Leaky Processors

Lessons from Spectre, Meltdown, and Foreshadow

Jo Van Bulck (@jovanbulck)[1], **Daniel Gruss (@lavados)**[2]

Red Hat Research Day, January 23, 2020

[1] imec-DistriNet, KU Leuven, [2] Graz University of Technology

Lessons from Spectre, Meltdown, and Foreshadow?

Spectre

Meltdown

Foreshadow

Lessons from Spectre, Meltdown, and Foreshadow?

Spectre

v1, v2, v4, v5, Spectre-BTB, Spectre-RSB, ret2spec, SGXPectre, SmotherSpectre, NetSpectre?

Meltdown

v3, v3.1, v3a, RDCL?

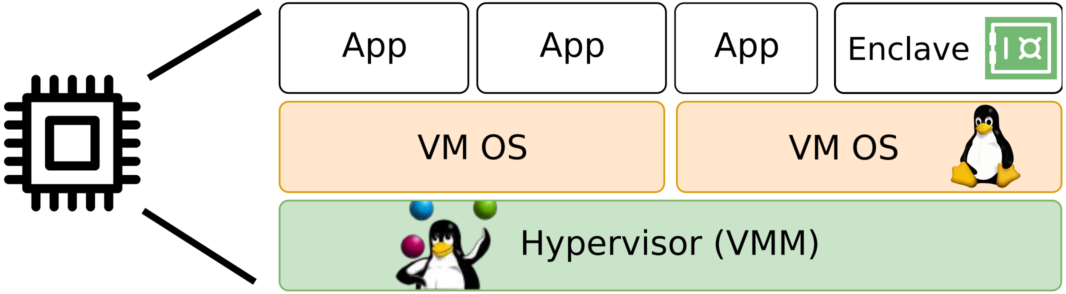
ZombieLoad, MDS?

Foreshadow

Foreshadow-NG, L1TF?

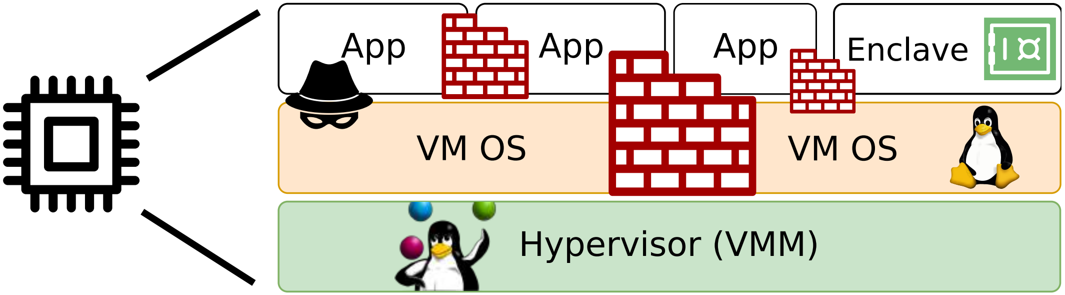
RIDL, Fallout?

Processor security: Hardware isolation mechanisms



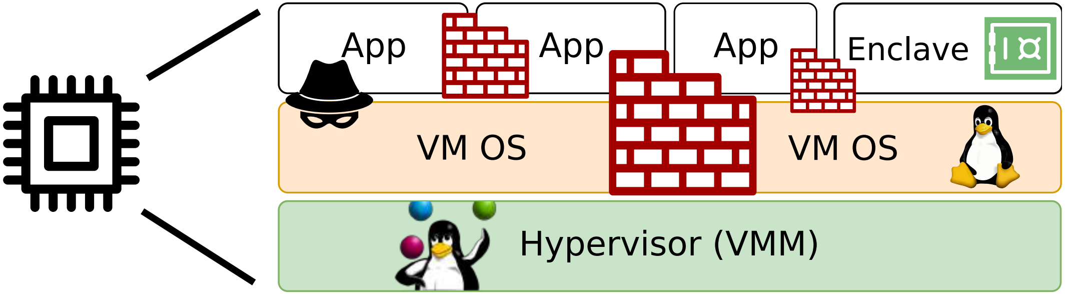
- Different software **protection domains:** user processes, virtual machines, enclaves

Processor security: Hardware isolation mechanisms



- Different software **protection domains:** user processes, virtual machines, enclaves
- CPU builds “walls” for **memory isolation** between applications and privilege levels

Processor security: Hardware isolation mechanisms



- Different software **protection domains:** user processes, virtual machines, enclaves
- CPU builds “walls” for **memory isolation** between applications and privilege levels

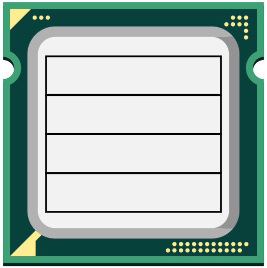
↔ Architectural protection walls permeate **microarchitectural side channels!**

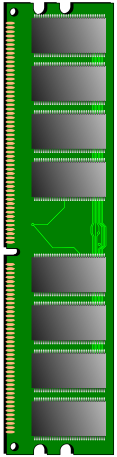
National Geographic

side channel
= obtaining meta-data and
deriving secrets from it
CHANGE MY MIND

CPU Cache

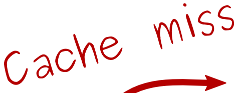
```
printf("%d", i);
printf("%d", i);
```

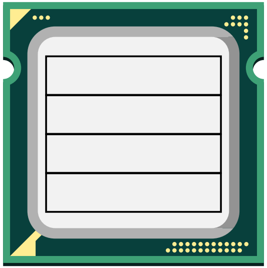


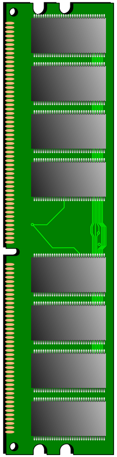


CPU Cache

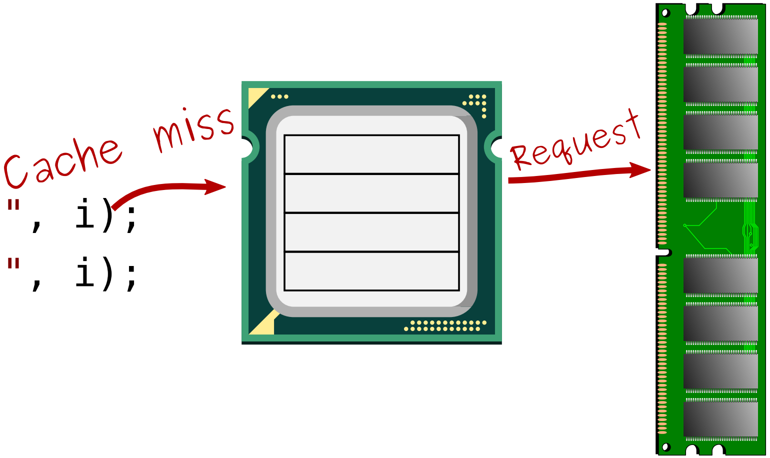
```
printf("%d", i);
printf("%d", i);
```





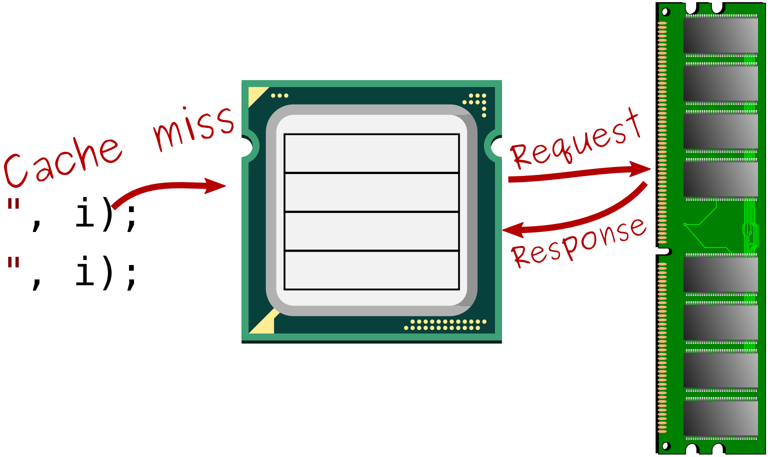


```
printf("%d", i);
printf("%d", i);
```



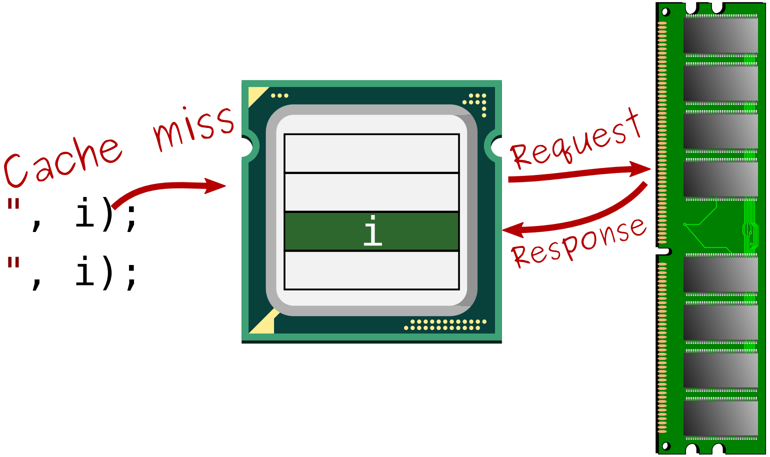
CPU Cache

```
printf("%d", i);
printf("%d", i);
```



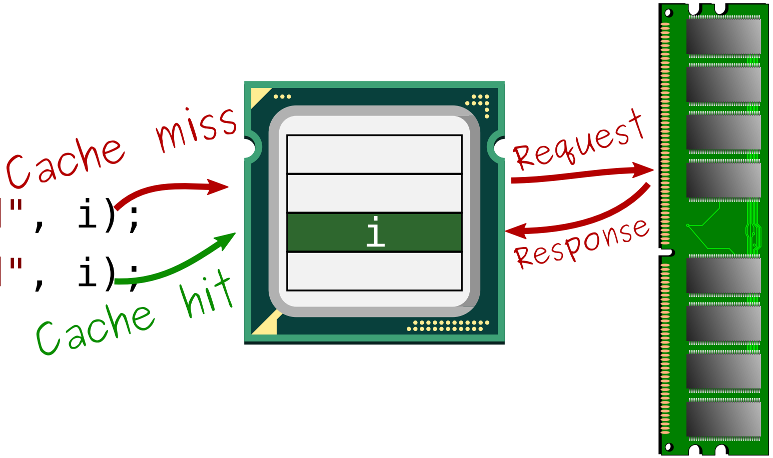
CPU Cache

```
printf("%d", i);
printf("%d", i);
```



