
SBA
Research

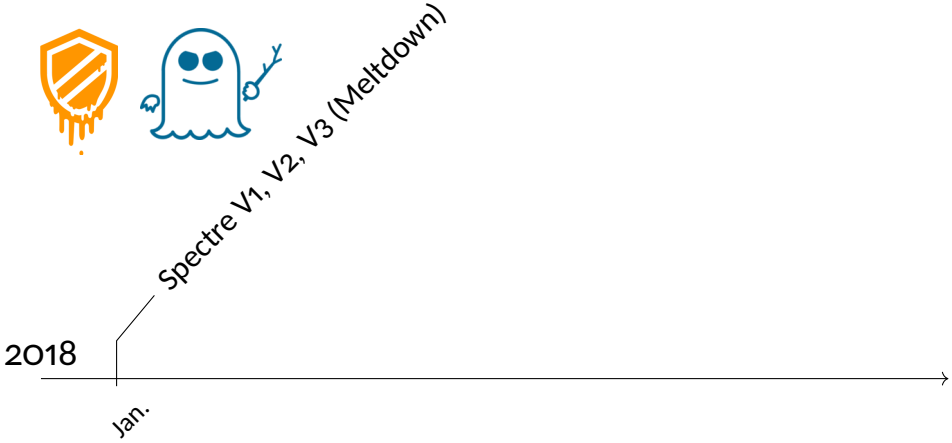
CPU Vulnerabilities

Where are we now?

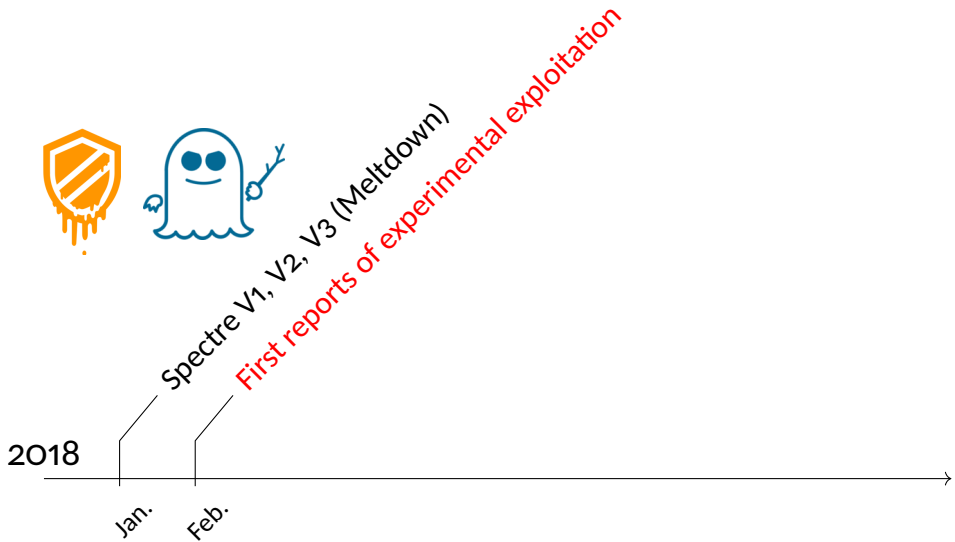
Manuel Wiesinger
mwiesinger@sba-research.at

SBA Research

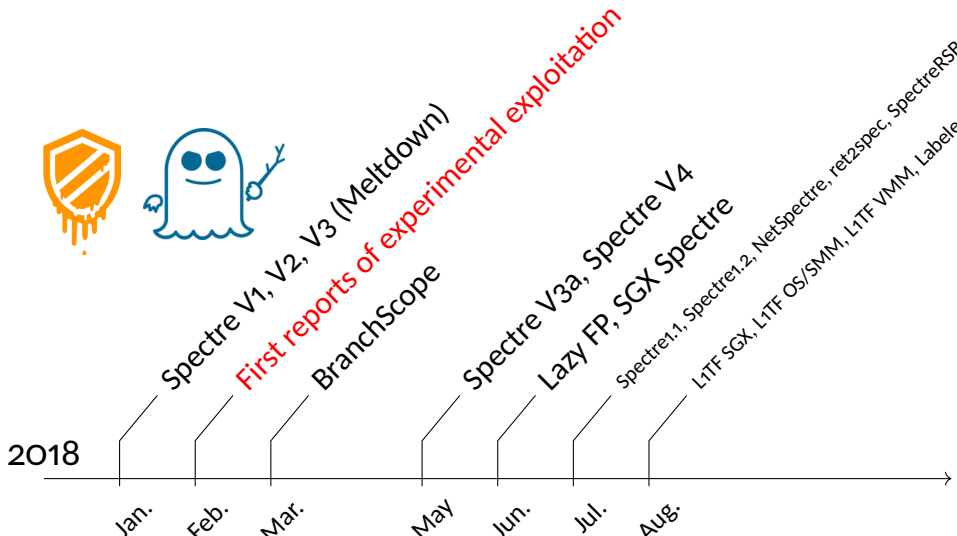
What happened so far?



