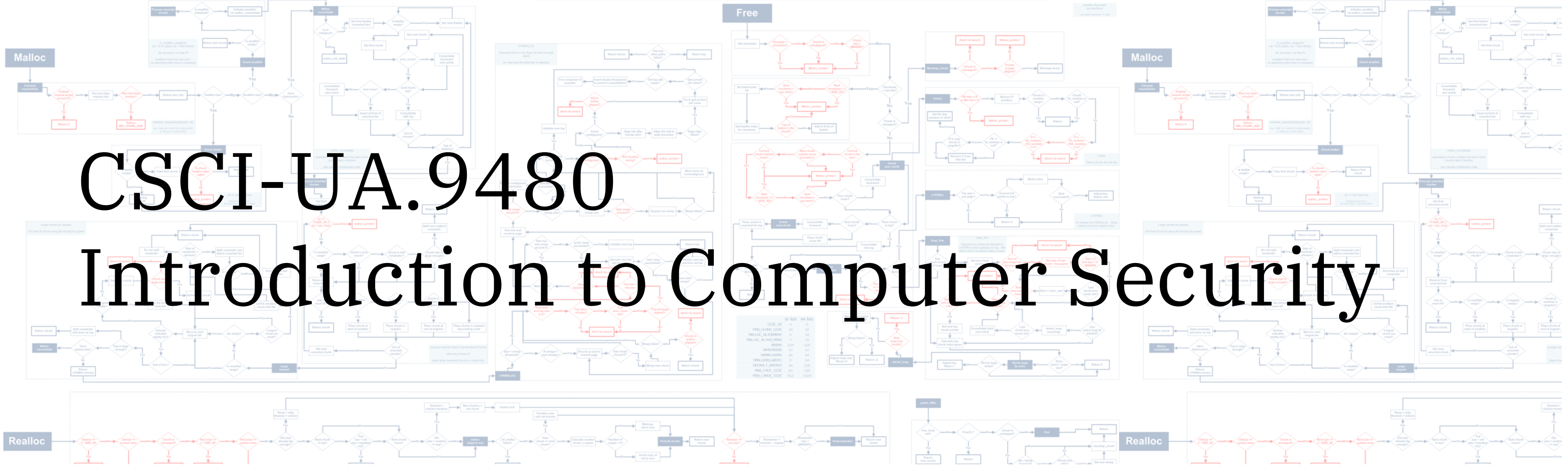


CSCI-UA.9480
Introduction to Computer Security

NYU

Session 3.5
Meltdown and Spectre

Prof. Nadim Kobeissi

But Nadim, why are we covering this?

ars TECHNICA — BIZ & IT | TECH | SCIENCE | POLICY | CARS | GAMING & CULTURE | STORE

TECH —

“Meltdown” and “Spectre:” Every modern processor has unfixable security flaws

Immediate concern is for Intel chips, but everyone is at risk.

PETER BRIGHT - 1/4/2018, 1:30 AM

intel Newsroom — Top News Sections | News By Category | All News | Search Newsroom

News Byte

January 3, 2018

Contact Intel PR

INTEL RESPONDS TO SECURITY RESEARCH FINDINGS

Intel and other technology companies have been made aware of new security research describing software analysis methods that, when used for malicious purposes, have the potential to improperly gather sensitive data from computing devices that are operating as designed. Intel believes these exploits do not have the potential to corrupt, modify or delete data.

WIRED — Technology | Science | Culture | Gear | Business | Politic

Hacking

The major Spectre and Meltdown flaws could linger for decades

Almost all microprocessors used in computers, phones, servers and more are affected by a massive security flaw. You should update your systems

BBC NEWS — Home | Video | World | UK | Business | Tech | Science | Stories | Entertainment & Arts

Technology

Meltdown and Spectre: All Macs, iPhones and iPads affected

5 January 2018

Share

cnet — BEST PRODUCTS | REVIEWS | NEWS | VIDEO | HOW TO | SMART HOME | CARS | DEALS

SECURITY

Nope, no Intel chip recall after Spectre and Meltdown, CEO says

CEO Brian Krzanich says the new security vulnerabilities may be deep but they're also being fixed with software updates.

ars TECHNICA — BIZ & IT | TECH | SCIENCE | POLICY | CARS | GAMING & CULTURE | STORE | FOR

THIS IS HUGE. REALLY. —

Meltdown and Spectre: Here’s what Intel, Apple, Microsoft, others are doing about it

Intel, Microsoft, ARM, and others have responded. We dig in.