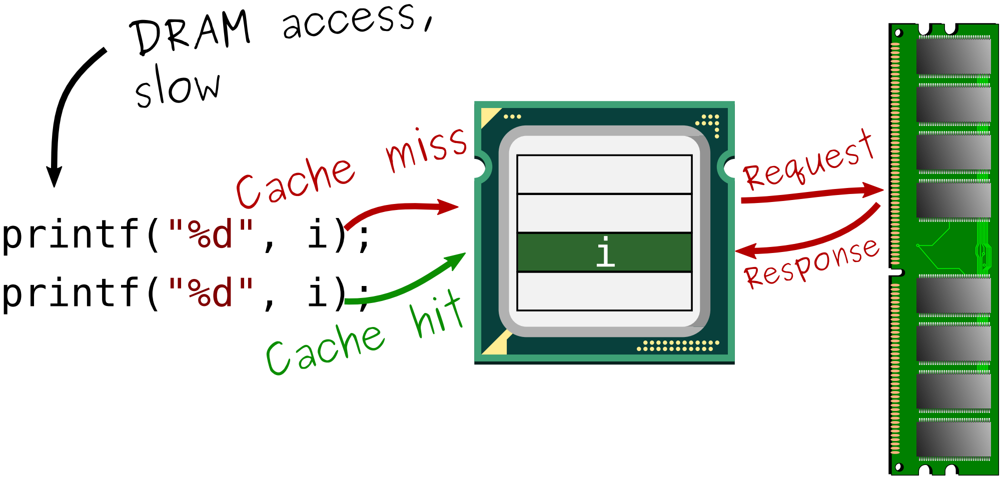
CPU Cache

```
printf("%d", i);
printf("%d", i);
```



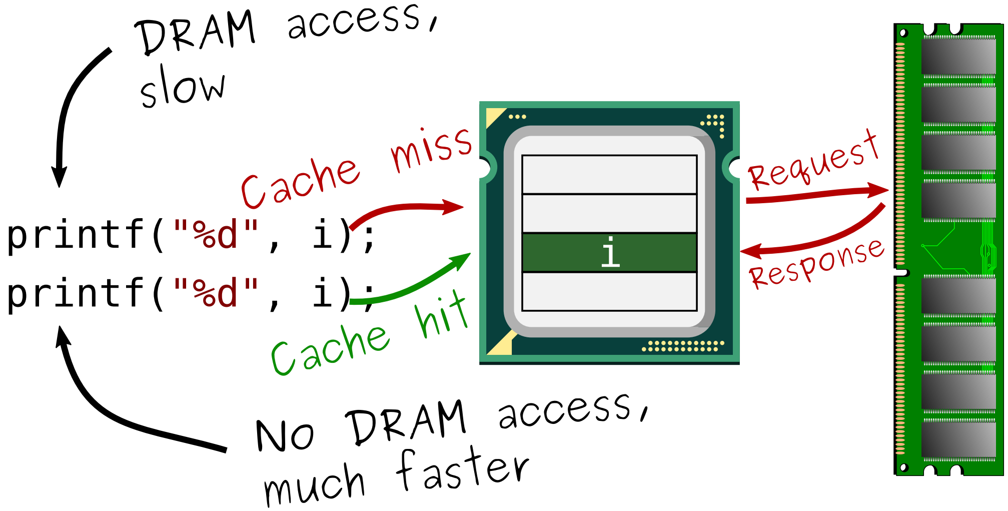
CPU Cache

```
printf("%d", i);
printf("%d", i);
```

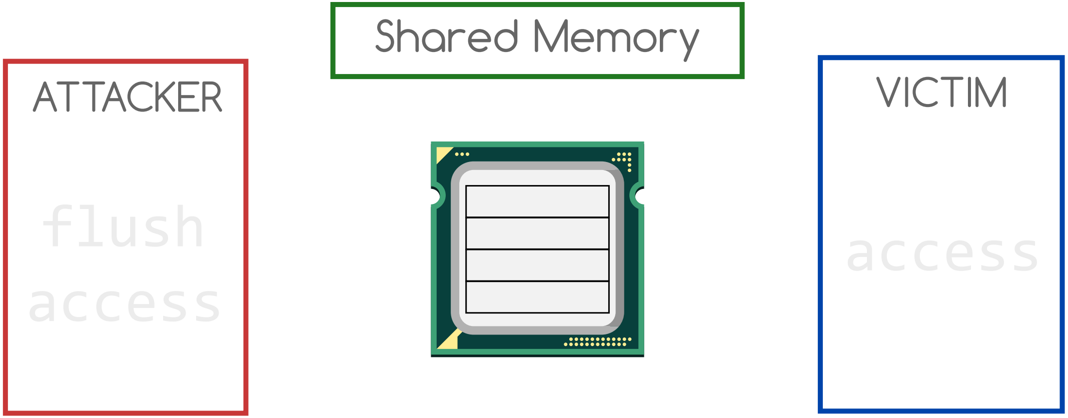


CPU Cache

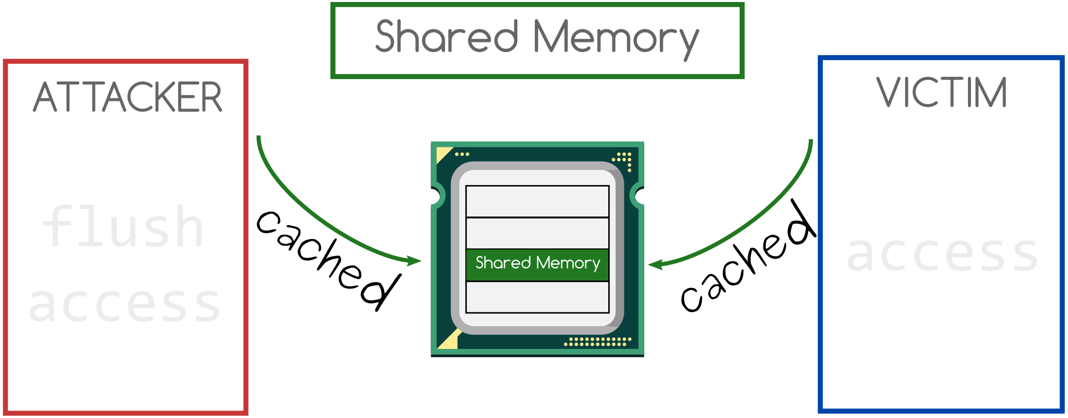
```
printf("%d", i);
printf("%d", i);
```