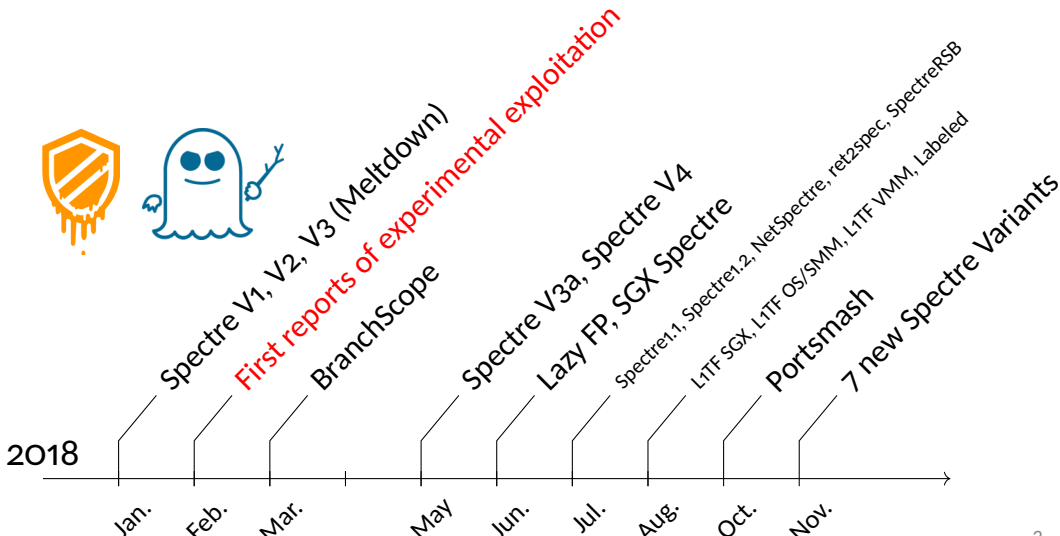
What happened so far?



What happened so far?



What happened so far?



Impact

- Allows data extraction from *arbitrary* local memory (!)

Impact

- Allows data extraction from *arbitrary* local memory (!)
- Exploitable from JavaScript environments (websites!!)

Impact

- Allows data extraction from *arbitrary* local memory (!)
- Exploitable from JavaScript environments (websites!!)
- Via the network (!!!)

Impact

- Allows data extraction from *arbitrary* local memory (!)
- Exploitable from JavaScript environments (websites!!)
- Via the network (!!!)

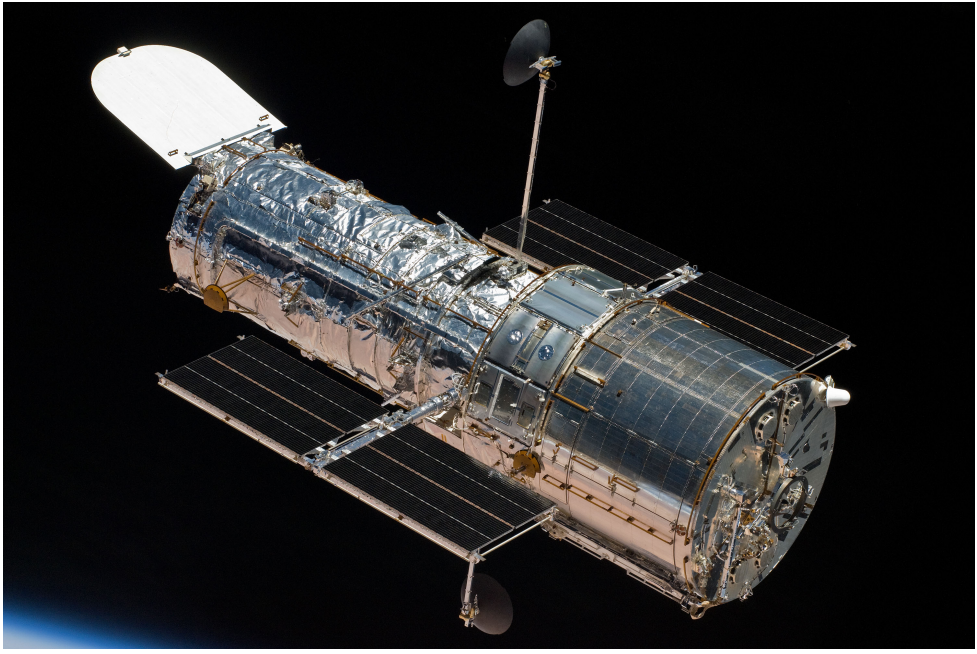
Who can feel safe?

