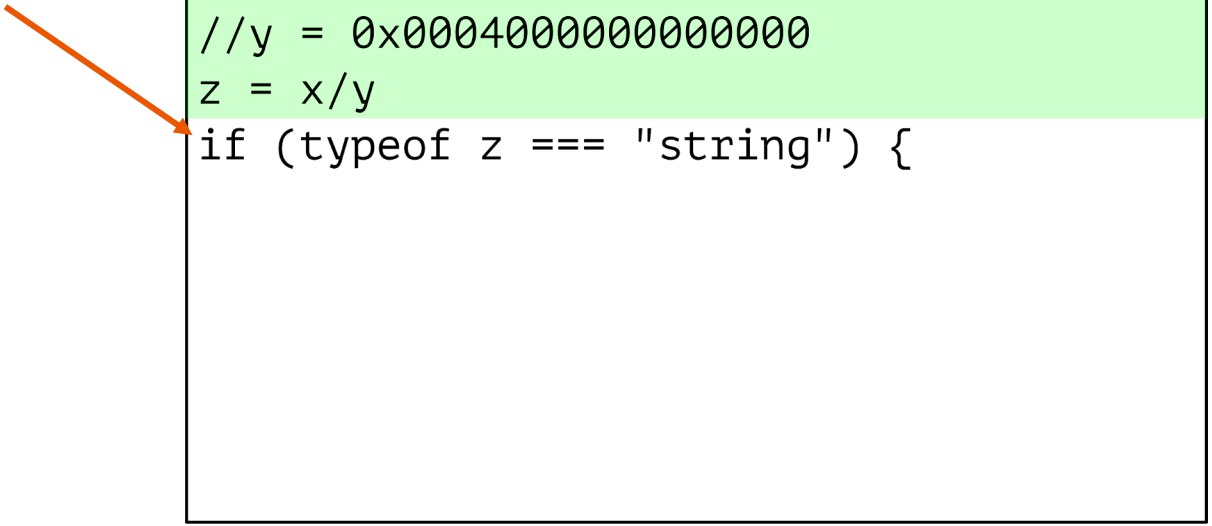


Floating-Point Value Injection (FPVI)

0x**fffb0**deadbeef000
JSVAL_TYPE_**STRING**
PAYLOAD:
0xdeadbeef000

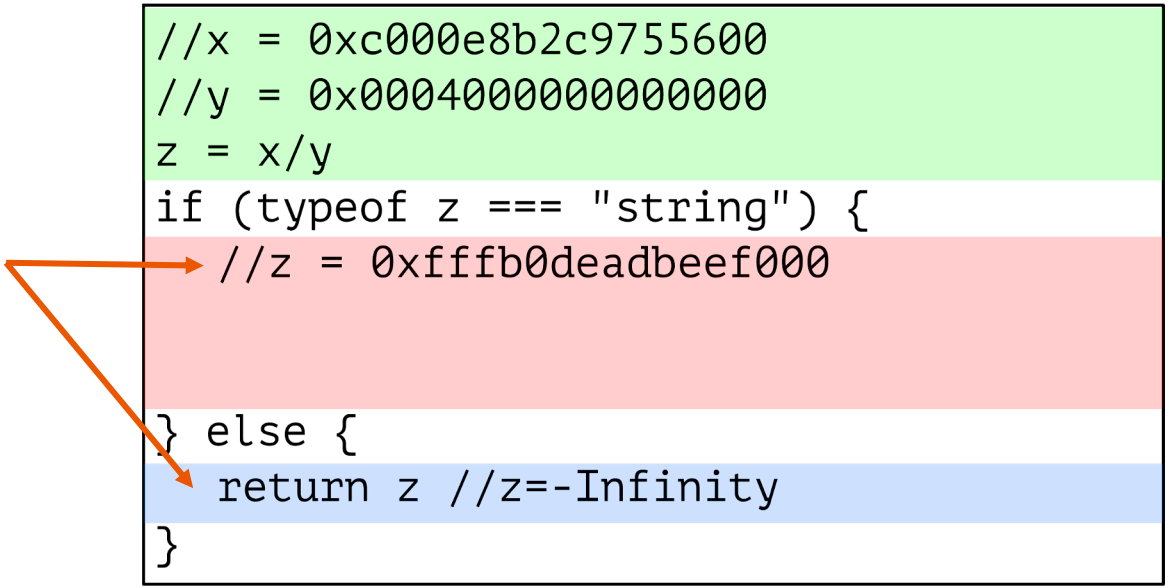
Floating-Point Value Injection (FPVI)

0x**fffb0**deadbeef000
JSVAL_TYPE_**STRING**
PAYLOAD:
0xdeadbeef000



Floating-Point Value Injection (FPVI)

0x**fffb0**deadbeef000 JSVAL_TYPE_**STRING** PAYLOAD: **0xdeadbeef000**	0x**fff00**00000000000 JSVAL_TYPE_**DOUBLE** PAYLOAD: **-Infinity**



Floating-Point Value Injection (FPVI)

0x**fffb0**deadbeef000 JSVAL_TYPE_**STRING** PAYLOAD: **0xdeadbeef000**	0x**fff00**00000000000 JSVAL_TYPE_**DOUBLE** PAYLOAD: **-Infinity**

```
//x = 0xc000e8b2c9755600
//y = 0x0004000000000000
z = x/y
if (typeof z === "string") {
   //z = 0xfffb0deadbeef000
   //leak byte @ 0xdeadbeef004
   return buf[(z.length&0xff)<<10]
} else {
   return z //z=-Infinity
}
```

Floating-Point Value Injection (FPVI)