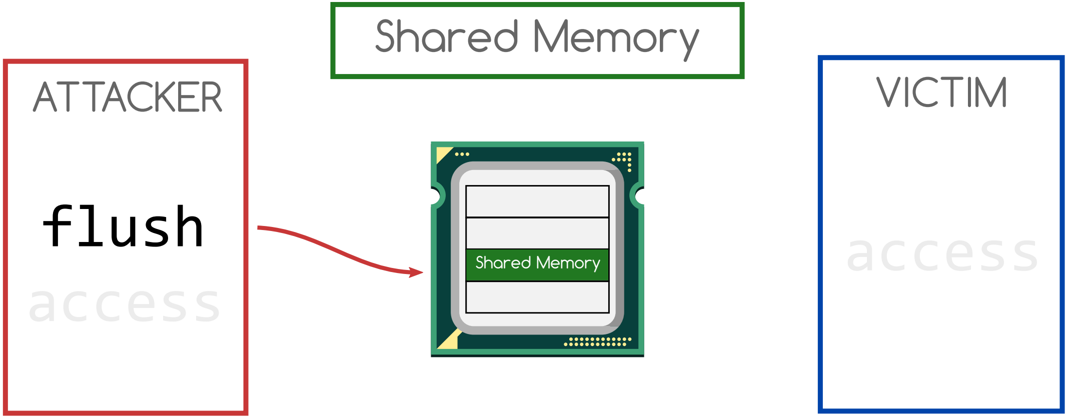
Flush+Reload



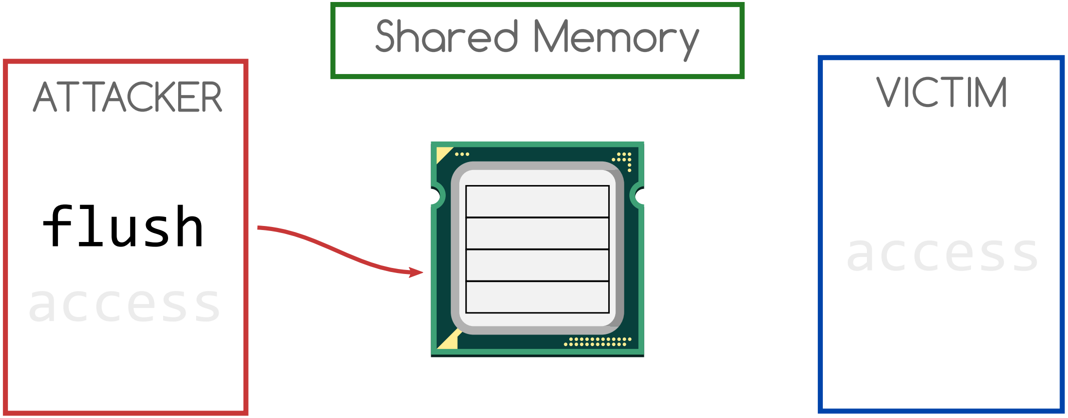
Flush+Reload



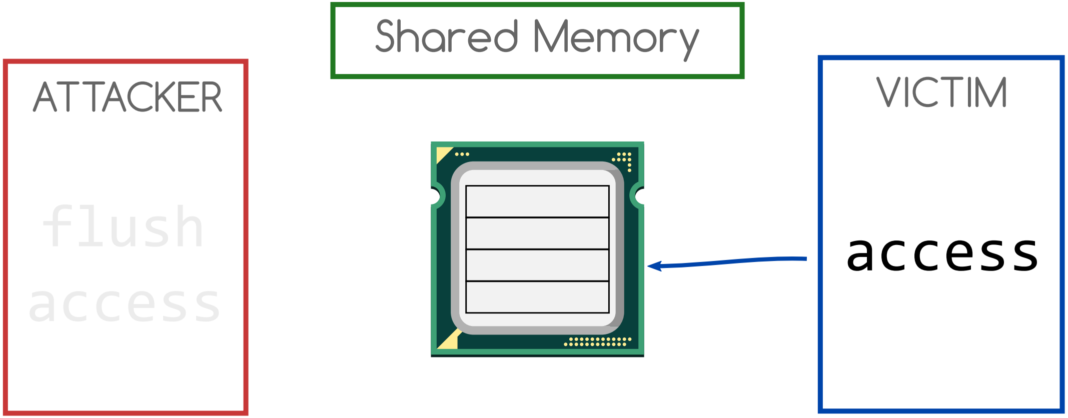
Flush+Reload



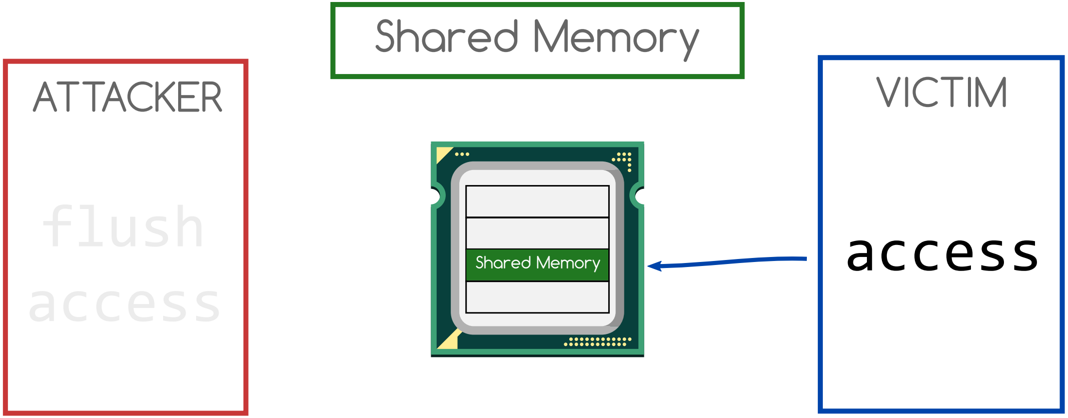
Flush+Reload



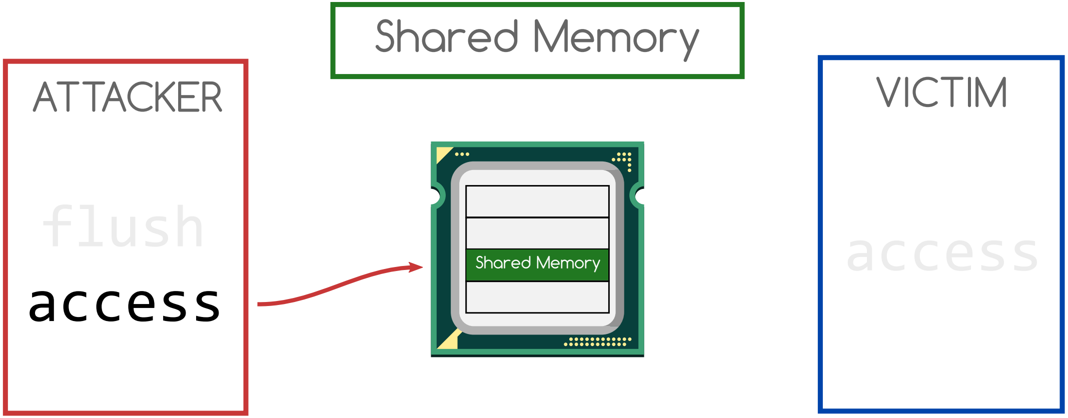
Flush+Reload



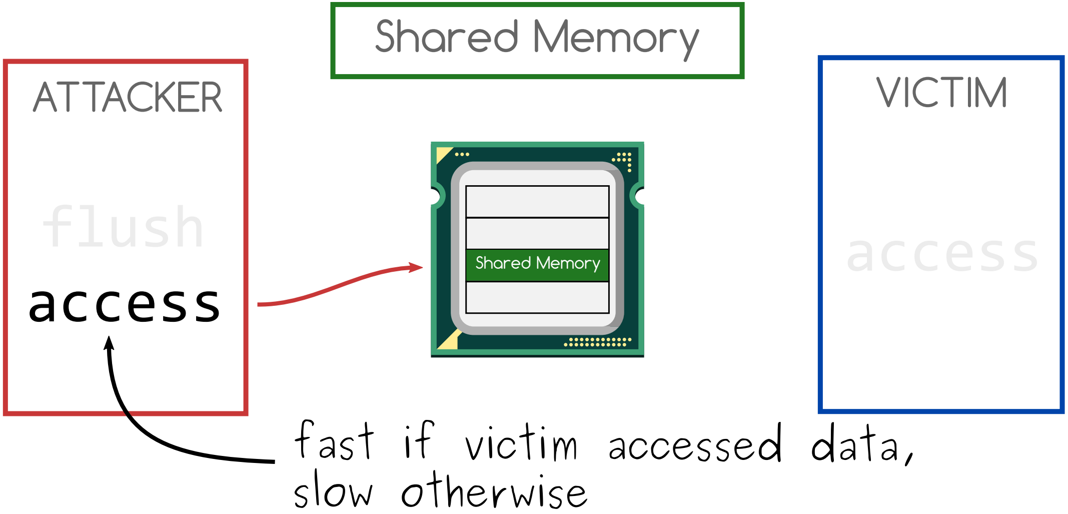
Flush+Reload



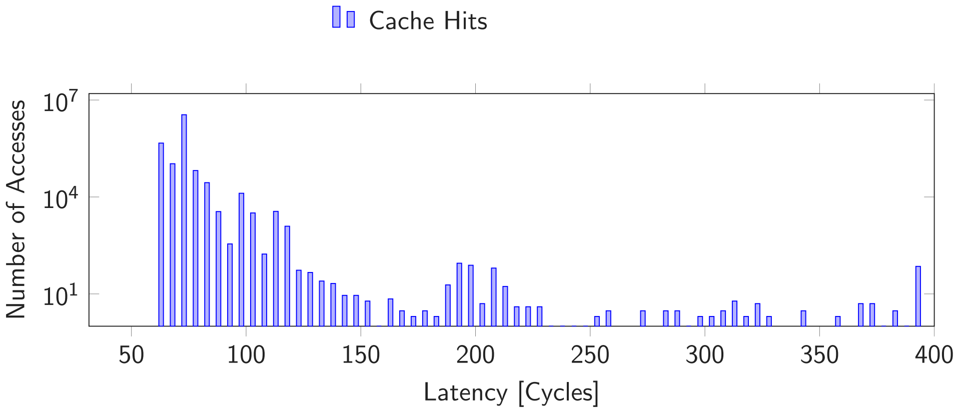
Flush+Reload



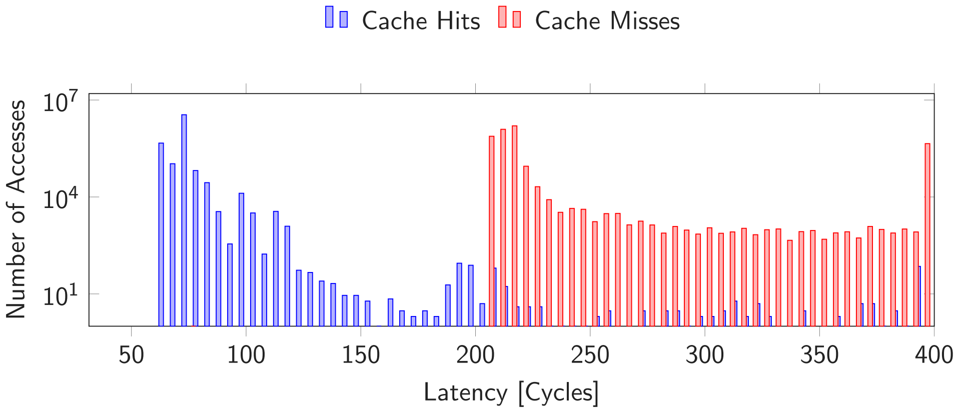
Flush+Reload

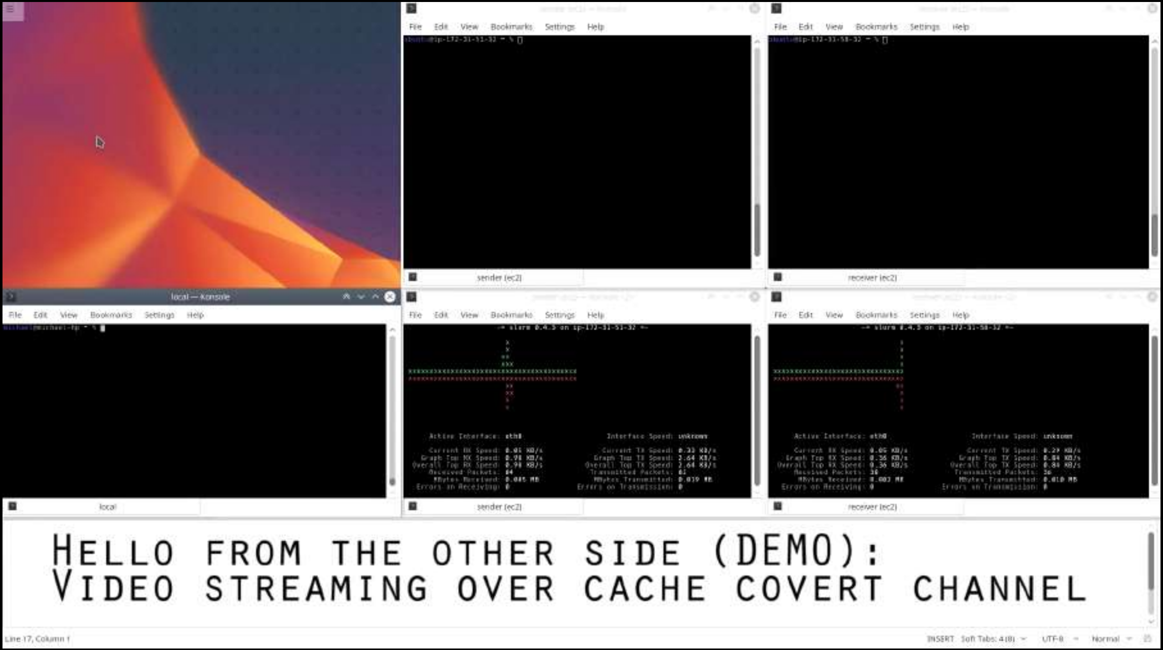


Memory Access Latency



Memory Access Latency





File Edit View Bookmarks Settings Help
sender (ec2)
receiver (ec2)
local — Konsole
local
Line 17, Column 1
INSERT Soft Tabs: 4 (8) UTF-8 Normal

```
ubuntu@ip-172-31-51-32 ~ %
```