- Nobody using computers built after 1995.
  - Any CPU manufacturer
  - Any operating system
  - Any Device type
- Don’t trust the memory!

How can we fix CPU vulnerabilities ?

Can software fix this?

Do we need to throw all our computers away?



Attack Limitations

- Difficult — conventional attacks typically easier

Attack Limitations

- Difficult — conventional attacks typically easier
- Via the network
  - Works only under laboratory conditions
  - Slow (15 bit / hour)

Attack Limitations

- Difficult — conventional attacks typically easier
- Via the network
  - Works only under laboratory conditions
  - Slow (15 bit / hour) — Still: extract a 256 bit key in $\sim$ 17 hours

Attack Limitations

- Difficult — conventional attacks typically easier
- Via the network
  - Works only under laboratory conditions
  - Slow (15 bit / hour) — Still: extract a 256 bit key in $\sim$ 17 hours
- Mitigations on the way
  - Partly already deployed via microcode and OS upgrades

Fixes for CPU-Vulnerabilities

Name	CVE	Aliases	CVSS	Impact	Fix available
Spectre V1	2017-5753	Bounds Check Bypass	5.6	Memory	Microcode/Browser/OS
Spectre V2	2017-5715		5.6	Memory	Microcode/Compiler ?
Spectre V3	2017-5754	Meltdown	5.6	Kernel memory	OS
Spectre V3a	2018-3640	Spectre V3a (RSRE)	5.6	Register data	Microcode?
Spectre V4	2018-3639	Speculative Store Bypass (SSB)	5.5	Memory	OS
Spectre V5	N/A	ret2spec	5.5	Memory	Browser?
SpectreRSB	N/A		N/A	Memory	OS
Lazy FP	2018-3665		5.6	Registers	OS
Spectre1.1	2018-3693		5.6	Kernel memory	OS
Spectre1.2	N/A		N/A	Kerslidesmory	OS
L1TF: SGX	2018-3615	Foreshadow (SGX)	6.4	SGX enclaves	Microcode
L1TF: OS/SMM	2018-3620	Foreshadow-NG (OS)	5.6	Kernel memory	Microcode
L1TF: VMM	2018-3646	Foreshadow-NG (VMM)	5.6	Kernel memory	Microcode
BranchScope	2018-9056		5.6	VM memory	Microcode?
SGXPectre	N/A		N/A	SGX enclaves	Microcode?
NetSpectre	N/A		N/A	Remote memory	OS?
TLBleed	N/A		N/A		Microcode?
PortSmash	2018-5407		4.8		Microcode?
7 new!	N/A		N/A		Microcode?/OS?

We do not know what else is out there!

How do CPU vulnerabilities work?

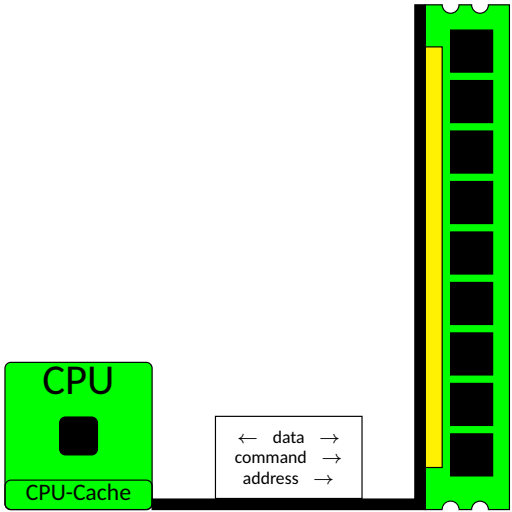
Two Basic Terms

- Side-channel
  - Passive
  - E.g Timing analysis, or even acoustic analysis

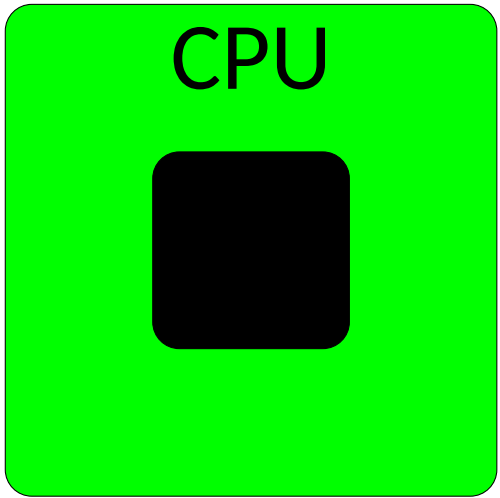
Two Basic Terms

- Side-channel
  - Passive
  - E.g Timing analysis, or even acoustic analysis
- Covered Channel
  - Active
  - E.g. Trojan Horse

