Meltdown (and Spectre) Vulnerability

Corso di Architettura dei Calcolatori Elettronici

These slides are extracted from:
Lipp M., Schwarz M., Gruss D. et Al. Meltdown, *arXiv preprint arXiv:1801.01207, 2018*

Salvatore Venticinque

Università
degli Studi
della Campania
Luigi Vanvitelli

Spectre and meltdown targets

Spectre

- Spectre breaks the isolation between different applications.
- Allows an attacker to trick error-free applications, which follow best practices, into leaking their secrets.
- The safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre

Meltdown

- Meltdown breaks the most fundamental isolation between user applications and the operating system.
- This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

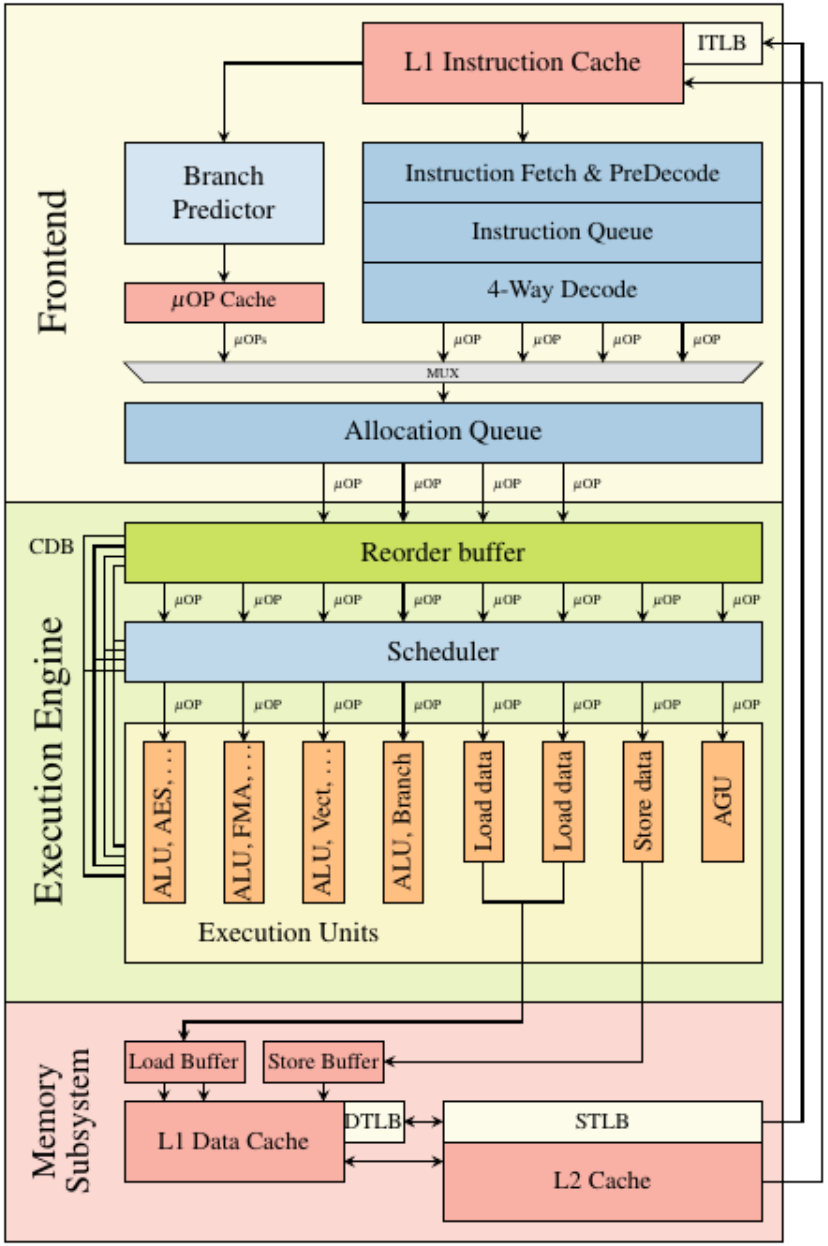
Differences

- Spectre tricks other applications into accessing arbitrary locations in their memory.