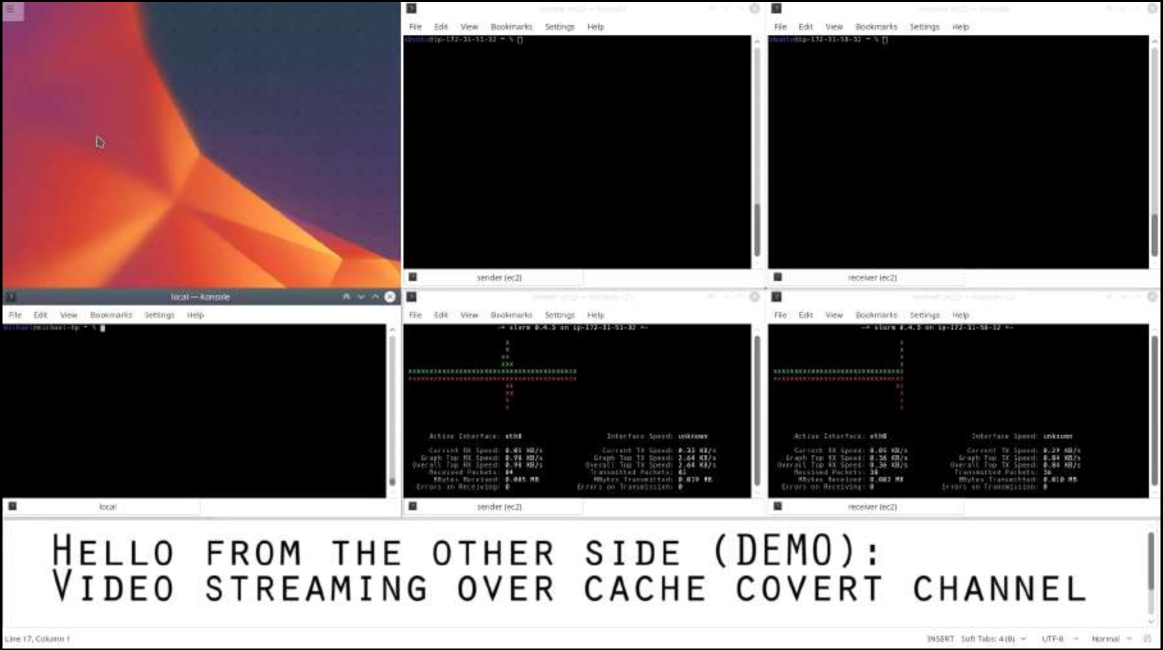
```
ubuntu@ip-172-31-58-32 ~ %
```

```
michael@michael-hp ~ %
```

```
                 -= slurm 0.4.3 on ip-172-31-51-32 =-

       Active Interface: eth0                  Interface Speed: unknown

      Current RX Speed: 0.05 KB/s            Current TX Speed: 0.33 KB/s
    Graph Top RX Speed: 0.98 KB/s          Graph Top TX Speed: 2.64 KB/s
  Overall Top RX Speed: 0.98 KB/s        Overall Top TX Speed: 2.64 KB/s
      Received Packets: 84               Transmitted Packets: 61
       MBytes Received: 0.005 MB          MBytes Transmitted: 0.019 MB
    Errors on Receiving: 0              Errors on Transmission: 0
```

```
                 -= slurm 0.4.3 on ip-172-31-58-32 =-

       Active Interface: eth0                  Interface Speed: unknown

      Current RX Speed: 0.06 KB/s            Current TX Speed: 0.29 KB/s
    Graph Top RX Speed: 0.36 KB/s          Graph Top TX Speed: 0.84 KB/s
  Overall Top RX Speed: 0.36 KB/s        Overall Top TX Speed: 0.84 KB/s
      Received Packets: 38               Transmitted Packets: 36
       MBytes Received: 0.003 MB          MBytes Transmitted: 0.010 MB
    Errors on Receiving: 0              Errors on Transmission: 0
```

Hello from the other side (DEMO): Video streaming over cache covert channel



```
ubuntu@ip-172-31-51-32 ~ %
```

```
ubuntu@ip-172-31-58-32 ~ %
```

```
michael@michael-hp ~ %
```

```
                 -= slurm 0.4.3 on ip-172-31-51-32 =-

   Active Interface: eth0                 Interface Speed: unknown

   Current RX Speed: 0.05 KB/s            Current TX Speed: 0.33 KB/s
  Graph Top RX Speed: 0.98 KB/s          Graph Top TX Speed: 2.64 KB/s
Overall Top RX Speed: 0.98 KB/s        Overall Top TX Speed: 2.64 KB/s
    Received Packets: 84              Transmitted Packets: 61
    MBytes Received: 0.005 MB          MBytes Transmitted: 0.019 MB
 Errors on Receiving: 0             Errors on Transmission: 0
```

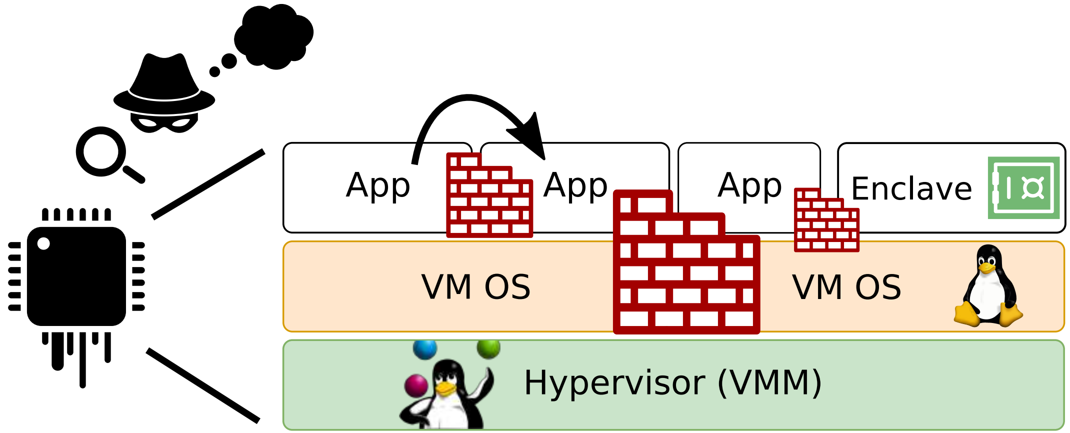
```
                 -= slurm 0.4.3 on ip-172-31-58-32 =-

   Active Interface: eth0                 Interface Speed: unknown

   Current RX Speed: 0.06 KB/s            Current TX Speed: 0.29 KB/s
  Graph Top RX Speed: 0.36 KB/s          Graph Top TX Speed: 0.84 KB/s
Overall Top RX Speed: 0.36 KB/s        Overall Top TX Speed: 0.84 KB/s
    Received Packets: 38              Transmitted Packets: 36
    MBytes Received: 0.003 MB          MBytes Transmitted: 0.010 MB
 Errors on Receiving: 0             Errors on Transmission: 0
```

local — Konsole
File Edit View Bookmarks Settings Help
sender (ec2)
receiver (ec2)
local
Line 17, Column 1
INSERT Soft Tabs: 4 (8) UTF-8 Normal

Hello from the other side (DEMO): Video streaming over cache covert channel

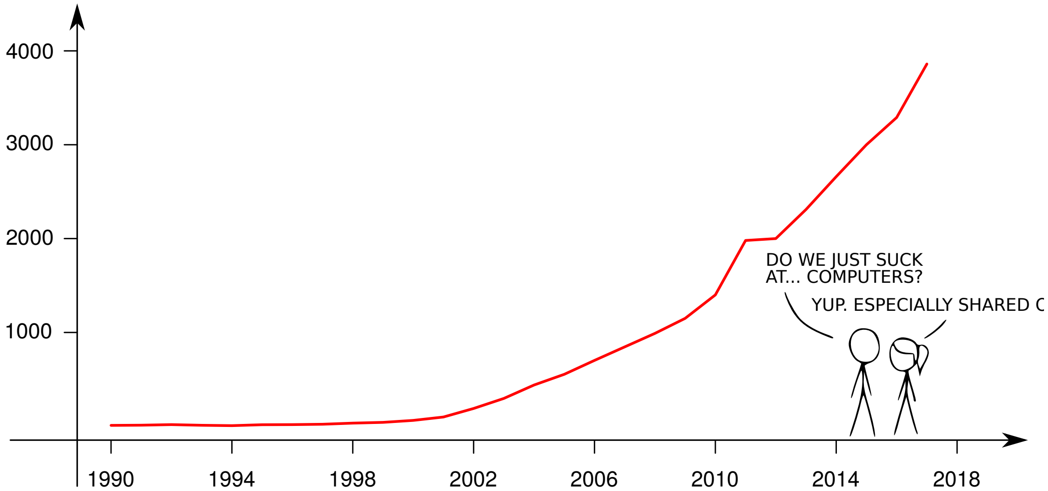
We can communicate across protection walls using microarchitectural side channels!

Leaky processors: Jumping over protection walls with side channels



SHARING IS NOT CARING
SHARING IS LOSING YOUR STUFF TO OTHERS
imgflip.com

Side-channel attacks are known for decades already – what's new?