PETER BRIGHT - 1/5/2018, 2:52 PM

Fixed confidentially across whole ecosystem.

The mysterious case of the Linux Page Table Isolation patches

[Various errors and updates are addressed in Quiet in the peanut gallery]

---

tl;dr: there is presently an embargoed security bug impacting apparently all contemporary CPU architectures that implement virtual memory, requiring hardware changes to fully resolve. Urgent development of a software mitigation is being done in the open and recently landed in the Linux kernel, and a similar mitigation began appearing in NT kernels in November. In the worst case the software fix causes huge slowdowns in typical workloads. There are hints the attack impacts common virtualization environments including Amazon EC2 and Google Compute Engine, and additional hints the exact attack may involve a new variant of Rowhammer.

https://sweetness.hmmz.org/2018-01-01-the-mysterious-case-of-the-linux-page-table.html

Meltdown: a high-level overview

Based on work by Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom and Mike Hamburg.

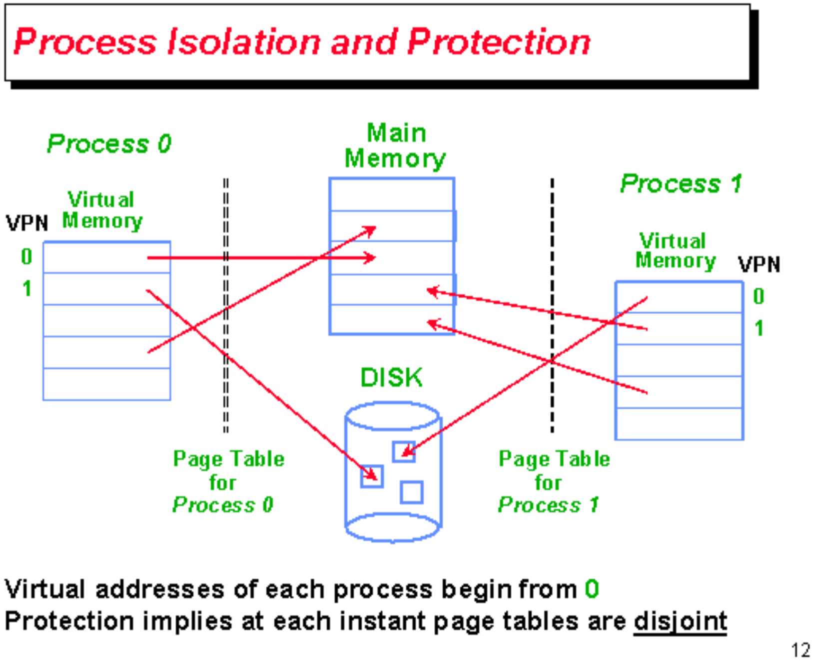
3.5a

“Meltdown breaks all security guarantees provided by address space isolation and, thus, every security mechanism building upon this foundation. On affected systems, Meltdown enables an adversary to read memory of other processes or virtual machines in the cloud without any permissions or privileges.”

– Meltdown paper authors.

What is process memory isolation?