- Meltdown breaks the mechanism that keeps applications from accessing arbitrary system memory.
- Consequently, applications can access system memory.

Both attacks use side channels to obtain the information from the accessed memory location.

Out of Order and Speculative execution



Kernel Memory Mapping

- The currently used translation table is held in a special CPU register.
- On each context switch, the operating system updates this register with the next process' translation table address to implement per process virtual address spaces.
- Each process can only reference data that belongs to its own virtual address space.

- Virtual address space itself is split into a user and a kernel part.
- User address space can be accessed by the running application.
- Kernel address space can only be accessed if the CPU is running in privileged mode.

- Kernel address space does not only have memory mapped for the kernel's own usage, but it also needs to perform operations on user pages, e.g., filling them with data.
- The entire physical memory is typically mapped in the kernel.
- On Linux and OS X, the entire physical memory is directly mapped to a pre-defined virtual address

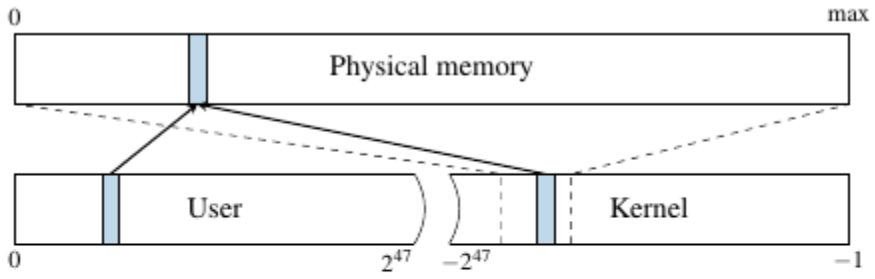


Figure 2: The physical memory is directly mapped in the kernel at a certain offset. A physical address (blue) which is mapped accessible for the user space is also mapped in the kernel space through the direct mapping.

Cache Covert channel

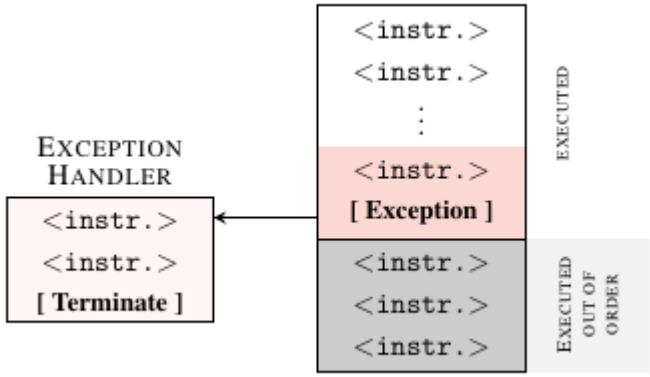
- In this code line 3 is executed only in a speculative way.

```
1 raise_exception();
2 // the line below is never reached
3 access(probe_array[data * 4096]);
```

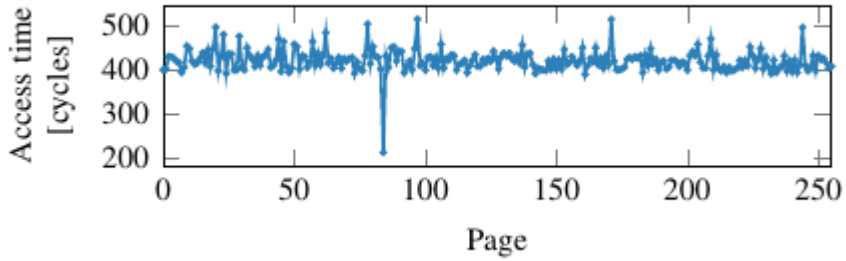
Listing 1: A toy example to illustrate side-effects of out-of-order execution.