Architectural Invariant

FPU always operates on normal numbers

Invariant Violation

Denormal FP operations

Security Implications

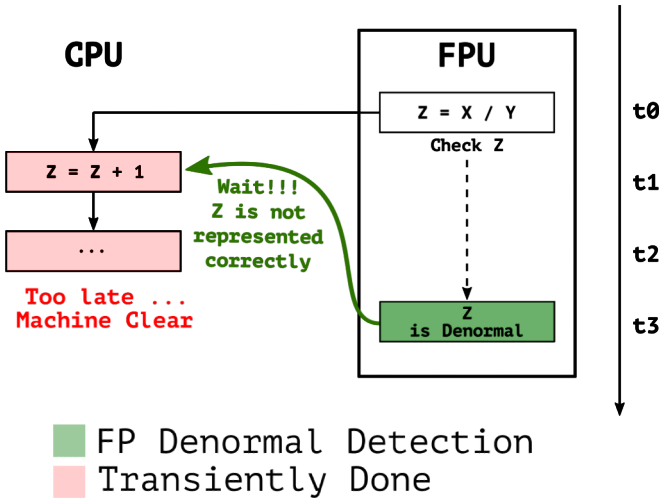
Transiently inject arbitrary FP values

Exploitation

Floating-Point Value Injection

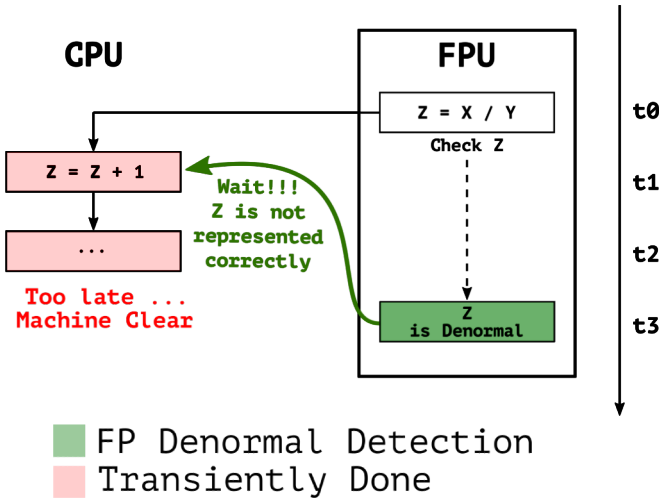
Floating-Point Value Injection (FPVI)

- Exploit leakage rate of 13 KB/s



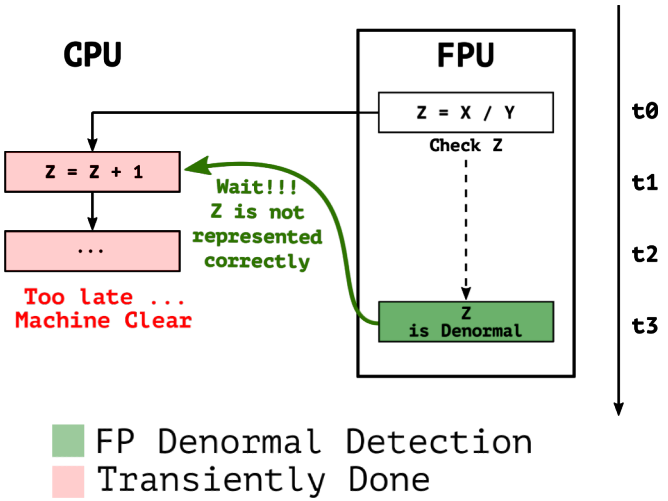
Floating-Point Value Injection (FPVI)

- Exploit leakage rate of 13 KB/s
- Mitigations:
  - ➔ Flush To Zero (FTZ) & Denormal Are Zero (DAZ)



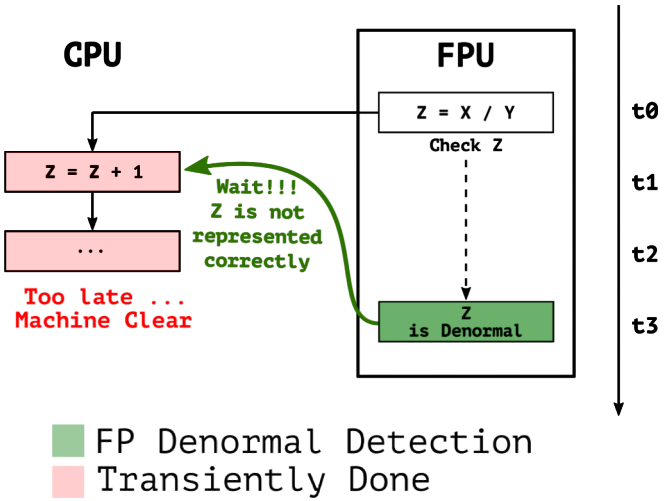
Floating-Point Value Injection (FPVI)

- **Exploit leakage rate of 13 KB/s**
- **Mitigations:**
  - ➔ **Flush To Zero (FTZ) & Denormal Are Zero (DAZ)**
  - ➔ We implemented a **LLVM pass** adding a serializing instruction in detected FPVI gadgets.
    With 53% geomean overhead for SPEC FP 2017.