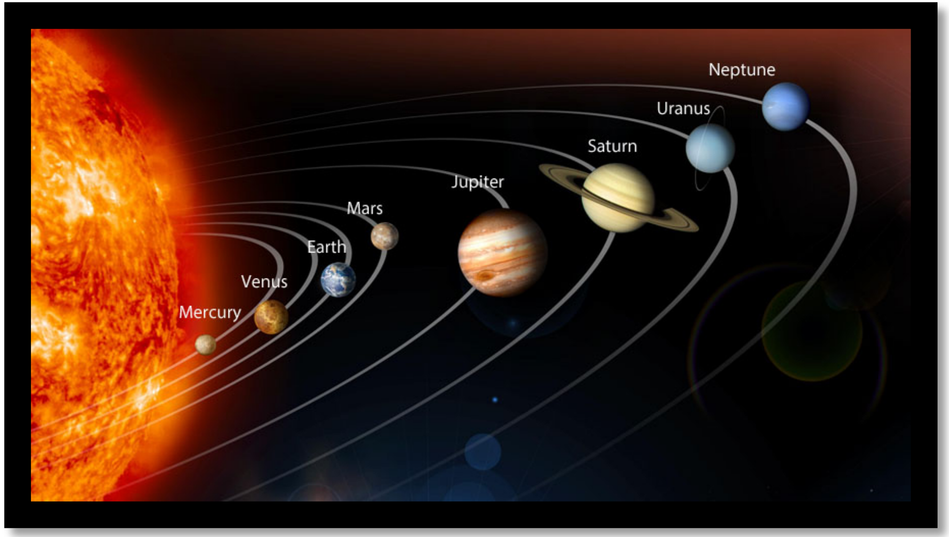
- Crucial component in systems security.
- Handled by the kernel.
- Ensures that processes can't access each other's reserved memory addresses and allocation regions.
- *ASLR* (address space layout randomization) is not a process memory isolation technique, but further improves on the security and integrity of data in memory.

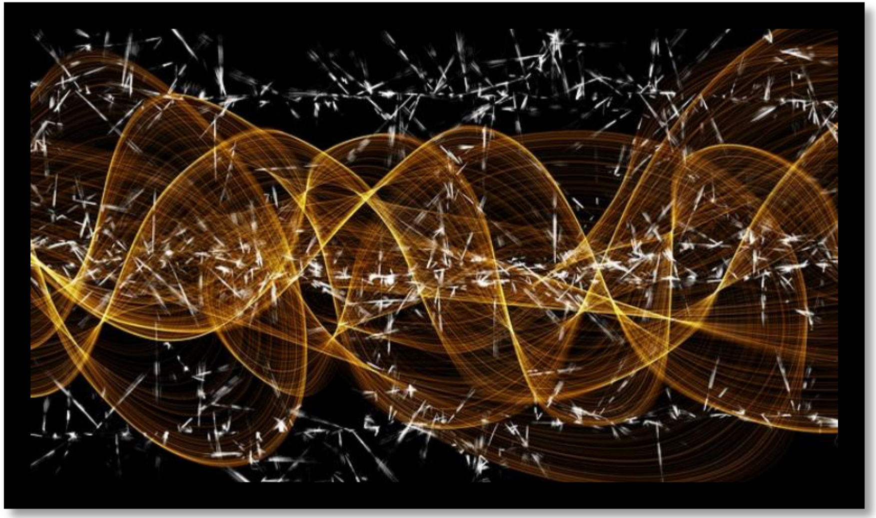


Meltdown: quick facts.

- Meltdown is a *hardware vulnerability.* Works regardless of software stack.
- Exploits side channels to allow an attacker who can run code on the processor to dump *entire computer memory*.
- Caused by *out-of-order optimizations* on modern CPUs.
- *Out-of-order execution*: Run faster instructions before slower instructions if there is no side effect on the result.

CPUs are like the universe...