- Anyway, eve if memory location is only accessed during out-of-order execution, but it remains cached.

- In the exception handler we can probe all 256 cache lines (data is one byte)



- 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.



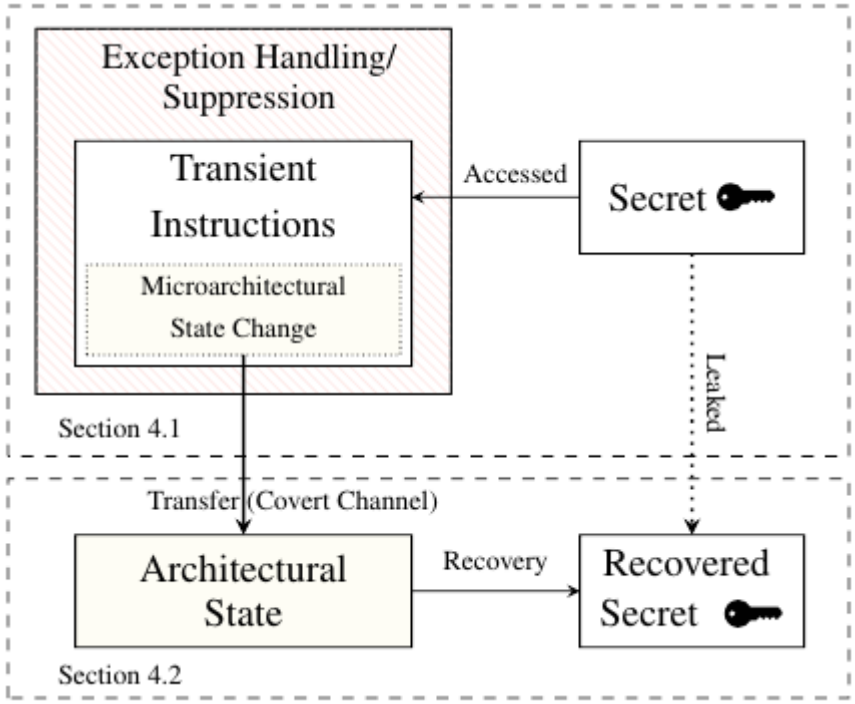
The Meltdown/Spectre approach

The idea is:

1)The transient code access a kernel address reading the secret

2)The transient code communicates the secret value using the convert channel

3)Another process, or non transient code, read the secret value checking timing of memory read.



Attack (assembler) example

Meltdown consists of 3 steps:

- ***Step 1*** The content of an attacker-chosen memory location, which is inaccessible to the attacker, is loaded into a register.
- ***Step 2*** A transient instruction accesses a cache line based on the secret content of the register.
- ***Step 3*** The attacker uses Flush+Reload to determine the accessed cache line and hence the secret stored at the chosen memory location.

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

- ***Step 1:*** Line 4 read from kernel address moving one byte in *al*
- ***Step 2:*** Line 5 shift *ax* to multiply 4KB (first page, second page, …), to point to multiple pages
- Line 6 retria if some error occurs
- ***Step 2:*** Line 7 read the one element from the probe array that is loaded in the rax cache line

The case of 0 (*jz retry*)

- If the exception is triggered while trying to read from an inaccessible kernel address, the register where the data should be stored, appears to be zeroed out.
- This is reasonable because if the exception is unhandled, the user space application is terminated, and the value from the inaccessible kernel address could be observed in the register contents stored in the core dump of the crashed process.

- If the zeroing out of the register is faster than the execution of the subsequent instruction (line 5 in Listing 2), the attacker may read a false value in the third step.
- To prevent the transient instruction sequence from continuing with a wrong value, i.e., ‘0’, Meltdown retries reading the address until it encounters a value different from ‘0’ (line 6).

- As the transient instruction sequence terminates after the exception is raised, there is no cache access if the secret value is 0.
- Thus, Meltdown assumes that the secret value is indeed ‘0’ if there is no cache hit at all.