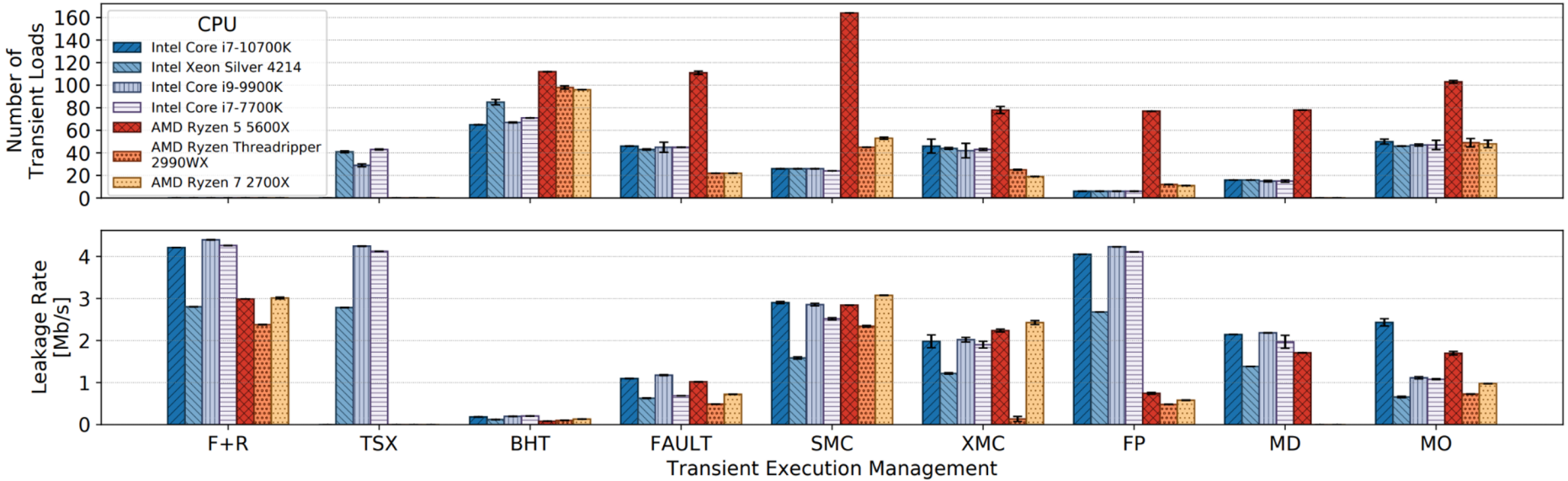


Floating-Point Value Injection (FPVI)

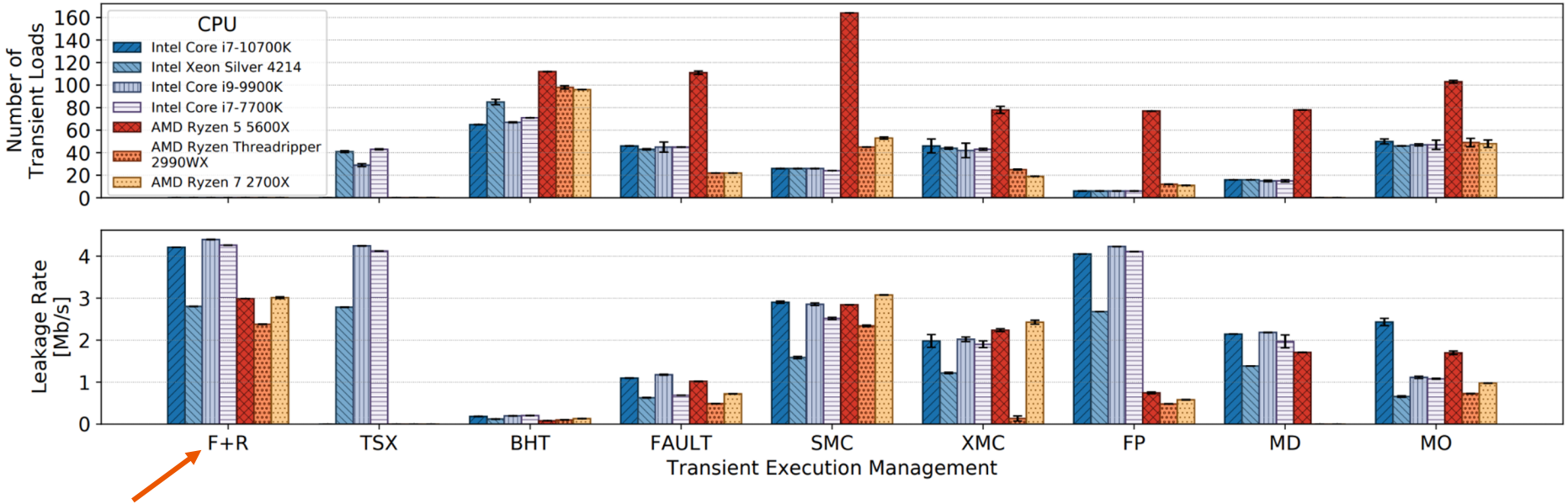
- **Exploit leakage rate of 13 KB/s**
- **Mitigations:**
  - ➔ **Flush To Zero (FTZ) & Denormal Are Zero (DAZ)**
  - ➔ We implemented a **LLVM pass** adding a serializing instruction in detected FPVI gadgets.
    With 53% geomean overhead for SPEC FP 2017.
  - ➔ Use **site-isolation** or **conditionally mask FP** operations in the browsers.



