TODO

TODO

Bachelor Thesis

by

TODO

in fulfillment of requirements for degree

Bachelor of Science (B.Sc.)

submitted to

Rheinische Friedrich-Wilhelms-Universität Bonn

Institut für Informatik IV

Arbeitsgruppe für IT-Sicherheit

in degree course

Computer Science (M.Sc.)

First Supervisor: TODO
University of Bonn

Second Supervisor: TODO
TODO

Sponsor: TODO
University of Bonn

TODO

Abstract

TODO

Contents

1 Introduction **1**

2 Catchy Lab title **3**

2.1 Abstract . . . 3

2.2 Introduction . . . 3

2.3 Brief Theoretical Background . . . 3

2.4 maybe Specification of the task . . . 4

2.5 Our Emulator Program/ backend . . . 4

2.6 Our Visualisation and Usage/ Frontend . . . 6

2.7 Demonstration/ evaluation . . . 7

2.8 Conclusion . . . 8

1 Introduction

Contrary to older processors, modern Intel CPUs implement a number of optimization techniques that increase their efficiency. One of which is the concept of out-of-order execution, which takes advantage of the mutual independence of instructions that would normally be executed sequentially. This allows a CPU to continue working on other instructions while one is waiting for data from memory, for example. A second optimization technique is called speculative execution and involves the prediction of branches. Rather than waiting for an instruction that determines which branch is taken, the outcome is predicted. With either technique, the CPU might encounter cases where the current CPU state must be rolled back to a previous one. For out-of-order execution, this happens if an instruction raises an exception (illegal memory access, for example). For speculative execution, this happens if a branch is mispredicted. A rollback causes some instructions that are currently being executed (in-flight) to continue execution for a short amount of time.

Even though rollbacks are meant to make sure in-flight instructions do not cause any lasting side effects on the microarchitectural state of the CPU, it was discovered that they can change the contents of caches and other buffers. The disclosure of both Spectre and Meltdown in early 2018 introduced a whole family of vulnerabilities that take advantage of both out-of-order and speculative execution to leak secrets over the processor's caches. And while the performance losses introduced by software and hardware mitigations are measurable, neither vulnerability can be exploited freely on a fully patched system. As a result, however, the process of trying to exploit one of the vulnerabilities for the sake of learning how they work in detail can be challenging. Apart from the software, which may be obtained by installing an older version of an operating system that does not implement any mitigations, one must also make sure their CPU is effected by the vulnerabilities and has not yet received any relevant microcode updates from Intel. Often times, this means that a user's personal computer does not meet these requirements.

We design and implement a graphical CPU emulator that supports single-step out-of-order and speculative execution and is vulnerable to select variants of Spectre and Meltdown. While it is a simplification in comparsion to real hardware, the emulator allows its users to gain a better understanding of how exactly the two vulnerabilities work and can be exploited. Furthermore, the user may experiment with (ineffective) mitigations or implement their own microprograms that are executed once rollbacks are completed. We supply example programs that can be run by the emulator and serve both as an entry point for the user as well as the basis of our evaluation.

Firstly, chapter 2 briefly gives an overview of relevant components of vulnerable Intel CPUs, presents the concepts of out-of-order and speculative execution in greater detail, and introduces both Meltdown and Spectre to which the resulting emulator is supposed to be vulnerable. Secondly, chapter 3 further describes the target audience of the emulator and which variants of the vulnerabilities have been chosen, while chapter 4 documents the implementation of the emulator by describing

each main component and explaining how rollbacks are implemented. Additionally, it contains an overview of the set of ISA instructions available to the user. Furthermore, chapter 5 explores the graphical user interface by defining the goals its design is supposed to accomplish and documenting design choices and important features. Chapter 6 provides a demonstration of the emulator, which includes example programs, and determines how effective the implemented mitigations are. Finally, chapter 7 summarizes the other chapters, briefly reflects on how valuable the emulator might be to our target audience, and gives ideas for future improvements.