One bit transmission

Single-bit transmission In the attack description, the attacker transmitted 8 bits through the covert channel at once and performed 28 = 256 Flush+Reload measurements to recover the secret.

However, there is a clear trade-off between:

- Transmitting more information
- Probing more line of the array

- The performance bottleneck in the generic attack description above is indeed, ***the time spent on Flush+ Reload measurements***.
-
- In fact, with this implementation, almost the entire time will be spent on Flush+Reload measurements.
-
- By transmitting only a single bit, we can omit ***all but one*** Flush+Reload measurement, i.e., the measurement on cache line 1.

If the transmitted bit was a '1', then we observe a cache hit on cache line 1. Otherwise, we observe no cache hit on cache line 1.

Firefox Passwords

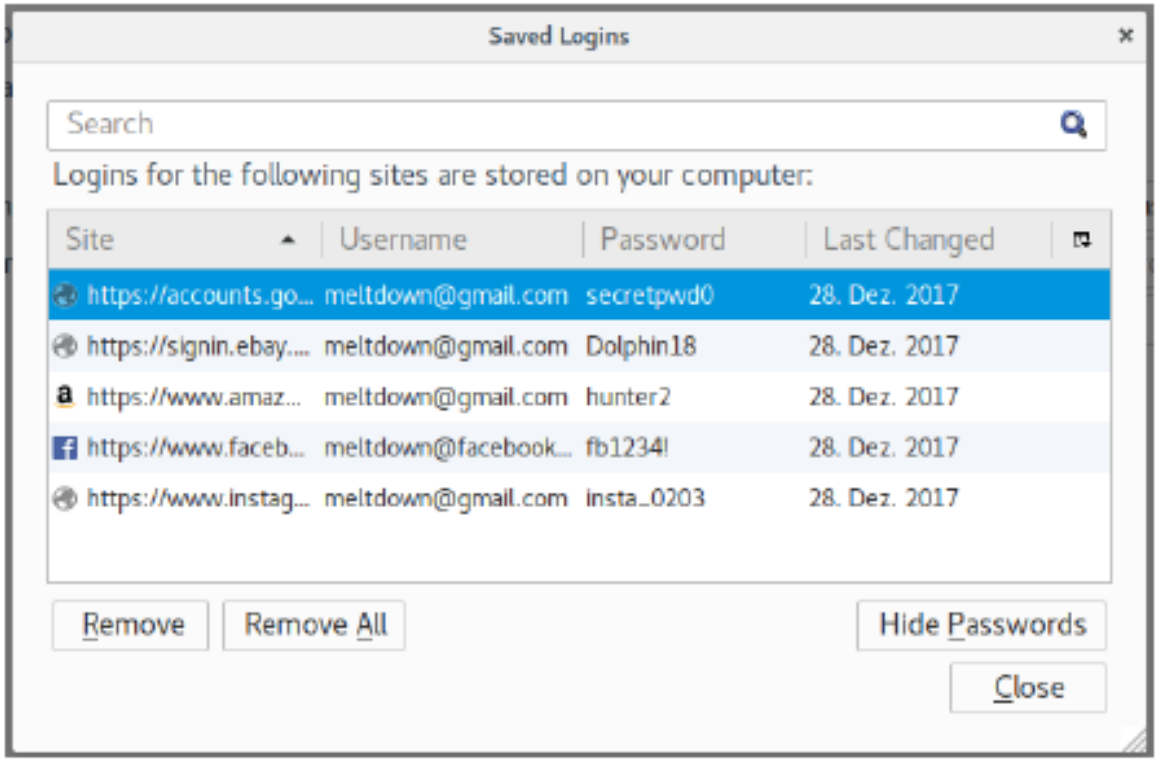


Figure 6: Firefox 56 password manager showing the stored passwords that are leaked using Meltdown in Listing 4.