Transient Execution Capabilities

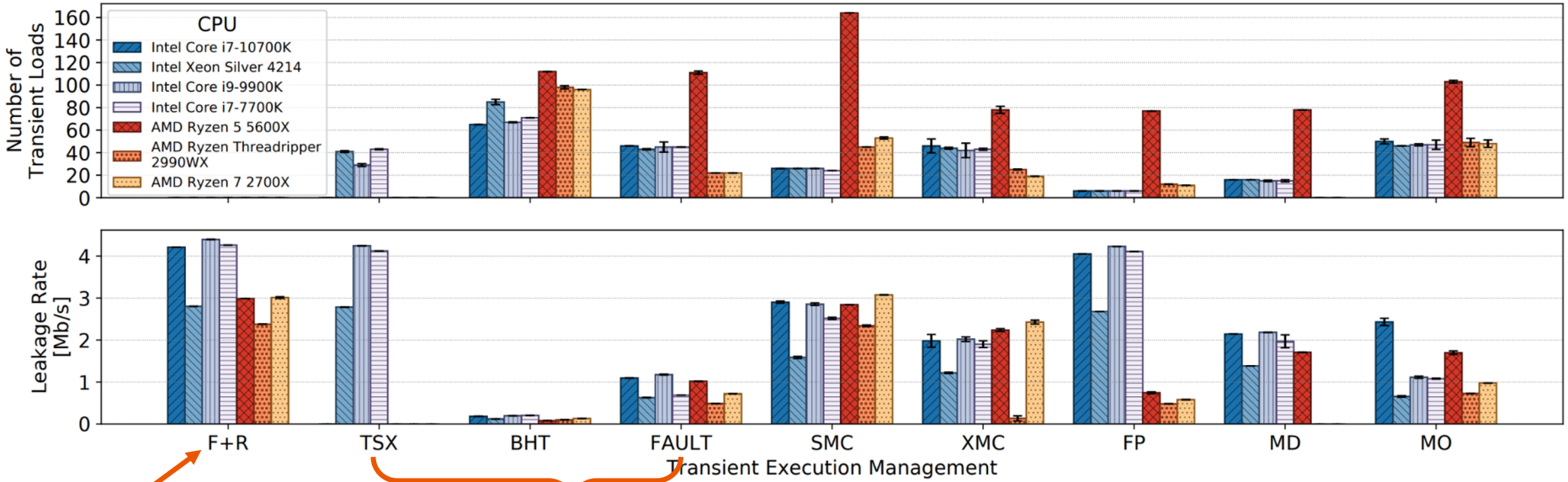


Transient Execution Capabilities



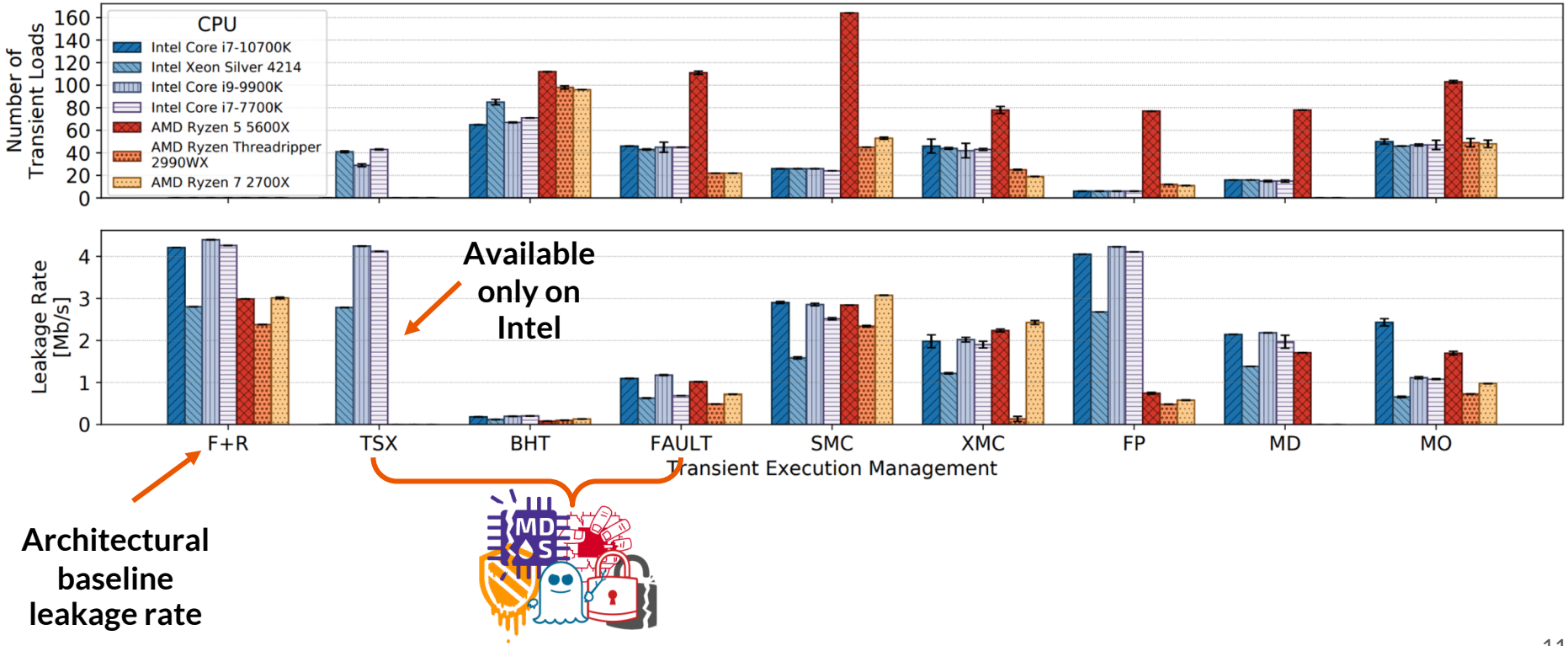
Architectural baseline leakage rate

Transient Execution Capabilities