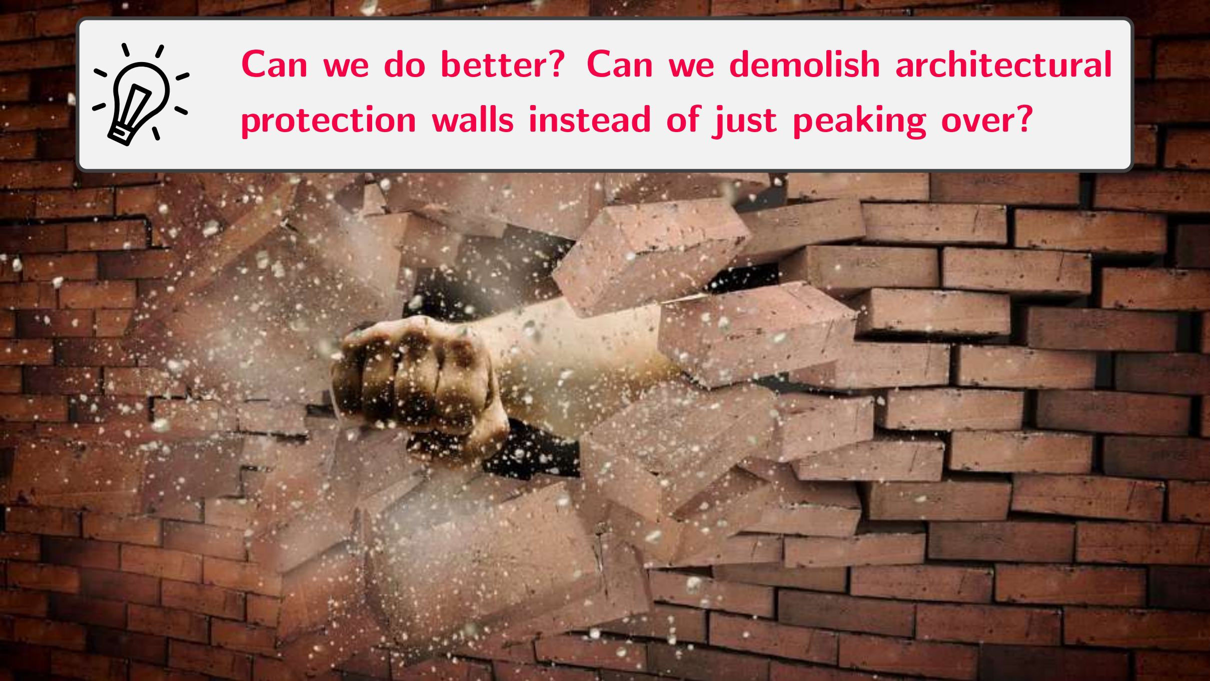


Based on `github.com/Pold87/academic-keyword-occurrence` and `xkcd.com/1938/`

Intel Analysis of Speculative Execution Side Channels

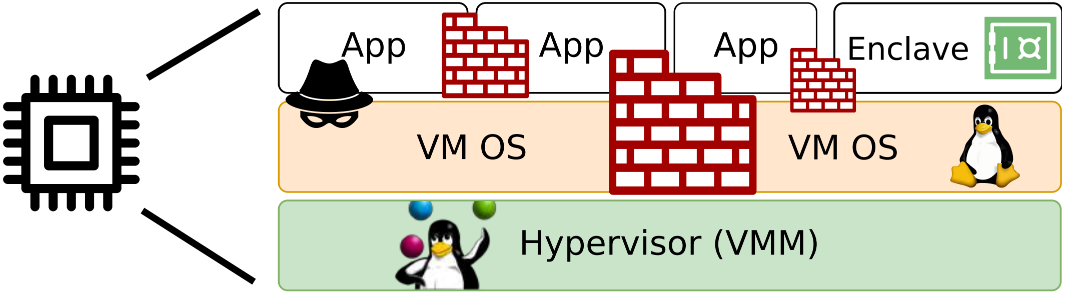
Intel Analysis of Speculative Execution Side Channels

White Paper

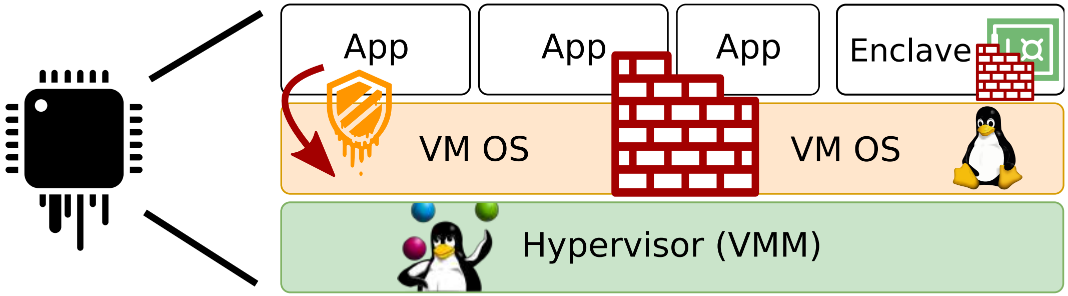


Can we do better? Can we demolish architectural protection walls instead of just peaking over?

Leaky processors: Breaking isolation mechanisms

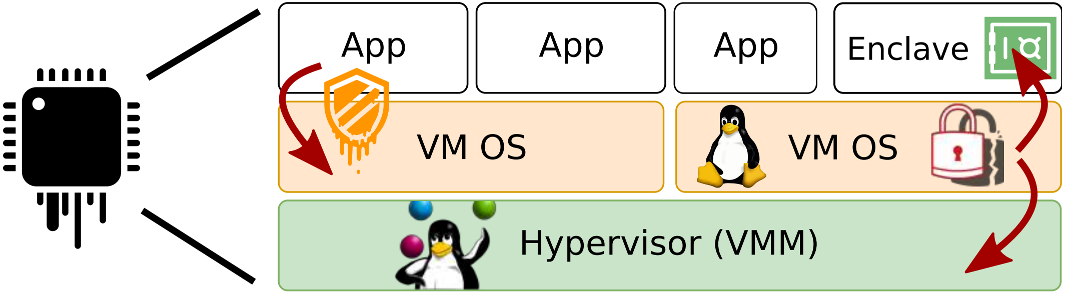


Leaky processors: Breaking isolation mechanisms



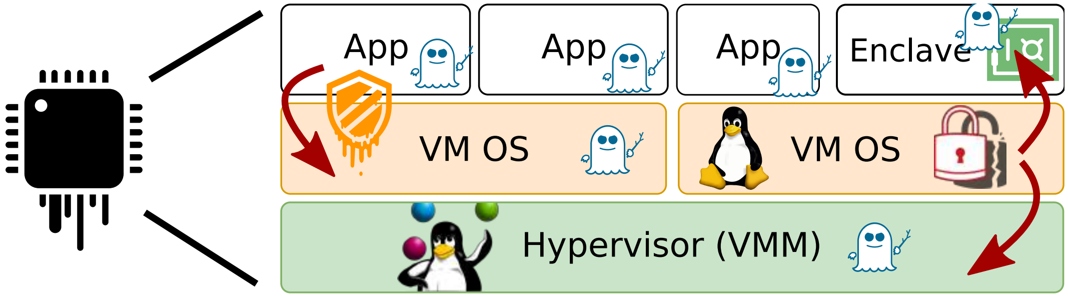
- Meltdown breaks user/kernel isolation

Leaky processors: Breaking isolation mechanisms



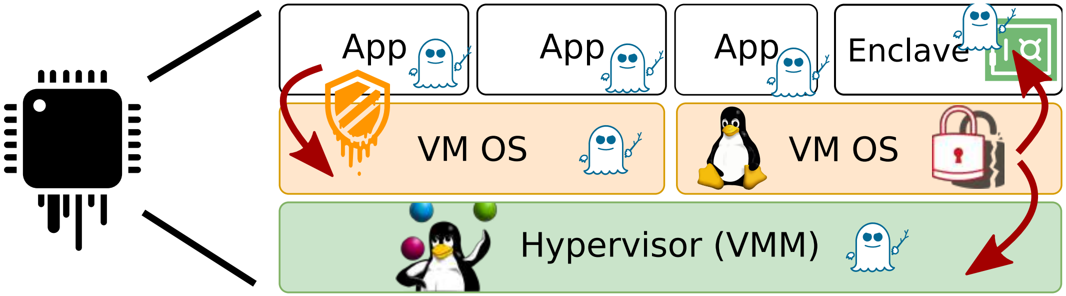
- Meltdown breaks user/kernel isolation
- Foreshadow breaks SGX enclave and virtual machine isolation

Leaky processors: Breaking isolation mechanisms



- Meltdown breaks user/kernel isolation
- Foreshadow breaks SGX enclave and virtual machine isolation
- Spectre breaks software-defined isolation on various levels

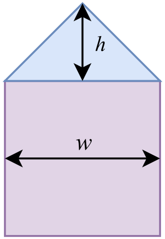
Leaky processors: Breaking isolation mechanisms



- Meltdown breaks user/kernel isolation
- Foreshadow breaks SGX enclave and virtual machine isolation
- Spectre breaks software-defined isolation on various levels
- . . . many more – but all exploit the same underlying insights!

WHAT IF I TOLD YOU
YOU CAN CHANGE RULES MID-GAME

Out-of-order and speculative execution

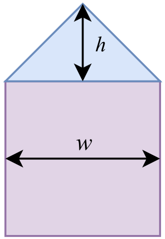


```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

Key **discrepancy:**

- Programmers write sequential instructions

Out-of-order and speculative execution



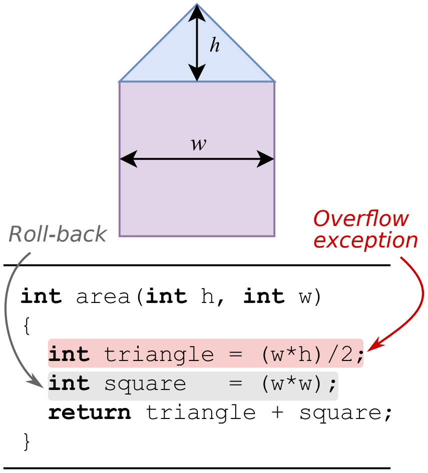
```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

Key **discrepancy:**

- Programmers write sequential instructions
- Modern CPUs are inherently parallel

$\Rightarrow$ *Execute instructions ahead of time*

Out-of-order and speculative execution



```
int area(int h, int w)
{
  int triangle = (w*h)/2;
  int square   = (w*w);
  return triangle + square;
}
```

Key **discrepancy:**

- Programmers write sequential instructions
- Modern CPUs are inherently parallel

⇒ *Execute instructions ahead of time*

Best-effort: What if triangle fails?

→ Commit in-order, roll-back square

. . . But side channels may leave traces (!)

Transient-execution attacks: Welcome to the world of fun!

CPU executes ahead of time in transient world

- Success → *commit* results to normal world ☺
- Fail → *discard* results, compute again in normal world ☹

Transient-execution attacks: Welcome to the world of fun!