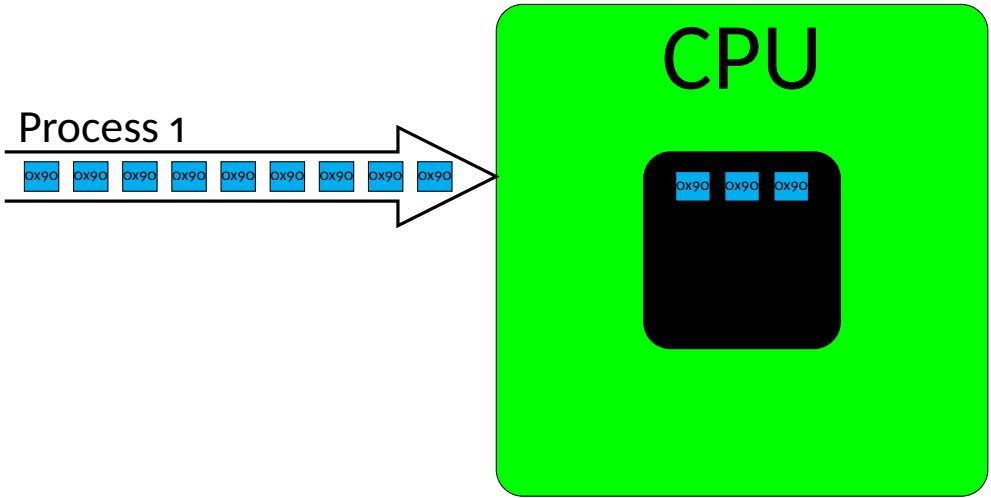
CPU-Caches



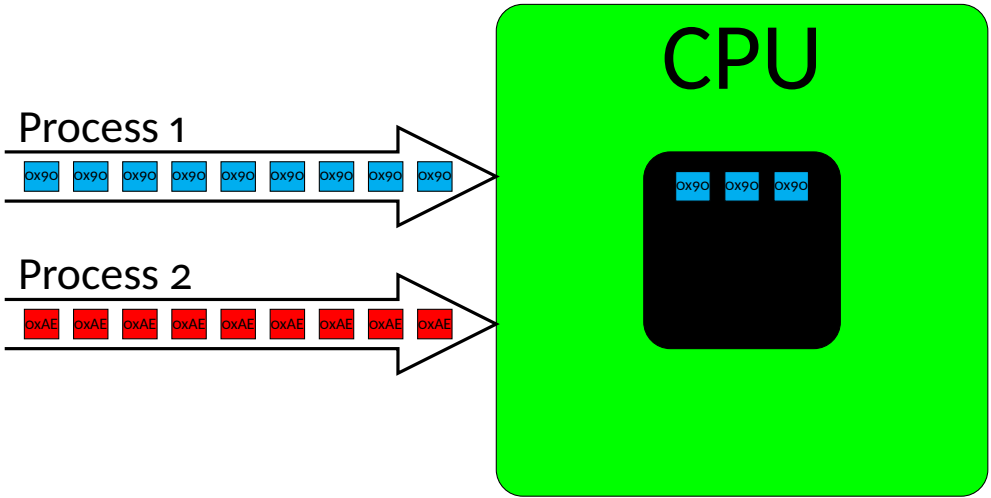
Hyper-Threading



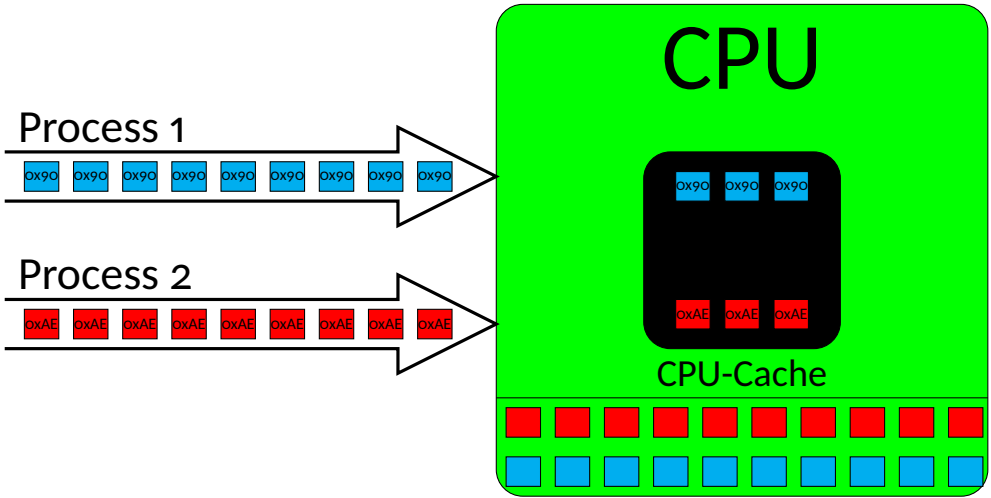
Hyper-Threading