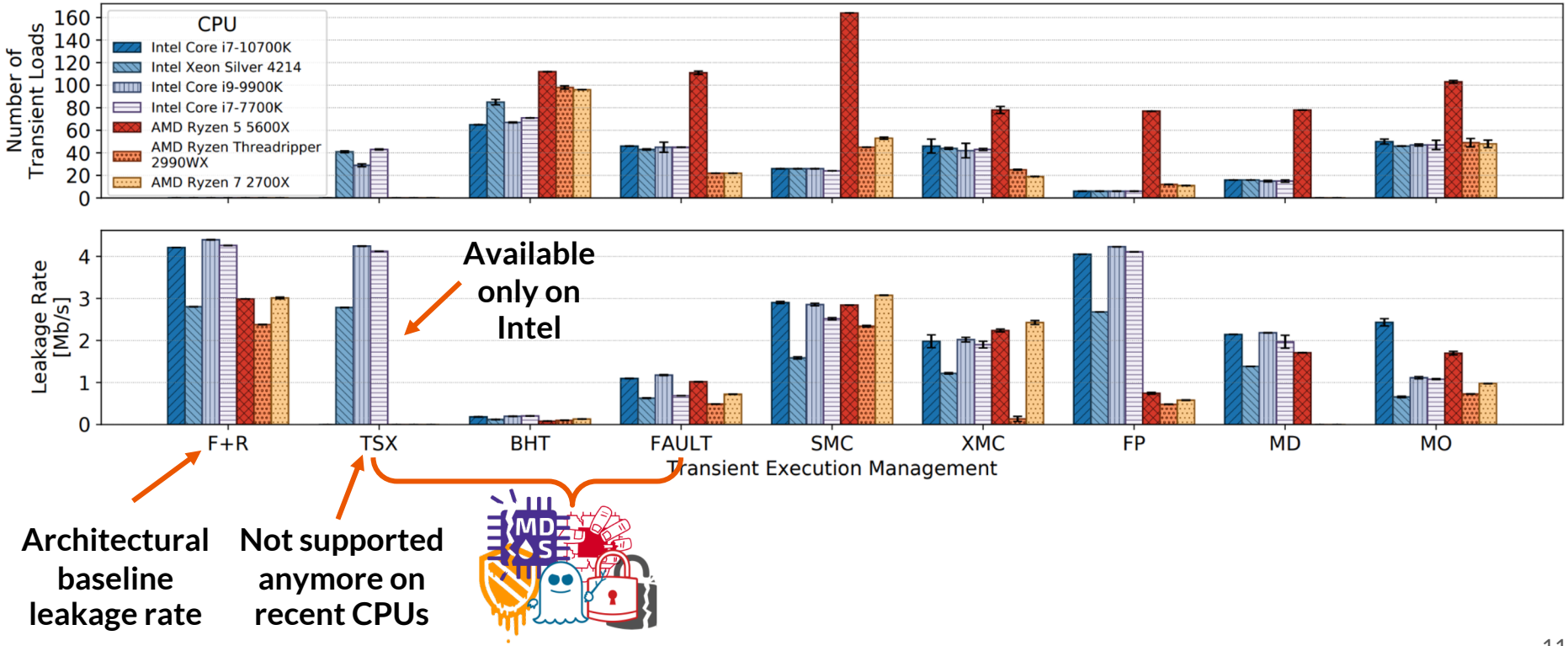


Architectural baseline leakage rate

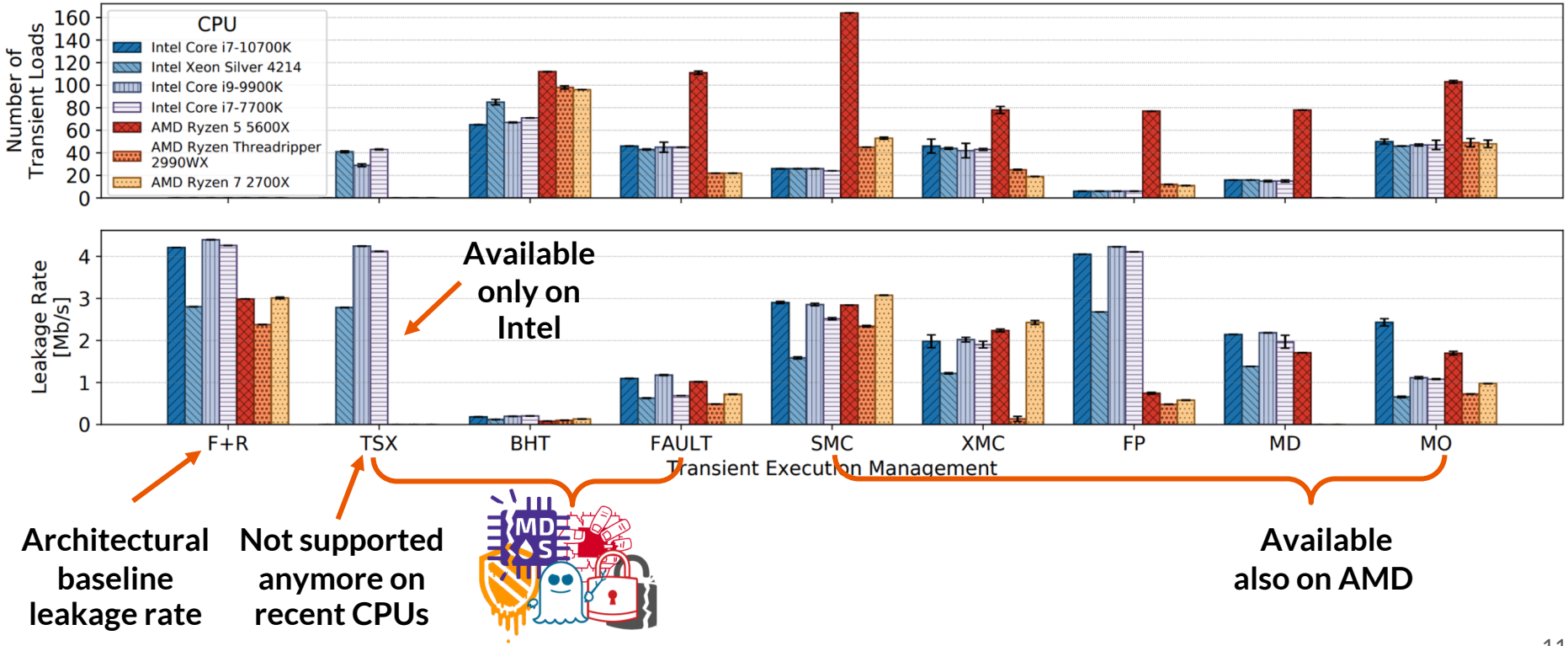
Transient Execution Capabilities



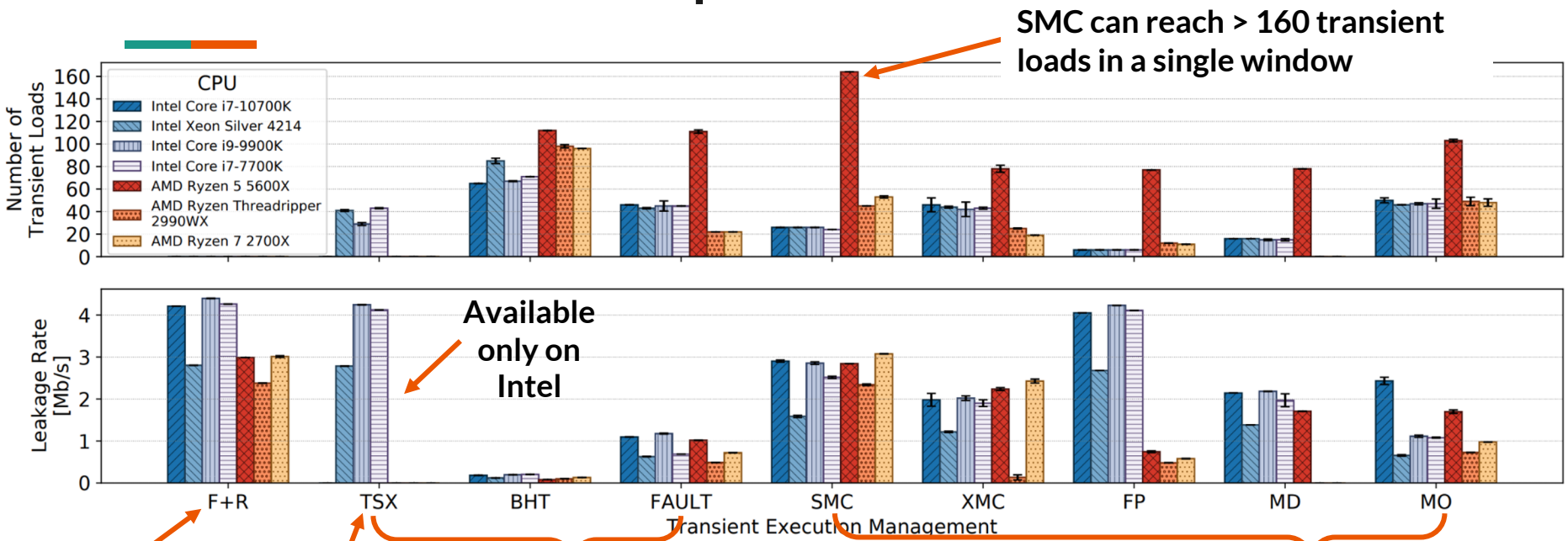
Transient Execution Capabilities