Key finding of 2018

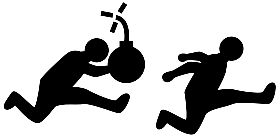
$\Rightarrow$ *Transmit secrets from transient to normal world*

Transient-execution attacks: Welcome to the world of fun!

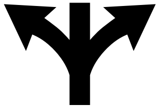
Key finding of 2018

⇒ *Transmit secrets from transient to normal world*

Transient world (microarchitecture) may temp bypass architectural software intentions:



Delayed exception handling



Control flow prediction

Transient-execution attacks: Welcome to the world of fun!

Key finding of 2018

$\Rightarrow$ *Transmit secrets from transient to normal world*

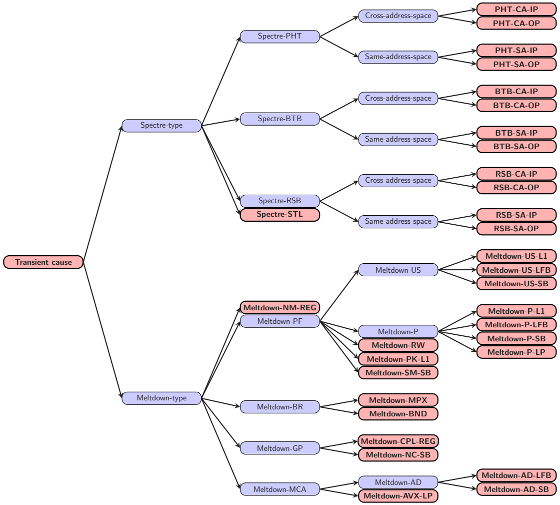
Transient world (microarchitecture) may temp bypass architectural software intentions:

CPU access control bypass

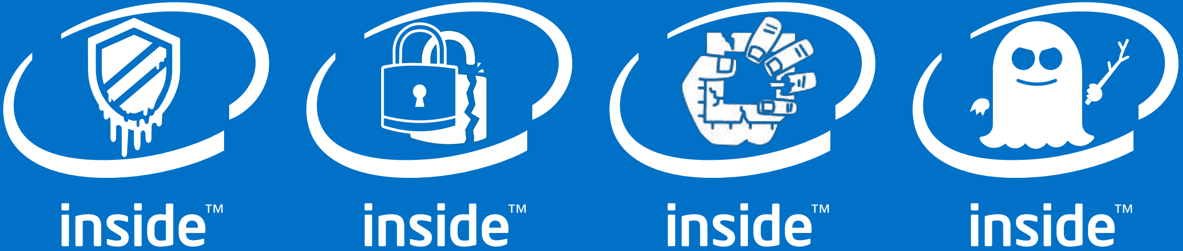
Speculative buffer overflow/ROP

The transient-execution zoo

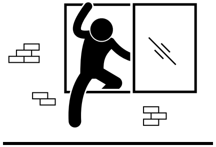
https://transient.fail



Canella et al. “A systematic evaluation of transient execution attacks and defenses”, USENIX Security 2019


inside™
inside™
inside™
inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

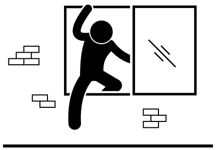
Listing 1: x86 assembly

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
        uint8_t *oracle,
        uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

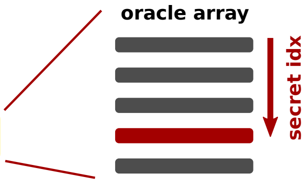
Transient out-of-order window

Listing 1: x86 assembly.

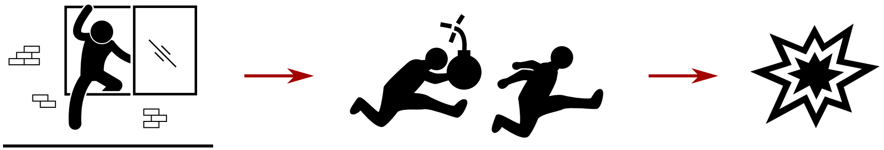
```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
        uint8_t *oracle,
        uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```



Meltdown: Transiently encoding unauthorized memory



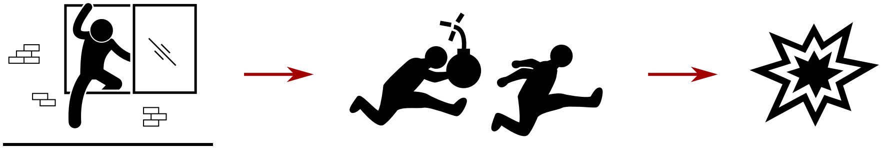
Listing 1: x86 assembly.

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
    uint8_t *oracle,
    uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

Meltdown: Transiently encoding unauthorized memory



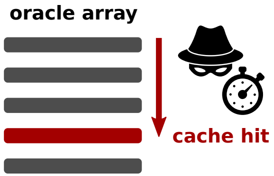
Unauthorized access | Transient out-of-order window | **Exception handler**

Listing 1: x86 assembly.