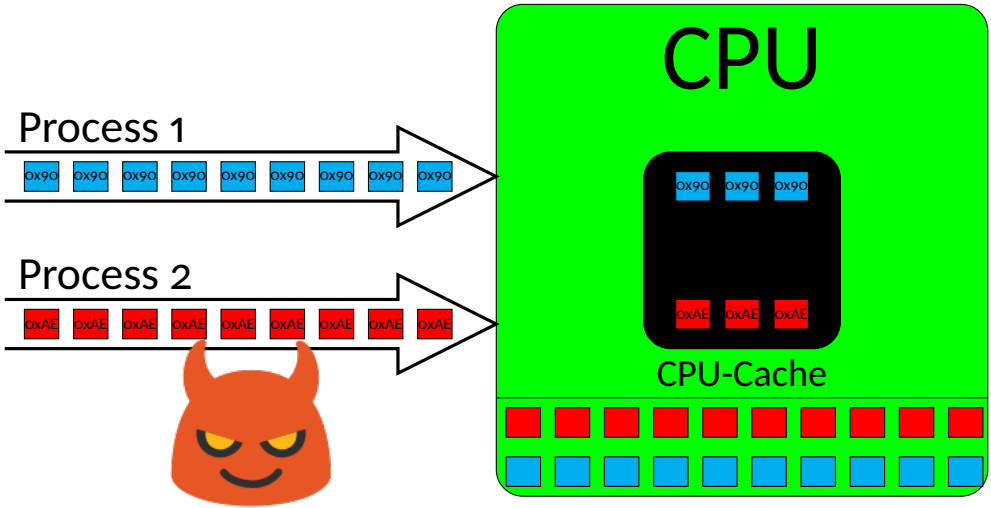
Hyper-Threading



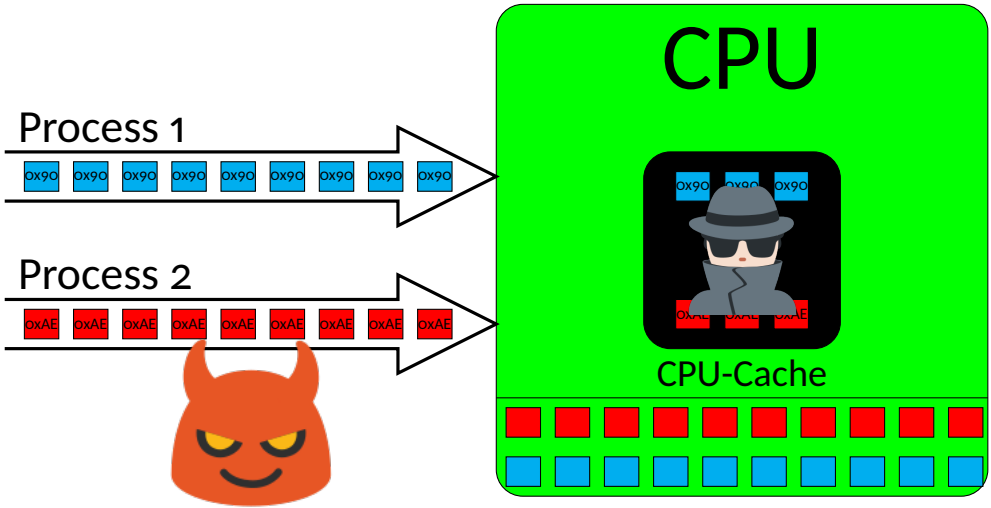
Hyper-Threading



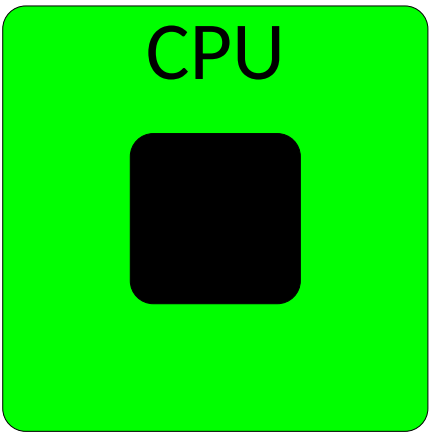
Hyper-Threading



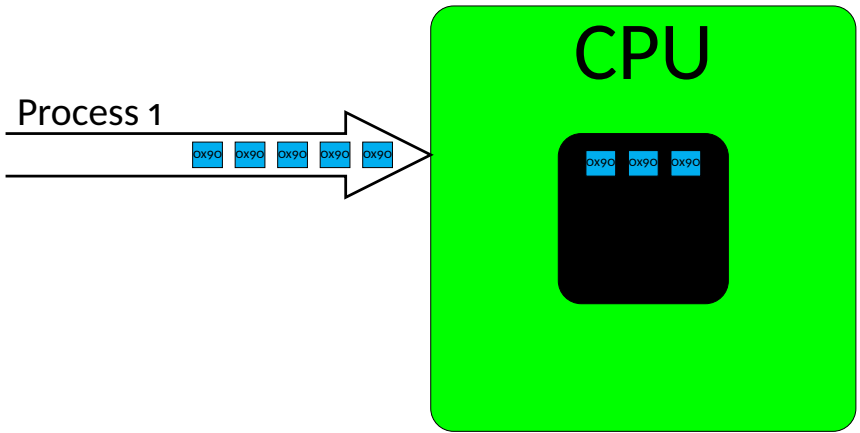
Hyper-Threading



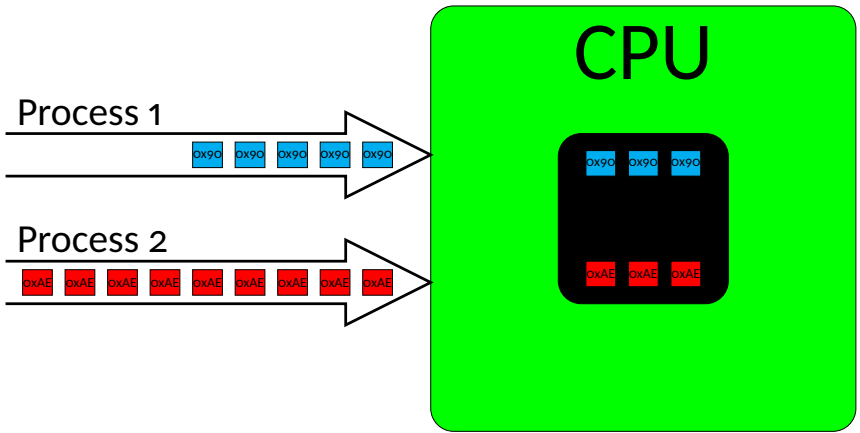
Speculative Execution



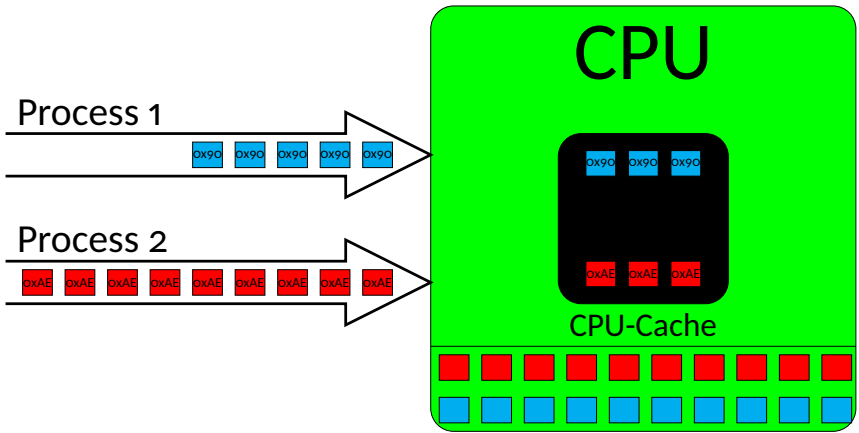
Speculative Execution



Speculative Execution

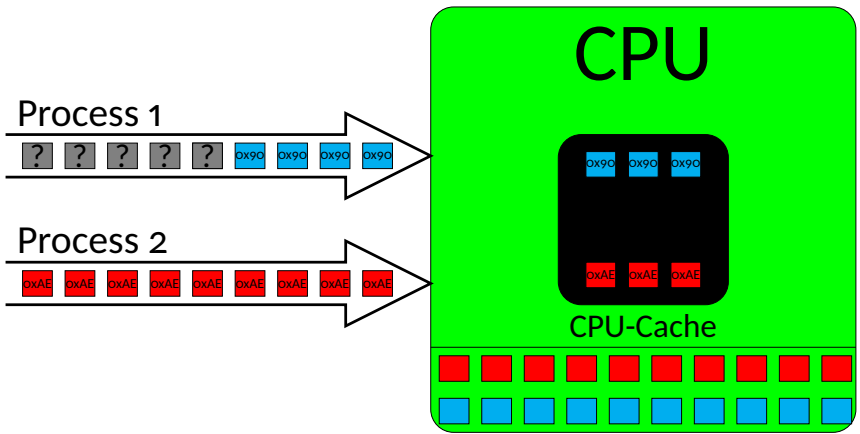


Speculative Execution

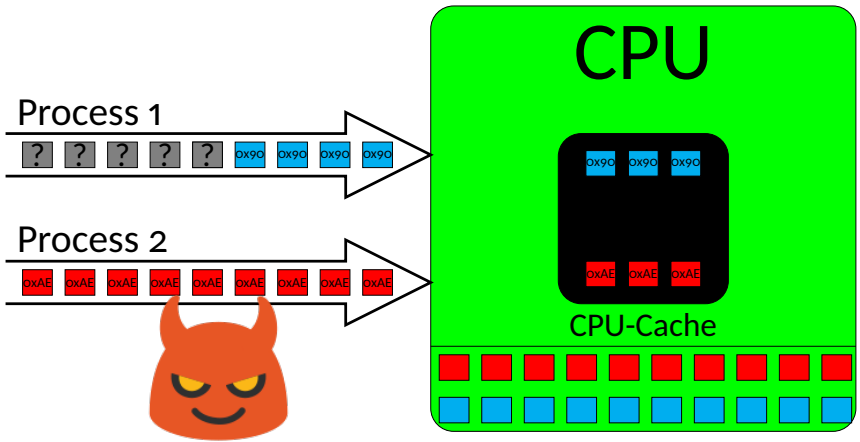


Speculative Execution

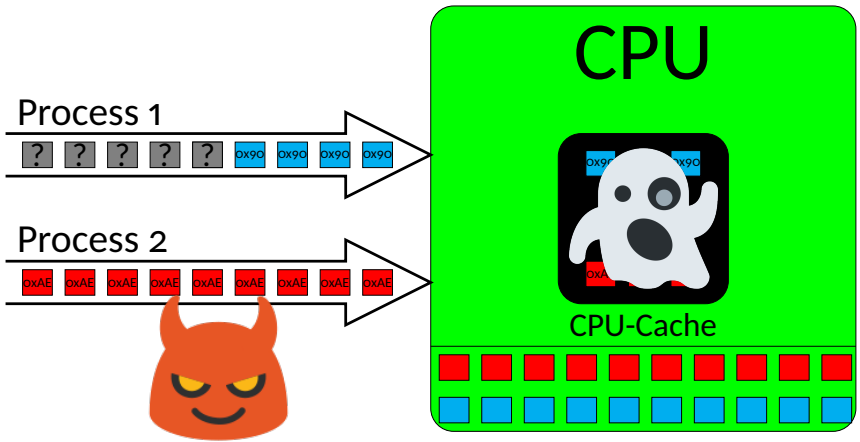
”I try to guess, so I’m faster!”



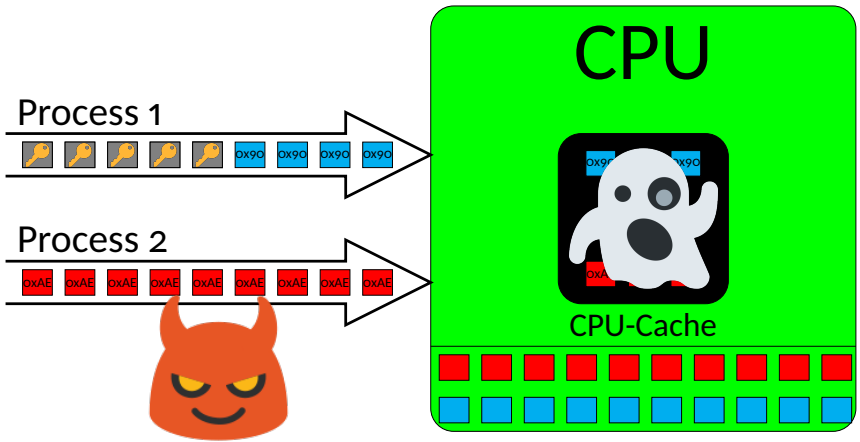
Speculative Execution



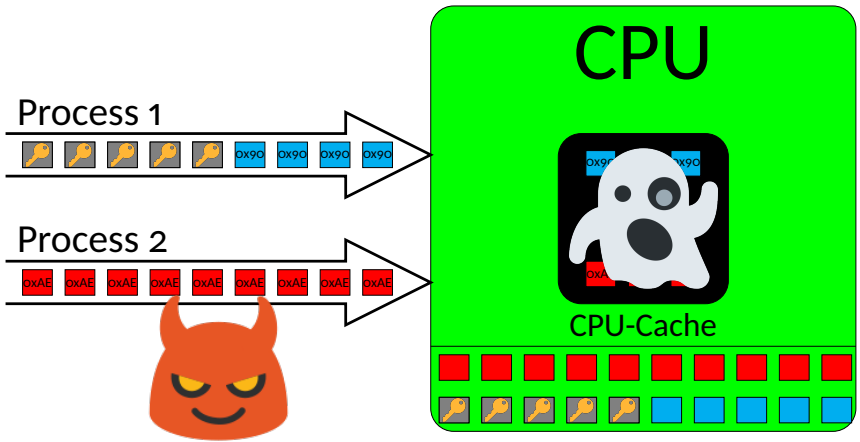
Speculative Execution



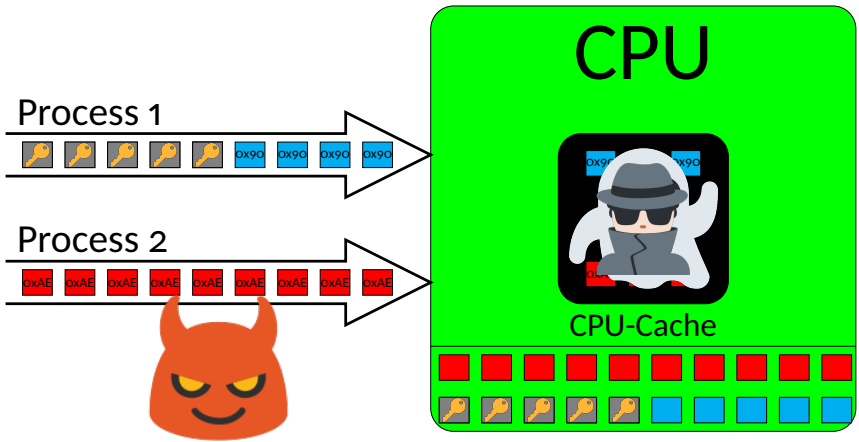
Speculative Execution



Speculative Execution



Speculative Execution



Hands on! — Meltdown

Hands on! — Meltdown

1. Access data *D* at an illegal address — get's executed speculatively
2. Make an address *A* of the data *D* — just a `shl`
3. Load data at address *A*
4. Program crashes
5. Do some tricks (e.g. `fork`)
6. Probe access time to *A* to learn if it is cached
7. Now we know that an address based on *D* is cached
8. Revert step 2 to get the data

How can you protect yourself?

How can you protect yourself?

- As always: Apply Security-Updates!

How can you protect yourself?

- As always: Apply Security-Updates!
- Don’t trust the memory!

How can you protect yourself?

- As always: Apply Security-Updates!
- Don’t trust the memory!
  - Overwrite critical data!

How can you protect yourself?

- As always: Apply Security-Updates!
- Don’t trust the memory!
  - Overwrite critical data!
  - C: `explicit_bzero()`

How can you protect yourself?

- As always: Apply Security-Updates!
- Don’t trust the memory!
  - Overwrite critical data!
  - C: `explicit_bzero()`
  - Java: `char[]`

How can you protect yourself?

- As always: Apply Security-Updates!
- Don’t trust the memory!
  - Overwrite critical data!
  - C: `explicit_bzero()`
  - Java: `char[]`
  - Python, Go, and co. (essentially all garbage collecting languages with immutable strings): No guaranteed solution.

Questions?

Backup Slides

Meltdown

```
1 retry:
2  mov al, byte [rcx]
3  shl rax, 0xc
4  jz retry
5  mov rbx, qword [rbx + rax]
```

Meltdown

```
1 retry:
2   mov al, byte [rcx]
3   shl rax, 0xc
4   jz retry
5   mov rbx, qword [rbx + rax]
```

Meltdown

```
1 retry:
2   mov al, byte [rcx]
3   shl rax, 0xc
4   jz retry
5   mov rbx, qword [rbx + rax]
```

Meltdown

```
1 retry:
2   mov al, byte [rcx]
3   shl rax, 0xc
4   jz retry
5   mov rbx, qword [rbx + rax]
```

Meltdown

```
1 retry:
2   mov al, byte [rcx]
3   shl rax, 0xc
4   jz retry
5   mov rbx, qword [rbx + rax]
```

Meltdown

```
1 retry:
2   mov al, byte [rcx]
3   shl rax, 0xc
4   jz retry
5   mov rbx, qword [rbx + rax]
```

Meltdown

```
1 retry:
2   mov al, byte [rcx]
3   shl rax, 0xc
4   jz retry
5   mov rbx, qword [rbx + rax]
```