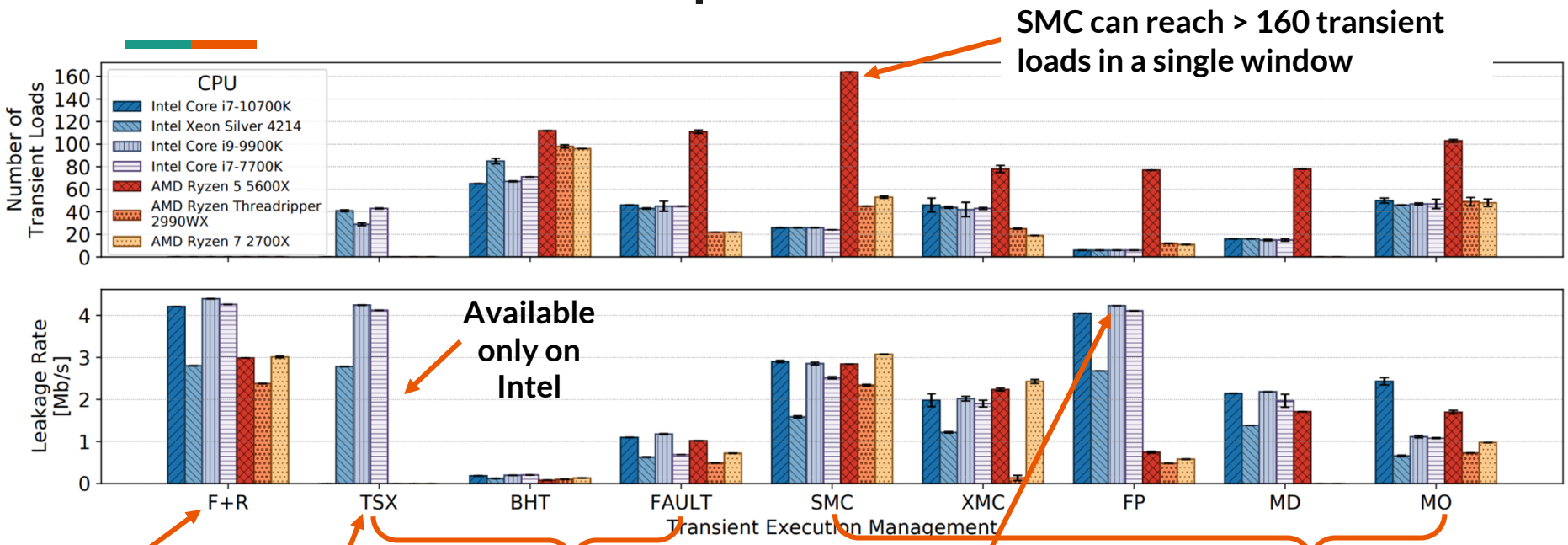
Transient Execution Capabilities



Transient Execution Capabilities

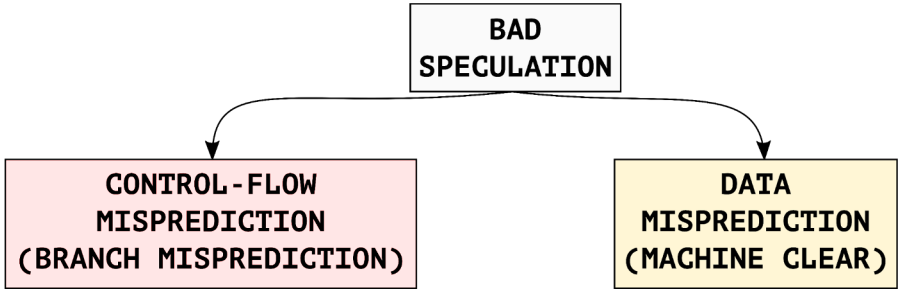


Transient Execution Capabilities

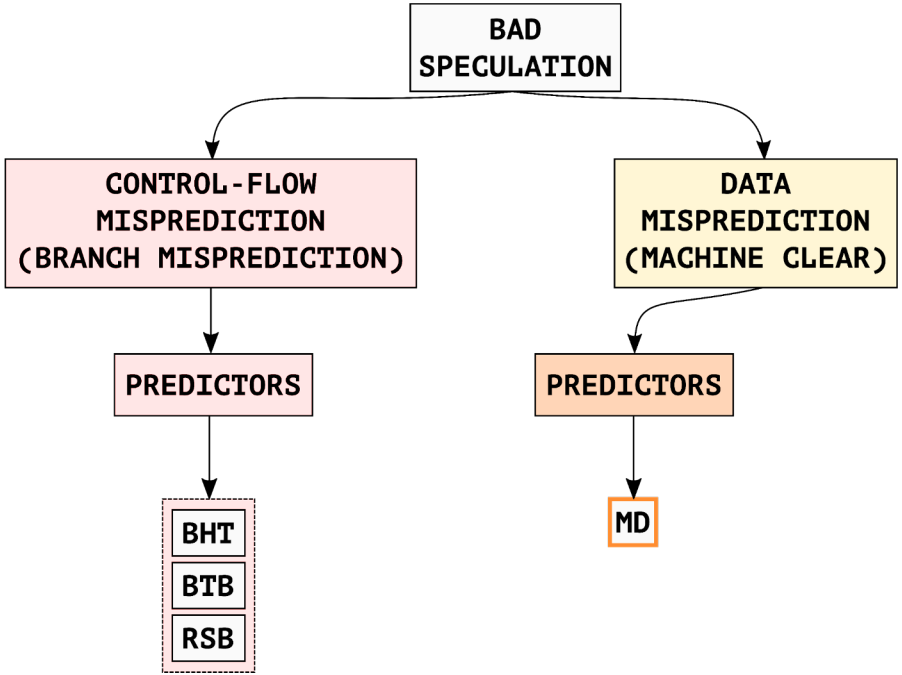