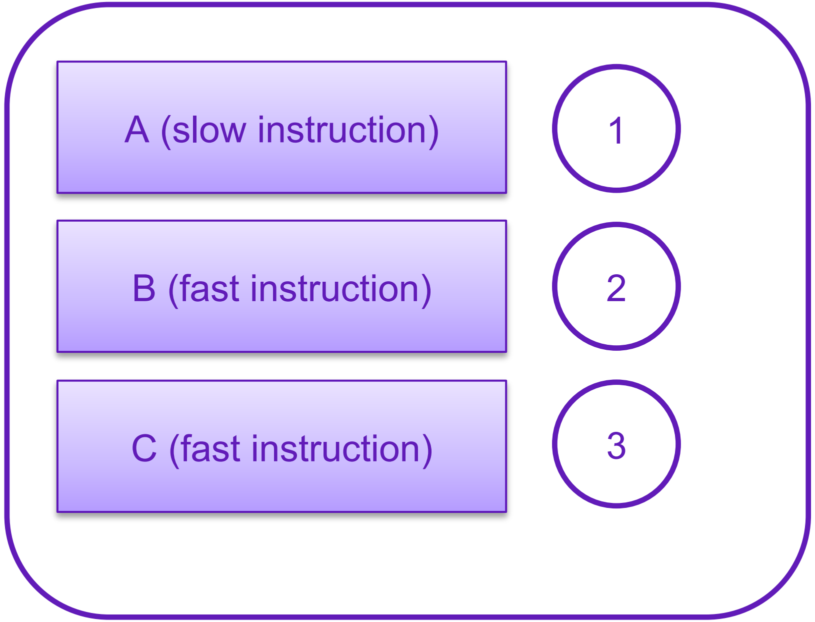


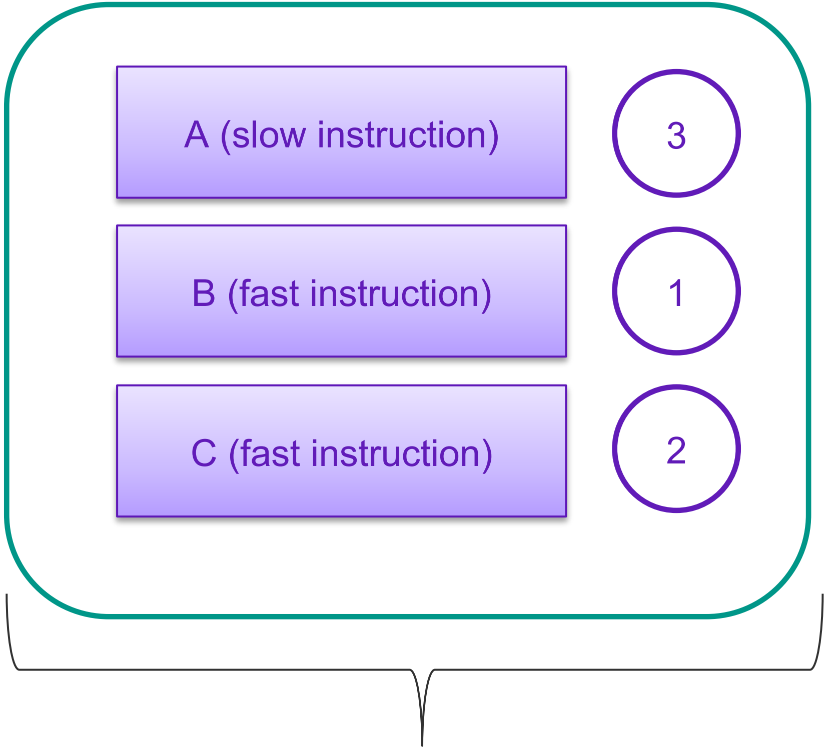


Organized and predictable on the macro scale...
(Developer sees programs executing sequentially)

Unpredictable and deranged on the quantum scale.
(Sequential execution is relaxed and reordered for performance)

In-order versus out-of-order execution.