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
      uint8_t *oracle,
      uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

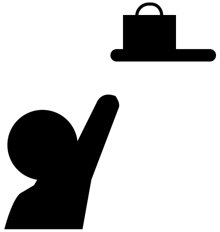


Recovering from a Meltdown: Re-building protection walls?

Mitigating Meltdown: Unmap kernel addresses from user space

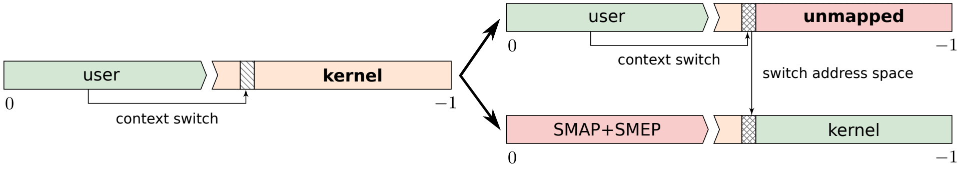
- OS software fix for faulty hardware ($\leftrightarrow$ future CPUs)

Mitigating Meltdown: Unmap kernel addresses from user space



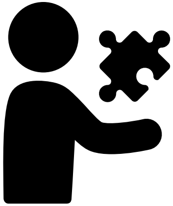
- OS software fix for faulty hardware ($\leftrightarrow$ future CPUs)
- Unmap kernel from user *virtual address space*

$\rightarrow$ Unauthorized physical addresses out-of-reach (˜cookie jar)



Gruss et al. "KASLR is dead: Long live KASLR", ESSoS 2017

Problem Solved?

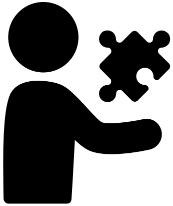


- Meltdown fully mitigated in software

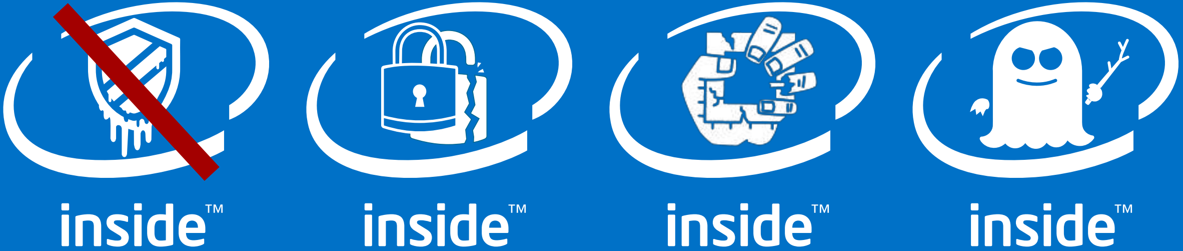
Problem Solved?

- Meltdown fully mitigated in software
- Problem seemed to be solved
- No attack surface left

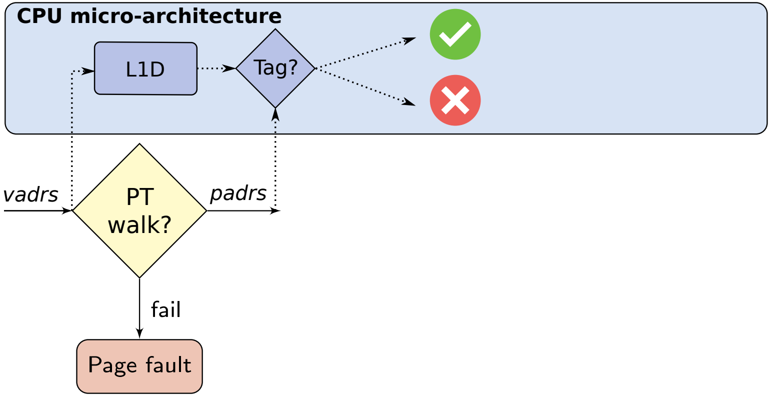
Problem Solved?



- Meltdown fully mitigated in software
- Problem seemed to be solved
- No attack surface left
- That is what everyone thought

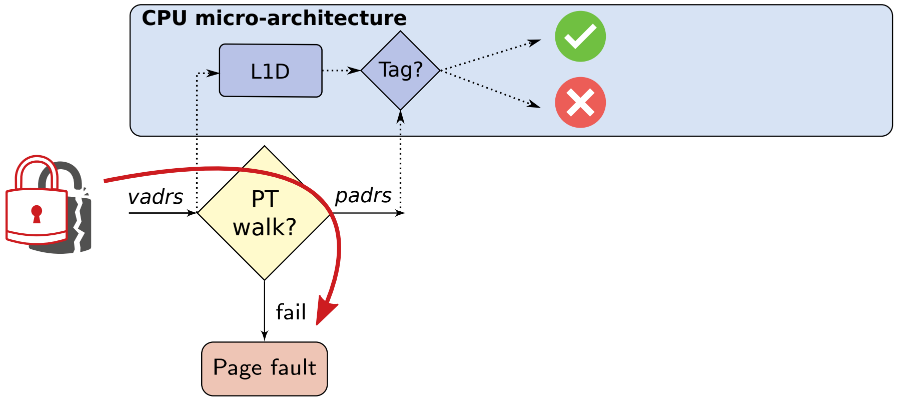

inside™
inside™
inside™
inside™

Foreshadow-NG: Breaking the virtual memory abstraction



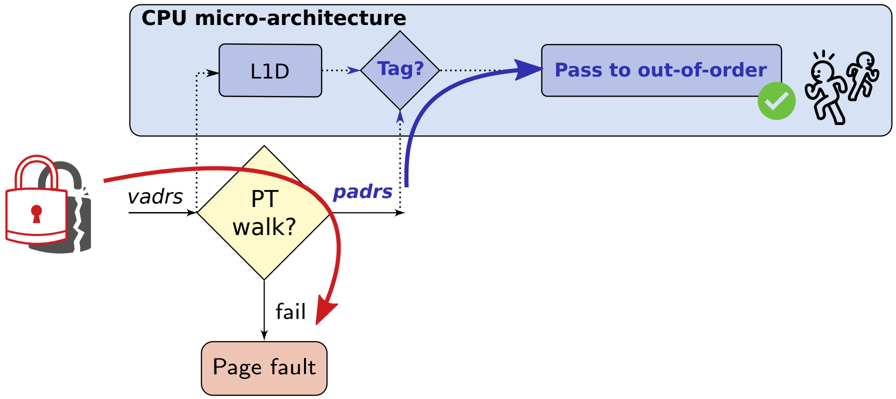
L1 cache design: Virtually-indexed, physically-tagged

Foreshadow-NG: Breaking the virtual memory abstraction



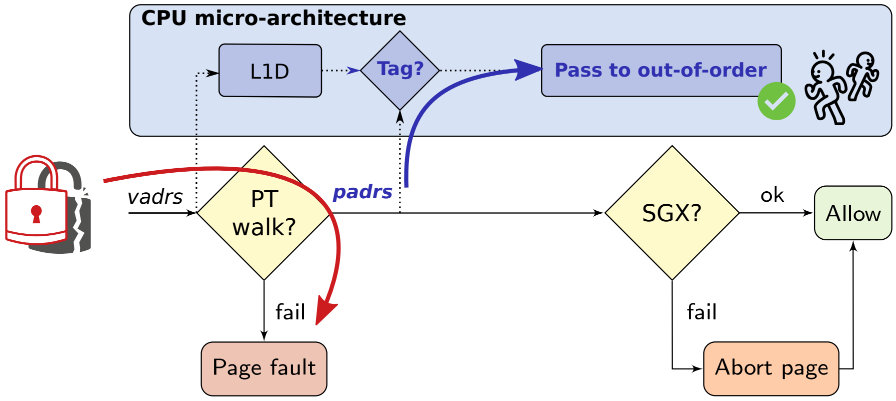
Page fault: Early-out address translation

Foreshadow-NG: Breaking the virtual memory abstraction



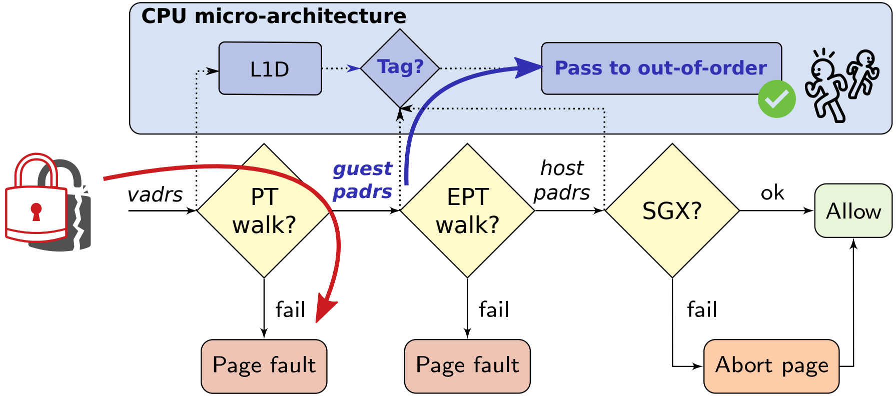
L1-Terminal Fault: match *unmapped physical address* (!)

Foreshadow-NG: Breaking the virtual memory abstraction



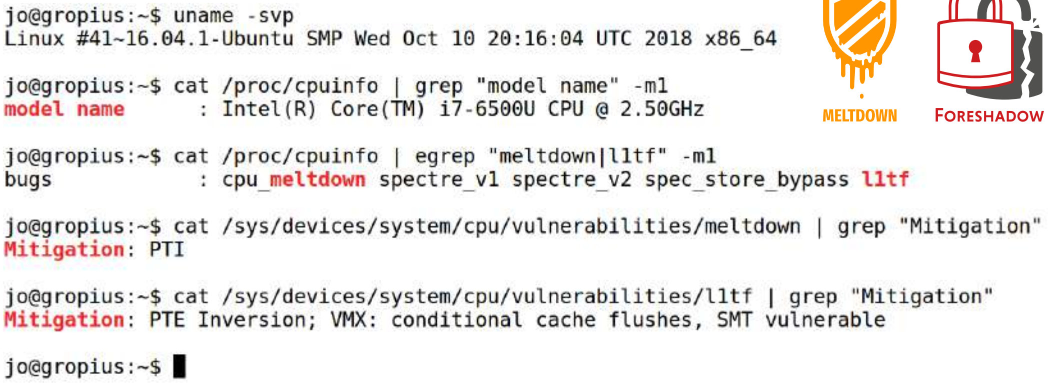
Foreshadow-SGX: bypass enclave isolation

Foreshadow-NG: Breaking the virtual memory abstraction



Foreshadow-VMM: bypass virtual machine isolation(!)

Mitigating Foreshadow/L1TF: Hardware-software cooperation



```
jo@gropius:~$ uname -svp
Linux #41~16.04.1-Ubuntu SMP Wed Oct 10 20:16:04 UTC 2018 x86_64

jo@gropius:~$ cat /proc/cpuinfo | grep "model name" -m1
model name      : Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz

jo@gropius:~$ cat /proc/cpuinfo | egrep "meltdown|l1tf" -m1
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf

jo@gropius:~$ cat /sys/devices/system/cpu/vulnerabilities/meltdown | grep "Mitigation"
Mitigation: PTI

jo@gropius:~$ cat /sys/devices/system/cpu/vulnerabilities/l1tf | grep "Mitigation"
Mitigation: PTE Inversion; VMX: conditional cache flushes, SMT vulnerable

jo@gropius:~$ █
```

Generalization – Lessons from Foreshadow

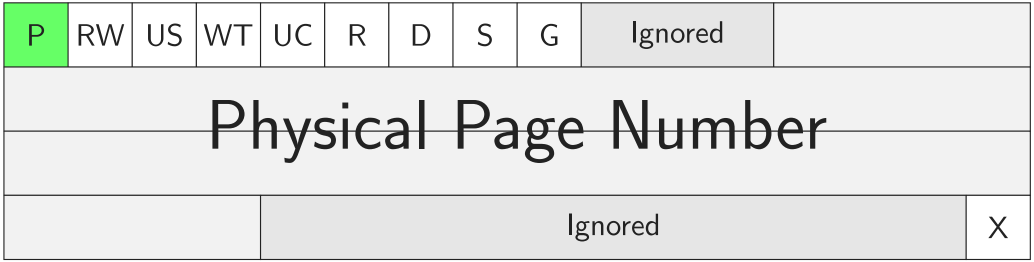
- Meltdown is a whole category of vulnerabilities

Generalization – Lessons from Foreshadow

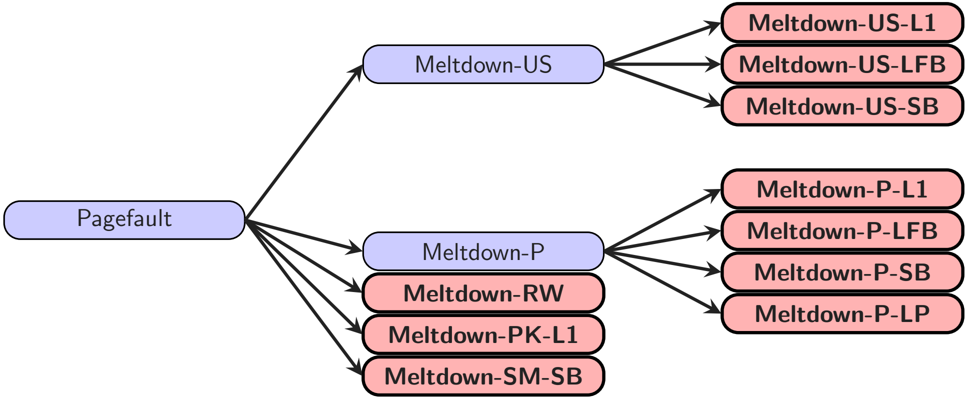
- Meltdown is a whole category of vulnerabilities
- Not only the user-accessible check

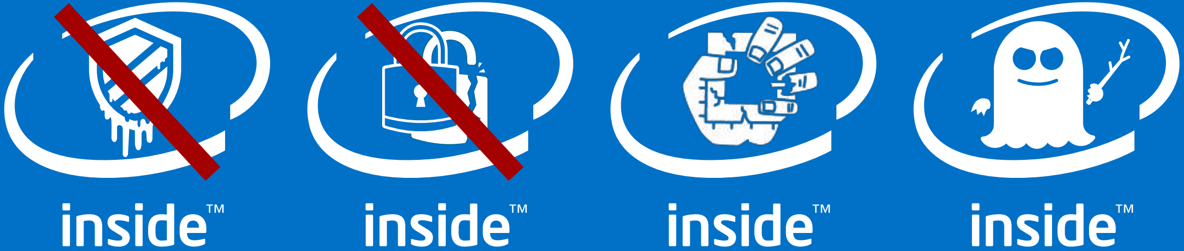
Generalization – Lessons from Foreshadow

- Meltdown is a whole category of vulnerabilities
- Not only the user-accessible check
- There are many more page table bits and exception types...

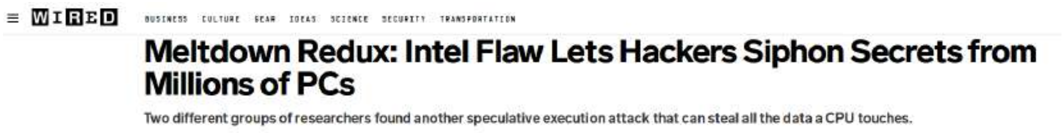


Meltdown subtree: Exploiting page-table bits



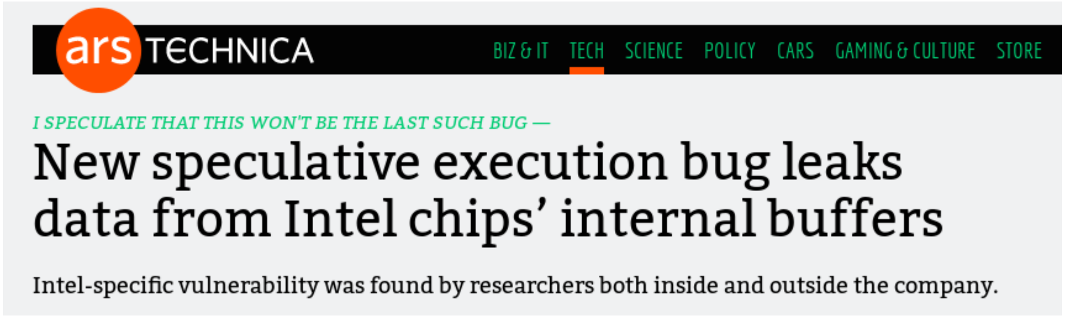

inside™
inside™
inside™
inside™

May 2019: Meltdown redux



Meltdown Redux: Intel Flaw Lets Hackers Siphon Secrets from Millions of PCs

Two different groups of researchers found another speculative execution attack that can steal all the data a CPU touches.



I SPECULATE THAT THIS WON'T BE THE LAST SUCH BUG —

New speculative execution bug leaks data from Intel chips' internal buffers

Intel-specific vulnerability was found by researchers both inside and outside the company.

Microarchitectural data sampling: RIDL, ZombieLoad, Fallout

- May 2019: 3 new Meltdown-type attacks
- Leakage from: line-fill buffer, store buffer, load ports

Microarchitectural data sampling: RIDL, ZombieLoad, Fallout

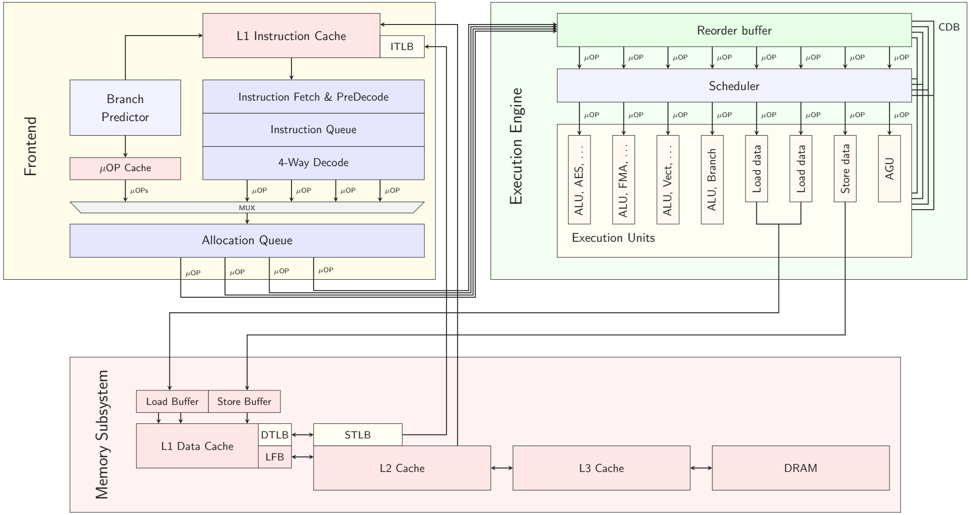
- May 2019: 3 new Meltdown-type attacks
- Leakage from: line-fill buffer, store buffer, load ports
- **Key take-aways:**
  1. Leakage from various intermediate buffers ($\supset$ L1D)
  2. Transient execution through microcode assists ($\supset$ exceptions)

Microarchitectural data sampling: RIDL, ZombieLoad, Fallout

- May 2019: 3 new Meltdown-type attacks
- Leakage from: line-fill buffer, store buffer, load ports
- **Key take-aways:**
  1. Leakage from various intermediate buffers ($\supset$ L1D)
  2. Transient execution through microcode assists ($\supset$ exceptions)

There is no noise. Noise is just someone else's data

MDS take-away 1: Microarchitectural buffers



MDS take-away 2: Microcode assists

- Optimization: only implement fast-path in silicon
- More complex edge cases (slow-path) in microcode

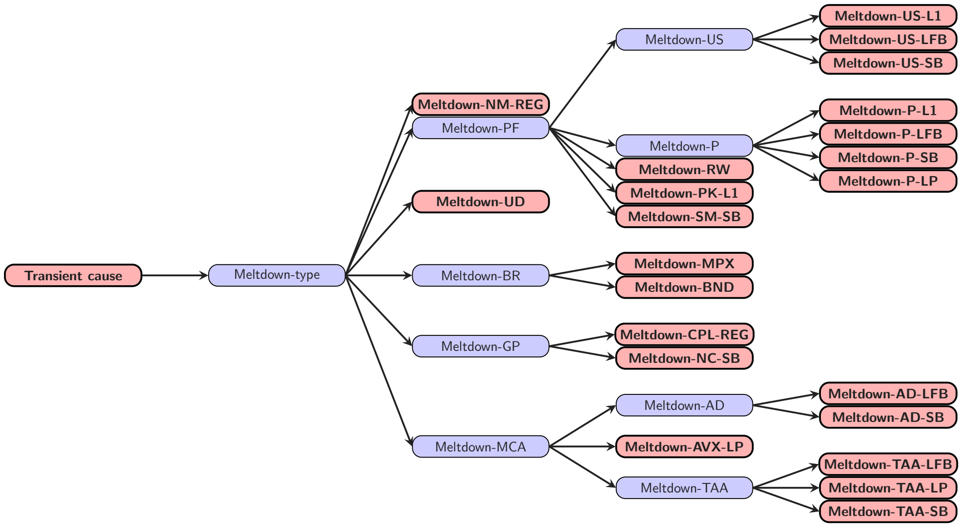
MDS take-away 2: Microcode assists

- Optimization: only implement fast-path in silicon
- More complex edge cases (slow-path) in microcode
- Need help? Re-issue the load with a **microcode assist**
    - assist == "microarchitectural fault"

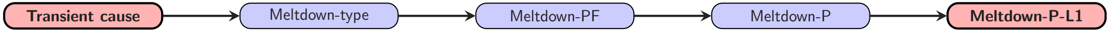
MDS take-away 2: Microcode assists

- Optimization: only implement fast-path in silicon
- More complex edge cases (slow-path) in microcode
- Need help? Re-issue the load with a **microcode assist**
  - assist == "microarchitectural fault"
- Example: setting A/D bits in the page table walk
  - Likely many more!

Extended Meltdown tree with microcode assists



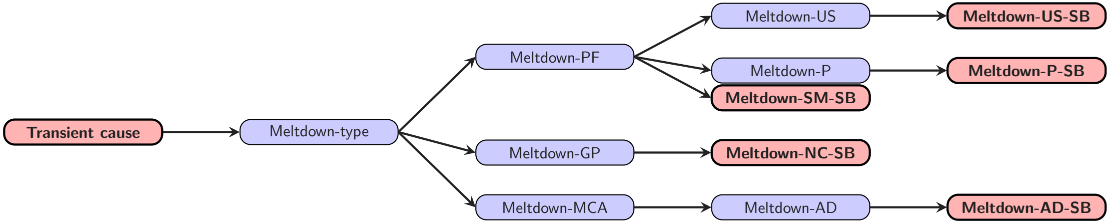
2018 era: Depth-first search (e.g., Foreshadow/L1TF)



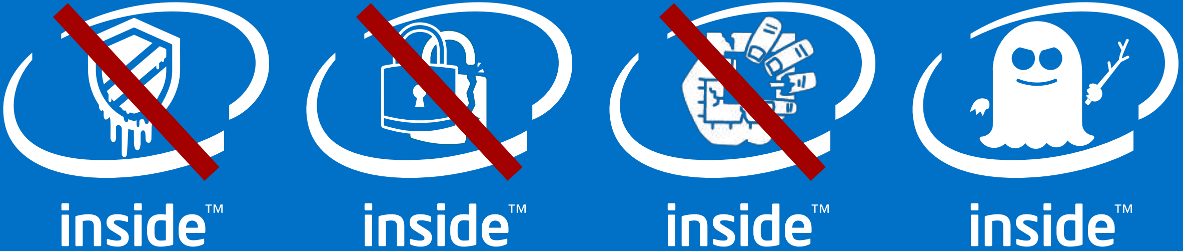
- Meltdown is a category of attacks and not a single instance or bug
- Systematic analysis **(tree search)** revealed several overlooked variants

Canella et al. "A Systematic Evaluation of Transient Execution Attacks and Defenses", USENIX Security 2019.

2019 era: Breadth-first search (e.g., Fallout)



Not "just another buffer", include systematic **fault-type analysis**


inside™
inside™
inside™
inside™

Conclusions and take-away

Update your systems! (+ disable HyperThreading)

Conclusions and take-away

Update your systems! (+ disable HyperThreading)

⇒ New emerging and powerful class of **transient-execution** attacks

⇒ Importance of fundamental **side-channel research**

⇒ Security **cross-cuts** the system stack: hardware, hypervisor, kernel, compiler, application

Leaky Processors

Lessons from Spectre, Meltdown, and Foreshadow

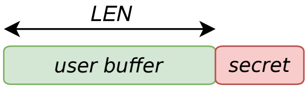
Jo Van Bulck (@jovanbulck)[1], **Daniel Gruss (@lavados)**[2]

Red Hat Research Day, January 23, 2020

[1] imec-DistriNet, KU Leuven, [2] Graz University of Technology

Appendix

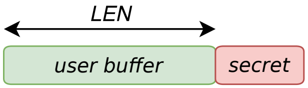
Spectre v1: Speculative buffer over-read



```
if (idx < LEN)
{
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* never access out-of-bounds memory

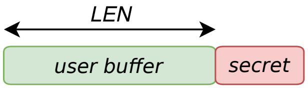
Spectre v1: Speculative buffer over-read



```
if (idx < LEN)
{
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* never access out-of-bounds memory
- Branch can be mistrained to speculatively (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**

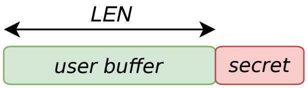
Spectre v1: Speculative buffer over-read



```
if (idx < LEN)
{
  asm("lfence\n\t");
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* never access out-of-bounds memory
- Branch can be mistrained to speculatively (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...

Spectre v1: Speculative buffer over-read



```
if (idx < LEN)
{
  asm("lfence\n\t");
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* never access out-of-bounds memory
- Branch can be mistrained to speculatively (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...
- Huge manual, error-prone effort. . .