Root-Cause Classification of Transient Execution

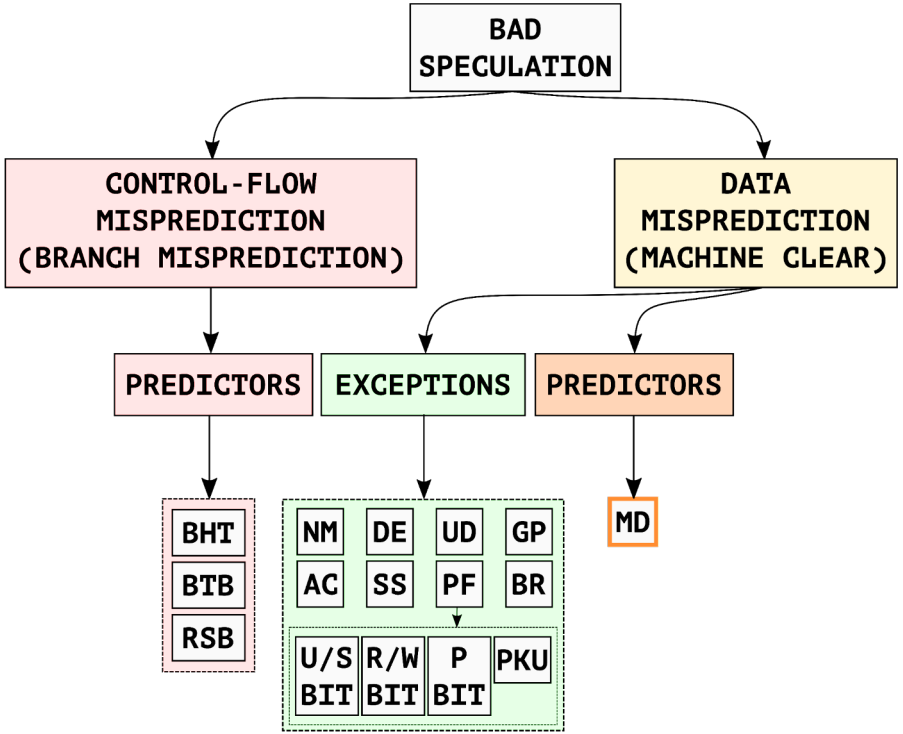
Root-Cause Classification of Transient Execution



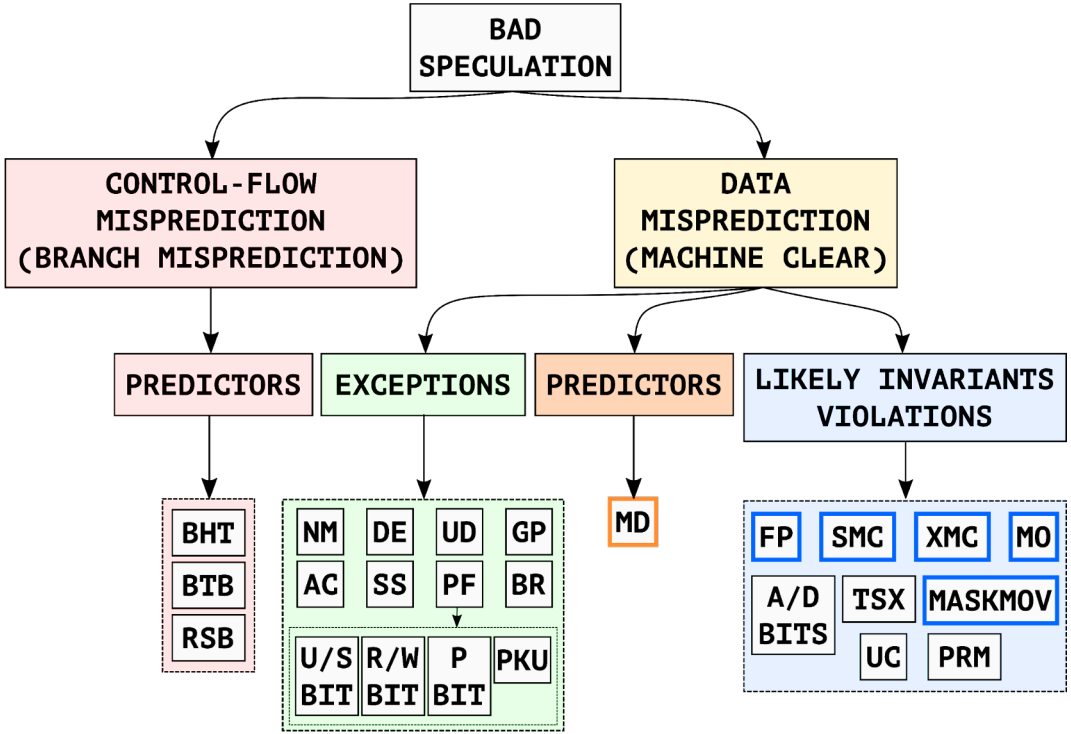
Root-Cause Classification of Transient Execution