Memory Dump

```
f94b7690: e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |................|
f94b76a0: e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |................|
f94b76b0: 70 52 b8 6b 96 7f XX XX  XX XX XX XX XX XX XX XX |pR.k............|
f94b76c0: 09 XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b76d0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b76e0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX 81 |................|
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 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 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_|
f94b7840: 69 63 6f 6e 73 2f 33 35  34 2f 33 35 34 33 39 39 |icons/354/354399|
f94b7850: 2d 36 34 2e 70 6e 67 3f  6d 6f 64 69 66 69 65 64 |-64.png?modified|
f94b7860: 3d 31 34 35 32 32 34 34  38 31 35 XX XX XX XX XX |=1452244815.....|
```

Listing 4: Memory dump of Firefox 56 on Ubuntu 16.10 on a Intel Core i7-6700K disclosing saved passwords (cf. Figure 6).

Meltdown concepts

- Meltdown allows attackers to read arbitrary physical memory (including kernel memory) from an unprivileged user process
- Meltdown uses out of order instruction execution to leak data via a processor covert channel (cache lines)
- Meltdown was patched (in Linux) with KAISER/KPTI

Kernel ASLR

- Kernel ASLR (Address Space Layout Randomization) Linux implements kernel ASLR by default since 4.12
- The 64-bit address space is huge, you wouldn’t want to dump the whole thing
  – 16EB theoretical limit, but 256TB practical limit
- Randomization is limited to 40 bits, meaning that locating kernel offsets is relatively easy

Windows ASLR

- Windows ASLR isn’t much different in that not all of the kernel is randomized
- Because of the way the Windows memory manager is implemented, it is unlikely that the entirety of physical memory is mapped into a single process
- Verdict: On an unpatched Windows system, most (but not all) kernel memory can be read from Windows

Linux with KAISER Patch

- The KAISER patch by Gruss et al. [8] implements a stronger isolation between kernel and user space.
- KAISER does not map any kernel memory in the user space, except for some parts required by the x86 architecture (e.g., interrupt handlers).
- Thus, there is no valid mapping to either kernel memory or physical memory (via the direct-physical map) in the user space, and such addresses can therefore not be resolved.
- Consequently, Meltdown cannot leak any kernel or physical memory except for the few memory locations which have tomapped in user space.
- We verified that KAISER indeed prevents Meltdown, and there is no leakage of any kernel or physical memory.
- Furthermore, if KASLR is active, and the few remaining memory locations are randomized, finding these memory locations is not trivial due to their small size being several kilobytes.

The Kaiser patch

As hardware is not as easy to patch, there is a need software workarounds until new hardware can be deployed.

- KAISER will be available in the upcoming releases of the Linux kernel under the name kernel page-table isolation (KPTI).
  - The patch will also be backported to older Linux kernel versions.
  - A similar patch was also introduced in Microsoft WindowsBuild 17035.
  - Also, Mac OS X and iOS have similar features.
- Although KAISER provides basic protection against Meltdown, it still has some limitations.
  - Due to the design of the x86 architecture, several privileged memory locations are required to be mapped in user space.
  - This leaves a residual attack surface for Meltdown, i.e., these memory locations can still be read from user space.

- Even though these memory locations do not contain any secrets, such as credentials, they might still contain pointers.
- Leaking one pointer can be enough to again break KASLR, as the randomization can be calculated from the pointer value.
- Still, KAISER is the best short-time solution currently

Spectre cannot be patched

Spectre abuses branch prediction and speculative execution to leak data from via a processor covert channel (cache lines)

Spectre can only read memory from the current process, not the kernel and other physical memory
Spectre does not appear to be patched

Conclusion

	Meltdown	Spectre
Allows kernel memory read	Yes	No
Was patched with KAISER/KPTI	Yes	No
Leaks arbitrary user memory	Yes	Yes
Could be executed remotely	Sometimes	Definitely
Most likely to impact	Kernel integrity	Browser memory
Practical attacks against	Intel	Intel, AMD, ARM