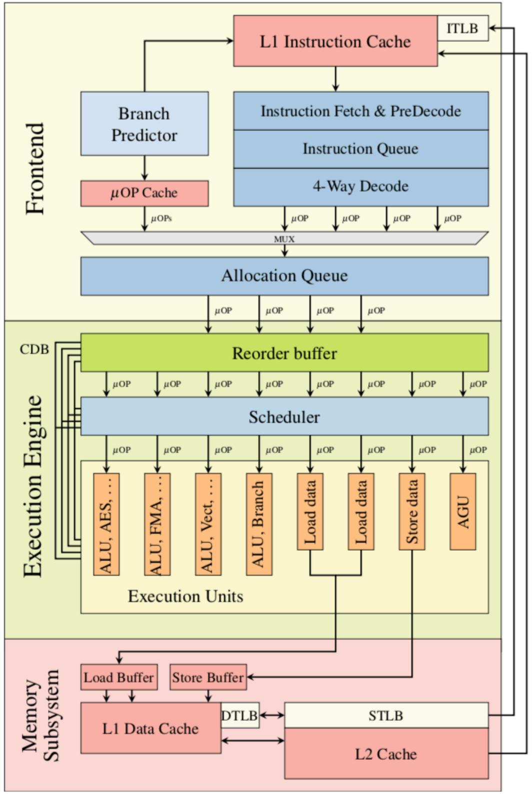


But what about side effects?

Out-of-order execution architecture.

- Fetch instructions from memory.
- Assign micro-operation.

- Determine operation order.
- Schedule execution depending on micro-operation.



Meltdown: simple example.

- Try to read from protected kernel memory (would result in a page fault).
- Multiply the byte retrieved by 4096 and then read from that address.

- First instruction should stop the process, right? *But what about out-of-order execution?*
- Address read by third instruction reveals byte from first instruction!

```
; rcx = a protected kernel memory address
; rbx = address of a large array in user space
mov al, byte [rcx]          ; read from forbidden kernel address
shl rax, 0xc                ; multiply the result from the read operation with 4096
mov rbx, qword [rbx + rax] ; touch the user space array at the offset that we just calculated
```

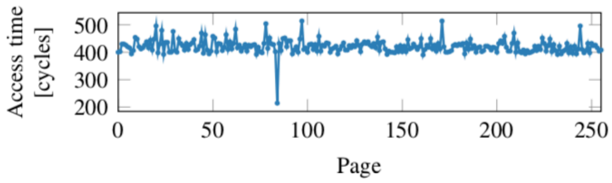


Figure 4: Even if a memory location is only accessed during out-of-order execution, it remains cached. Iterating over the 256 pages of probe_array shows one cache hit, exactly on the page that was accessed during the out-of-order execution.

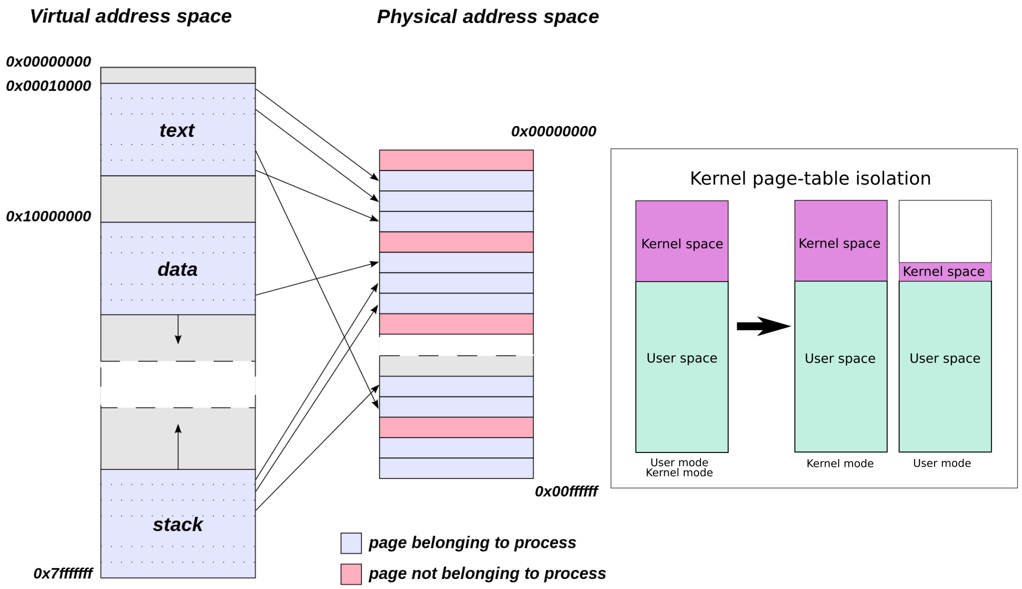
Meltdown: yup, it's practical!

```
f94b76f0: 12 XX e0 81 19 XX e0 81 44  6f 6c 70 68 69 6e 31 |........Dolphin1|
f94b7700: 38 e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |8...............|
f94b7710: 70 52 b8 6b 96 7f XX XX XX  XX XX XX XX XX XX XX |pR.k............|
f94b7720: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7730: XX XX XX XX 4a XX XX XX XX  XX XX XX XX XX XX XX |....J...........|
f94b7740: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7750: XX XX XX XX XX XX XX XX XX  XX e0 81 69 6e 73 74 |............inst|
f94b7760: 61 5f 30 32 30 33 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |a_0203..........|
f94b7770: 70 52 18 7d 28 7f XX XX XX  XX XX XX XX XX XX XX |pR.}(...........|
f94b7780: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7790: XX XX XX XX 54 XX XX XX XX  XX XX XX XX XX XX XX |....T...........|
f94b77a0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b77b0: XX XX XX XX XX XX XX XX XX  XX XX XX 73 65 63 72 |............secr|
f94b77c0: 65 74 70 77 64 30 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |etpwd0..........|
f94b77d0: 30 b4 18 7d 28 7f XX XX XX  XX XX XX XX XX XX XX |0..}(...........|
f94b77e0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b77f0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7800: e5 e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |................|
f94b7810: 68 74 74 70 73 3a 2f 2f 61  64 64 6f 6e 73 2e 63 |https://addons.c|
f94b7820: 64 6e 2e 6d 6f 7a 69 6c 6c  61 2e 6e 65 74 2f 75 |dn.mozilla.net/u|
f94b7830: 73 65 72 2d 6d 65 64 69 61  2f 61 64 64 6f 6e 5f |ser-media/addon_|
```

Listing (4) Memory dump of Firefox 56 on Ubuntu 16.10 on a Intel Core i7-6700K disclosing saved passwords.

KAISER: mitigation for Meltdown.

- Also called Kernel page-table isolation (KPTI).
- Increases separation between mapping virtual addresses to physical addresses (maintained in ”page tables”) in *kernel space* and *user space.*



Spectre: a high-level overview

Based on work by Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz and Yuval Yarom.

3.5b

"Spectre attacks involve inducing a victim to speculatively perform operations that would not occur during correct program execution and which leak the victim's confidential information via a side channel to the adversary."

– Spectre paper authors.

Spectre and speculative execution.

- *Out-of-order execution*: Run faster instructions before slower instructions if there is no side effect on the result.
- *Speculative execution*: If calculating which branch to follow is more expensive than the resulting branches, start calculating most likely branch before deciding which one to follow.

```
if (slowFetchFromMemory()) {
        doSomethingFast();
} else {
        anotherFastThing();
}
```

Toy example: green code is estimated to be more likely based on previous runs, is speculatively executed before red code.

```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * 4096)|0) & (32*1024*1024-1))|0;
  localJunk ^= probeTable[index|0]|0;
}
```

Listing 2: Exploiting Speculative Execution via JavaScript.

Spectre and speculative execution.

- *Out-of-order execution*: Run faster instructions before slower instructions if there is no side effect on the result.
- *Speculative execution*: If calculating which branch to follow is more expensive than the resulting branches, start calculating most likely branch before deciding which one to follow.

```
if (slowFetchFromMemory()) {
        doSomethingFast();
} else {
        anotherFastThing();
}
```

If incorrect path was executed, then CPU has to roll back execution to maintain functional correctness.

```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * 4096)|0) & (32*1024*1024-1))|0;
  localJunk ^= probeTable[index|0]|0;
}
```

Listing 2: Exploiting Speculative Execution via JavaScript.

Spectre and speculative execution.

- *Out-of-order execution*: Run faster instructions before slower instructions if there is no side effect on the result.
- *Speculative execution*: If calculating which branch to follow is more expensive than the resulting branches, start calculating most likely branch before deciding which one to follow.

```
if (slowFetchFromMemory()) {
        doSomethingFast();
} else {
        anotherFastThing();
}
```

But what about cache modifications?
The called value is still "warm" in cache!

```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * 4096)|0) & (32*1024*1024-1))|0;
  localJunk ^= probeTable[index|0]|0;
}
```

Listing 2: Exploiting Speculative Execution via JavaScript.

Speculative execution: making CPUs faster.

- *Out-of-order execution*: Run faster instructions before slower instructions if there is no side effect on the result.
- *Speculative execution*: If calculating which branch to follow is more expensive than the resulting branches, start calculating most likely branch before deciding which one to follow.

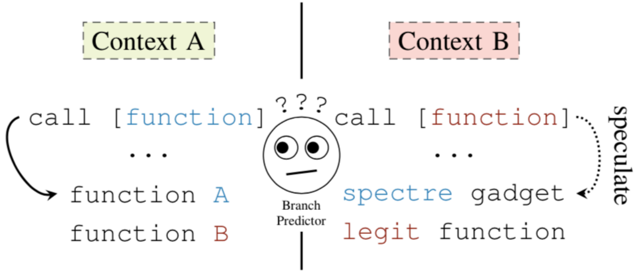


Fig. 2: The branch predictor is (mis-)trained in the attacker-controlled context A. In context B, the branch predictor makes its prediction on the basis of training data from context A, leading to speculative execution at an attacker-chosen address which corresponds to the location of the Spectre gadget in the victim's address space.

```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * 4096)|0) & (32*1024*1024-1))|0;
  localJunk ^= probeTable[index|0]|0;
}
```

Listing 2: Exploiting Speculative Execution via JavaScript.

Spectre and speculative execution.

- *Out-of-order execution*: Run faster instructions before slower instructions if there is no side effect on the result.
- *Speculative execution*: If calculating which branch to follow is more expensive than the resulting branches, start calculating most likely branch before deciding which one to follow.

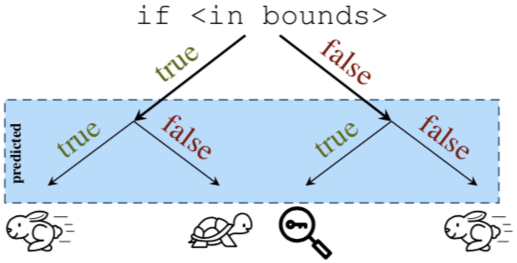


Fig. 1: Before the correct outcome of the bounds check is known, the branch predictor continues with the most likely branch target, leading to an overall execution speed-up if the outcome was correctly predicted. However, if the bounds check is incorrectly predicted as true, an attacker can leak secret information in certain scenarios.

```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * 4096)|0) & (32*1024*1024-1))|0;
  localJunk ^= probeTable[index|0]|0;
}
```

Listing 2: Exploiting Speculative Execution via JavaScript.

Variant 1: Violating JavaScript's Sandbox

Teach JIT that index is in bounds for **`simpleByteArray[]`** so it can omit bounds check in next line. Want length uncached for attack passes

`index` will be in-bounds on training passes, and out-of-bounds on attack passes

Do the out-of-bounds read on attack passes!

```
if (index < simpleByteArray.length) {
  index = simpleByteArray[index | 0];
  index = (((index * TABLE1_STRIDE)|0) & (TABLE1_BYTES-1))|0;
  localJunk ^= probeTable[index|0]|0;
}
```

"|0" is a JS optimizer trick
(makes result an integer)

4096 bytes (= page size)

Need to use the result so the operations aren't optimized away

Leak out-of-bounds read result into cache state!