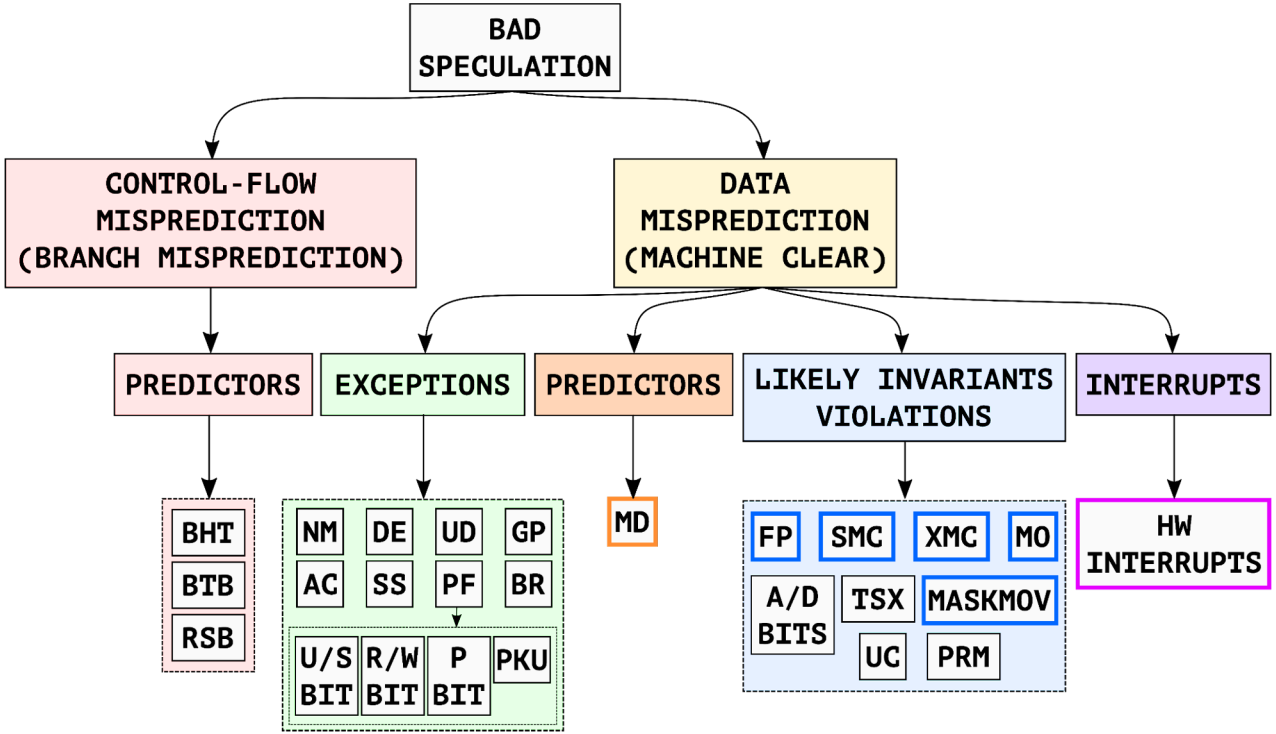
Root-Cause Classification of Transient Execution



Root-Cause Classification of Transient Execution



Root-Cause Classification of Transient Execution



Disclosure & Affected CPUs

- We disclosed FPVI and SCSB to CPU, browser, OS, and hypervisor vendors in February 2021.

Disclosure & Affected CPUs

- We disclosed FPVI and SCSB to CPU, browser, OS, and hypervisor vendors in February 2021.

CPU Vendor	**Affected by SCSB** (CVE-2021-0089) (CVE-2021-26313)	**Affected by FPVI** (CVE-2021-0086) (CVE-2021-26314)
Intel	✓	✓
AMD	✓	✓*
ARM	✗	✓**

* No exploitable NaN-boxed transient results were found
** ARM reported that some FPU implementations are affected by FPVI

Disclosure & Affected CPUs

- We disclosed FPVI and SCSB to CPU, browser, OS, and hypervisor vendors in February 2021.

- Mozilla confirmed the FPVI vulnerability (CVE-2021-29955) and deployed a mitigation based on conditionally masking malicious NaN-boxed FP results in Firefox 87.

CPU Vendor	Affected by SCSB (CVE-2021-0089) (CVE-2021-26313)	Affected by FPVI (CVE-2021-0086) (CVE-2021-26314)
Intel	✓	✓
AMD	✓	✓*
ARM	✗	✓**

* No exploitable NaN-boxed transient results were found
** ARM reported that some FPU implementations are affected by FPVI

Disclosure & Affected CPUs

- We disclosed FPVI and SCSB to CPU, browser, OS, and hypervisor vendors in February 2021.
- Mozilla confirmed the FPVI vulnerability (CVE-2021-29955) and deployed a mitigation based on conditionally masking malicious NaN-boxed FP results in Firefox 87.
- Xen hypervisor mitigated SCSB and released a security advisory (XSA-375) following our proposed mitigation.

CPU Vendor	Affected by SCSB (CVE-2021-0089) (CVE-2021-26313)	Affected by FPVI (CVE-2021-0086) (CVE-2021-26314)
Intel	✓	✓
AMD	✓	✓*
ARM	✗	✓**

* No exploitable NaN-boxed transient results were found
** ARM reported that some FPU implementations are affected by FPVI

Rage Against The Machine Clear

- **Bad Speculation is not caused only by classic mispredictions**

Rage Against The Machine Clear

- **Bad Speculation is not caused only by classic mispredictions, but also by architectural invariants violations, i.e. Machine Clear.**

Rage Against The Machine Clear

- Bad Speculation is not caused only by classic mispredictions, but also by architectural invariants violations, i.e. Machine Clear.

- Architectural invariants can be exploited, creating new security threats, e.g. FPVI & SCSB

Rage Against The Machine Clear

- Bad Speculation is not caused only by classic mispredictions, but also by architectural invariants violations, i.e. Machine Clear.

- Architectural invariants can be exploited, creating new security threats, e.g. FPVI & SCSB

- Defenses must focus on the wider class of root-causes of bad speculation.

Rage Against The Machine Clear

- Bad Speculation is not caused only by classic mispredictions, but also by architectural invariants violations, i.e. Machine Clear.

- Architectural invariants can be exploited, creating new security threats, e.g. FPVI & SCSB

- Defenses must focus on the wider class of root-causes of bad speculation.

VUSec

@hanyrax @enrico_barberis

Code, exploit demo and more can be found here: