Meltdown and Spectre
Hardware-Centered Attacks

https://iuk.one/1033-1311

Clemens H. **Cap**
ORCID: 0000-0003-3958-6136

Department of Computer Science
University of **Rostock**
Rostock,Germany
clemens.cap@uni-rostock.de

Version 2

Overview

1. Background Information

Properties of the hardware
which enable specific attacks

Isolation

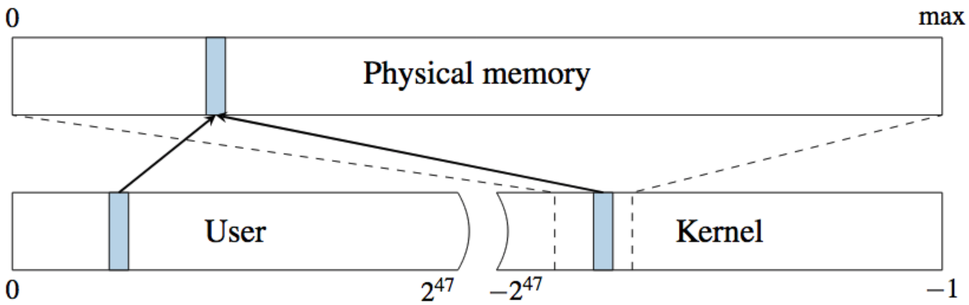


Fig. 1: Physical memory is mapped into user and kernel space

Supervisor Bit

Controls memory access

- Defines whether memory page of the kernel can be accessed.
- **Set** when entering the kernel.
- **Cleared** when leaving the kernel into user space.
- Allows system to map kernel into address space of every user process.
- Efficient transition from user process to kernel.
- No change of memory mapping when switching from user process to kernel.

Non-Sequential Execution (1)

Basic Idea: Important performance feature to overcome memory latency

Example 1: "Out of order" execution

- Memory fetch unit of processor is waiting for data to arrive
- Processor schedules subsequent operations to idle execution units
- Processor must check that there is interdependency of operations

Practical Application:

- Haswell i7-4650U has a reorder buffer for 192 micro operations.
- Spectre discoverer demonstrated that it is used for up to 188 instructions.

Non-sequential execution (2)

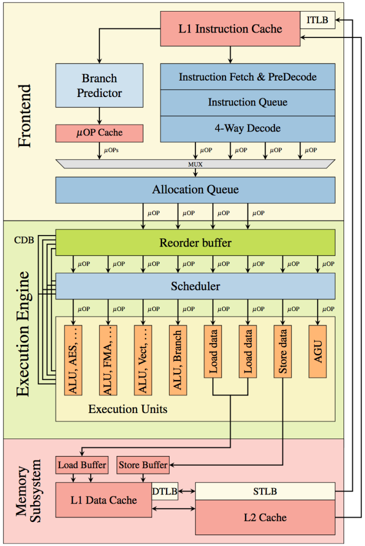
Example 2: Speculative execution

- Operations may be scheduled before the CPU knows whether the instruction will need to be executed at all
- **If yes:** Effects are made persistent in memory and registers
- **If no:** Effects are not made persistent, performed changes are un-done

Example 3: Branch prediction

- CPU tries to predict which branch of a conditional instruction will be taken even before the condition can be evaluated
- **Static prediction:** Only information of the instruction itself is used
- **Dynamic prediction:** Statistics gathered on program at runtime is used

Non-sequential execution (3)



Core Problem: State versus Micro-State

State comprises what a program can see as CPU change

- Values of registers, memory, status-flags, caches
- But probably (and that is the issue!) not the fact that something has been cached

Micro-state comprises state changes of the CPU which do not affect program flow

Issues

- What is the proper distinction between state and micro-state?
- Can we do side-channel attacks on the micro-state?
- What can these attacks achieve?

Side Effects of Out-of-Order Execution

1. CPU executes an instruction which dynamically raises an exception
   For example: Division by zero or overflow
2. CPU branches to exception handler
   Instructions following the instruction which caused the exception
   might have already been executed speculatively
   For example: Accessing a memory cell
3. CPU will prevent the speculatively executed instructions
   from having long lasting data-flow effects
   For example: Changing the value of a register
4. CPU will have changed micro-state as result of these executions
   For example: A certain value is already cached

Example: Out-of-Order Execution (1)

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

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

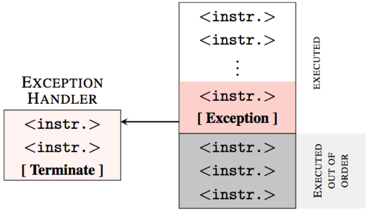


Fig. 3: Out-of-Order Execution, Code and CPU.

Example: Out-of-Order Execution (2)

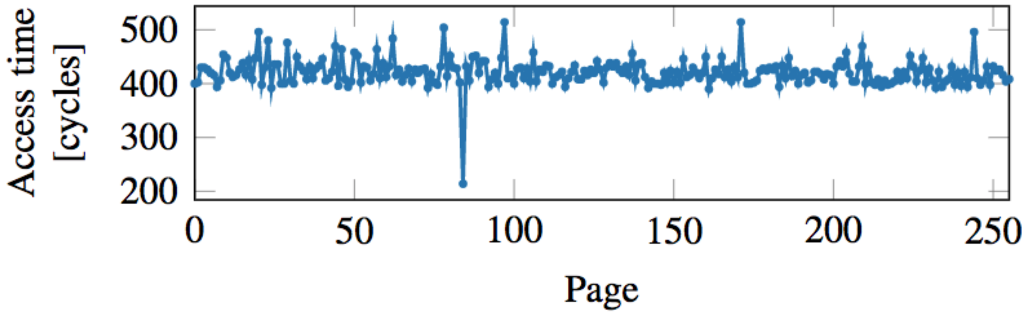


Fig. 4: Effect of memory management side-effects on memory access time

Example: Out-of-Order Execution (3)

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

Src. 1: Example code for side-effects of out-of-order execution

Consequences:

- the access address depends on data and
- the value of data defines a cache unit (4K)
- the cache timing behavior allows exact prediction of the value of data

Cache Attacks

Architecture

- CPUs have caches for storing frequently used data.
- Caches may be private to a core or shared between them.

Flush and Reload Attacks

- Side-channel attack where an attacker finds out about executional details of an algorithm.
- Attacker flushes a shared cache cell
- Attacker measures how long it takes for attacked process to reload the data
- Attacker learns if attacked process has loaded data from cache or memory

Cache Covert Channels can be established by

- encoding value into cache line information
- reading out timing information

2. Spectre and Meltdown

Two specific side-channel attacks on a CPU

General Idea of Spectre

2 Variants of Attack

- Spectre 1: Array bounds check bypass
- Spectre 2: Branch Prediction for Conditional Branches

Phases of Spectre (1)

Training Phase:

- Attacker invokes code with valid inputs
- Code trains the branch predictor to expect that the if condition will be true.

Exploit Phase:

- Attacker invokes code with a value of `x` outside of the array bounds of `array1`.

```
if (x < array1_size) y = array2[array1[x] * 4096];
```

Src. 2:

Phases of Spectre (2)

Complete Attack:

1. CPU guesses that bounds check will be true
2. CPU speculatively executes assignment
3. Read from `array2` depends on value of `array1` at position `x`
4. CPU discovers that branch is not to be taken
5. CPU states are reverted
6. CPU cache states are not reverted
7. Attacker can find out by timing analysis, which part of `array2` is cached
8. This reveals the value of `array1` at position `x`

Practical Use:

- Attack demonstrated for numerous Intel, AMD and some ARM processors
- Read rate into memory of 10KB/s with error rate of 0.01%
- Proof of concept code exists for C and Javascript

How Practical is This? Really

Requirements for attack:

- Attacker must be able to execute code on the attacked system
- This requires specific setups

How frequent or likely are such setups?

- Browsers
- Macros in documents or plugins, activeX
- Hypervisors
- Cloud installations

General Idea of Meltdown

Description:

- Conceptually very similar to Spectre
- Uses a cache side-channel
- Utilizes exceptions generated by the CPU

Effect is the same as with Spectre:

- Assume: Attacker can run code on the processor
- Attacker obtains a dump of the entire kernel address space.
- CPU allow an unprivileged process to load data from a privileged address into temporary register.
- CPU performs further computations based on the value of this register

Phase 1: Transient Instruction

Idea: Execute an instruction targeting a protected address.

Problem: Exception will occur and will lead to:

- Exception handling
- Exception suppression

Approach 1: Fork application before accessing memory location

- Child accesses memory location prepares cache status and crashes
- Parent does not access memory location but reads out state via side-channel

Approach 2: Install a signal handler

Approach 3: Use transactional memory instruction

- Transactional memory instruction groups memory into one atomic operation
- Upon exception, architectural state is reset
- Execution continues...
- but micro-state remains and may be read out

3. Diagnosis, Therapy and Side-Effects

What can we do about it?

Vulnerable Hardware

Intel Processors:

- Most core-family processors vulnerable for Spectre and Meltdown
- Early Atom processors not vulnerable

ARM Processors:

- Wide range, depending on specific architecture

AMD Processors:

- Vulnerable for Spectre
- Secure against Meltdown
- Due to use of privilege layers on the paging architecture

General Therapy

Approaches:

- Improve isolation of kernel memory from user memory
- Reduce amount of speculation
- Reduce accessibility of side-channels

Problem:

- Damages performance enhancing features.
- Reduces the performance.

Level of Interventions (1)

Processor design:

- Design fault of the processor
- Requires complete overhaul of CPU architecture
- Needs fresh CPU design

Processor microcode:

- Complex CPUs are in large parts programmable via microcode
- Many instructions are even implemented in microcode
- Microcode can be patched
  Example: Adjust working of instructions
  Example: Introduce new instructions

Level of Intervention (2)

Operating system:

- Adjust memory management system
- Store important secrets in special protected areas

Application programs:

- Reduce precision of timer available to application code
- Design without secrets in memory
- Reduce the time secrets are available in memory
- Further protect secrets in memory (eg: encryption)

Example: Browsers

- Particularly vulnerable, since they run a lot of remote code
- Realistic intervention, since
- Granularity up to 20 microsec from 5 microsec
- Jitter of up to 20 microsec
- Implemented by Edge, Explorer, Firefox and Chrome

Mitigation of Problem (1)

Dual Memory Mapping:

- Do not map kernel memory into user memory
- Switch to a different memory mapping for kernel and user
- Can have unwanted side-effects on some software

Make it difficult to use result

- How would user exploit the information?
- Entire kernel memory dump interesting but not very helpful for attacker
- Randomize memory layout so as to not hint the attacker to useful data

Reset Branch Prediction:

- On context switch, reset status of CPU branch predictors

Mitigation of Problem (2)

Speculation Barrier:

- Implement speculation barrier as a new instruction
- Insert speculation barrier in particular situations

Remove Cache Entries:

- Now and then clear the caches
- Devise different cache architectures

Side Effects of OS-Patches

Studies showed:

- Reduce speculative execution
- Increase memory overhead on kernel-user context switch
- Performance goes down, depending on load
- Studies reference 2-30% depending on load
- Up to 30% for programs doing only kernel calls

Appendix

Contents of Appendix

List of Codes

List of Figures

Terms of Use (1)

Terms of Use (2)

Disclaimer: Die sich immer wieder ändernde Rechtslage für digitale Urheberrechte erzeugt ein nicht unerhebliches Risiko bei der Einbindung von Materialien, deren Status nicht oder nur mit unverhältnismäßig hohem Aufwand abzuklären ist. Ebenso kann den Rechteinhabern nicht auf sinnvolle oder einfache Weise ein Honorar zukommen, obwohl deren Leistungen genutzt werden.

Daher binde ich gelegentlich Inhalte nur als Link und nicht durch Framing ein. Lt EuGH Urteil 13.02.2014, C-466/12 (Pressemitteilung, Blog-Beitrag, Urteilstext). ist das unbedenklich, da die benutzten Links ohne Umgehung technischer Sperren auf im Internet frei verfügbare Inhalte verweisen.

Wenn Sie diese Rechtslage stört, dann setzen Sie sich für eine Modernisierung des völlig veralteten Vergütungs- und Anreizsystems für urheberrechtliche Leistungen ein. Bis dahin klicken Sie bitte auf die angegebenen Links und denken Sie darüber nach, warum wir keine für das digitale Zeitalter sinnvoll angepaßte Vergütungs- und Anreizsysteme digital erbrachter Leistungen haben.

Zu Risiken und Nebenwirkungen fragen Sie Ihren Rechtsanwalt oder Gesetzgeber.

Weitere Hinweise finden Sie im Netz hier und hier oder hier.

Citing This Document

If you use contents from this document or want to cite it,
please do so in the following manner:

Clemens H. Cap: Meltdown and Spectre
Hardware-Centered Attacks. Electronic document. https://iuk.one/1033-1311 4. 7. 2023.

Bibtex Information: https://iuk.one/1033-1311.bib

```
@misc{doc:1033-1311,
   author      = {Clemens H. Cap},
   title       = {Meltdown and Spectre
Hardware-Centered Attacks},
   year        = {2023},
   month       = {7},
   howpublished = {Electronic document},
   url         = {https://iuk.one/1033-1311}
}
```

Typographic Information:
Typeset on ?today?
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020) kpathsea version 6.3.2
This is pgf in version 3.1.5b
This is preamble-slides.tex myFormat©C.H.Cap

List of Slides

Legend:
⧉ continuation slide
○ slide without title header
🖼 image slide