This AND keeps the JIT from adding unwanted bounds checks on the next line

Credit: Jann Horn, Real World Crypto 2018

Spectre: harder to mitigate than Meltdown.

- Prevent speculative execution altogether? Would be a serious performance hit for Intel and other CPU manufacturers.
- Employ better process isolation within specific applications and use cases? Example: Chrome executes each browser tab as a separate CPU process.

Spectre is here to stay
An analysis of side-channels and speculative execution

Ross Mcilroy
Google
rcmilroy@google.com

Jaroslav Sevcik
Google
jarin@google.com

Tobias Tebbi
Google
tebbi@google.com

Ben L. Titzer
Google
titzer@google.com

Toon Verwaest
Google
verwaest@google.com

February 15, 2019

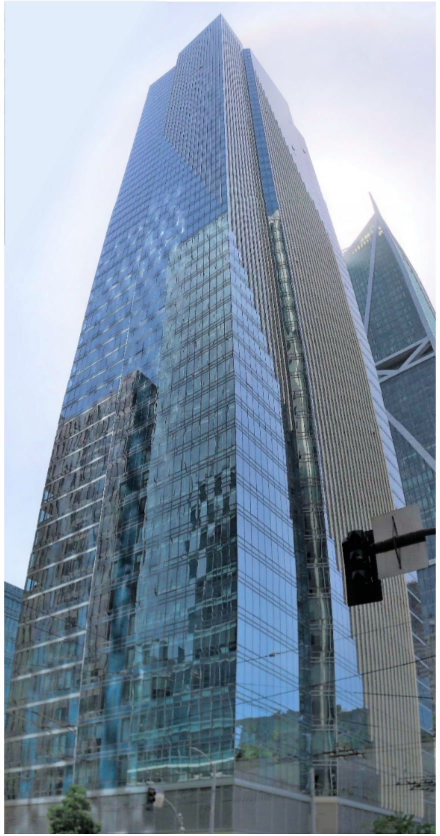
Abstract

The recent discovery of the Spectre and Meltdown attacks represents a watershed moment not just for the field of Computer Security, but also of Programming Languages. This paper explores speculative side-channel attacks and their implications for programming languages. These attacks leak information through micro-architectural side-channels which we show are not mere bugs, but in fact lie at the foundation of optimization. We identify three open problems, (1) finding side-channels, (2) understanding speculative vulnerabilities, and (3) mitigating them. For (1) we introduce a mathematical meta-model that clarifies the source of side-channels in simulations and CPUs. For (2) we introduce an architectural model with speculative semantics to study recently-discovered vulnerabilities. For (3) we explore and evaluate software mitigations and prove one correct for this model. Our analysis is informed by extensive offensive research and defensive implementation work for V8, the production JavaScript virtual machine in Chrome. Straightforward extensions to model real hardware suggest these vulnerabilities present formidable challenges for effective, efficient mitigation. As a result of our work, we now believe that speculative vulnerabilities on today's hardware defeat all language-enforced confidentiality with no known comprehensive software mitigations, as we have discovered that untrusted code can construct a universal read gadget to read all memory in the same address space through side-channels. In the face of this reality, we have shifted the security model of the Chrome web browser and V8 to process isolation.

Graphic courtesy of Paul Kocher

Society
Nation states
People
Business objectives

Security goals



Exponential growth

Abstractions

U.I.
Applications
AI
Libraries/frameworks
Browsers
Languages
Compilers
Operating systems
Protocols
Drivers
Circuit board
CPU architectures
Chip
Microarchitectures
Logic block
Transistors

Clouds

Dependencies & assumptions

Foundations

Abstraction creates security challenges

Security-critical details hidden in layers

Needs of distant layers unclear

People specialize then miss big picture

Economics don't fund adequate investment

Risks in other layers deter improvements

Changes aren't communicated across layers

Next time: Browser Security Model

The first section of Part 4 of this course: Web Security.

4.1