2 Catchy Lab title

2.1 Abstract

2.2 Introduction

```
general task/ goal
    CPU emulator with actual out of order execution, maybe speculative
    execution and comprehensible implementation and documentation
brief context of the task/ goal (e.g. a sentence on Meltdown and why it is
    important to understand it)
structure of the report
```

2.3 Brief Theoretical Background

```
premise according to Felix: reader knows SCA lecture -> brief recaps/
    reminders to reference back to from the other chapters

### really brief introduction to CPU

    multiple components
        maybe mostly via picture

    how they work together

    maybe data-flow from instruction to result

    a lot of hypothticals due to trade secrets

    suggested literature:
        maybe Gruss Diss. and other works on cbsca
        maybe textbooks
        maybe CPU wiki for pictures

### out-of-order execution

    a bit more in depth b.c. this is not content of the sca lecture

    maybe why do we need it/ advantages

    brief explanation of Tomasulo and how it works
```

```
26
27      suggested literature:
28          original Tomasulo paper from 1965
29          maybe a more recent/ didactically edited explanation (is this in
        texbooks?)
30
31  ### Meltdown and Spectre
32
33      brief introduction to Meltdown (and Spectre) in general
34
35      slightly more in-depth introduction to the Meltdown attack we picked
        and want to run in our emulator
36
37      highlight the relevant parts of the CPU, maybe mention how they
        interact in Meltdown (and Spectre)
38
39      maybe briefly mitigations and how successfull they are
40
41      suggested literature:
42          ggf. Dissertation Gruss if we mention basic cbsca
43          Gruss et al. papers on Meltdown and Spectre e.g. https://gruss.cc/
        files/meltdown.pdf or https://gruss.cc/files/meltdown_cacm.pdf
44          Canella, Gruss et al. on mititgations https://gruss.cc/files/
        transient-attacks.pdf
```

2.4 maybe Specification of the task

```
1   emulator to execute and teach Meltdown
2       who is the target audience
3       which Meltdown attacks specifically
4       etc. further concretisations
5
6   are there existing solutions/ related works?
7       Felix' old emulator? citable?
8       suggested literature:
9           todo
```

2.5 Our Emulator Program/ backend

```
1   maybe general info e.g. that we wrote it in python, if we used special
        tools/ libraries
2   for everything we implemented:
3           which part of a real life cpu does this model/ how is this done in
        real life cpus
4           briefly how does it work
5           why did we choose to model it like we do
```

```
            nice code structure/ code easy to understand by students
            some features more or less relevant for meltdown
            etc.
        maybe what did we leave out
        challenges?

### CPU Components and our equivalents/ models

    modular setup based on real life CPUs and nice coding conventions

    #### CPU

        contains/ controls the rest

        preparation (loading the program, init the rest)

        ticks

        coordinate rollbacks?

    #### program handling

        instructions

        parser

    #### data handling

        how we model and handle data

        byte, word

    #### scheduling

        BPU

        frontend

    #### memory

        mmu

            how do we store data

            how do we model data access (i.p. wrt Meltdown)

        cache

            what kinds of caches can we represent
```

```
55
56              how do we model if something is cached and the access times (i.
    p. wrt Meltdown)
57
58      #### execution
59
60          Reservation Station/ Slots
61
62          unlimited execution units
63
64  ### Rollbacks/ Exception handling
65
66          in particular wrt making Meltdown possible
67
68          general goal/ concept of rolling back after a misprediction or an
    exception
69
70          snapshots and interaction of the CPU components
71
72  ### ISA
73
74      overview of ISA
75
76      maybe µ-instr. vs. ISA
77
78      reasoning behind the choice of instructions (manageable size and
    instructions (e.g. no divide by zero) balanced with functionality
    particularly wrt Meltdown)
79
80  ### Tomasulo
81
82      our version of out-of-order execution
83
84      where in our program do we implement which components
85
86      what did we leave out/ do differently
87
88  ### config files
89
90      what can be configured without changing the source code
91          why these variables? relevant for Meltdown?
```

2.6 Our Visualisation and Usage/ Frontend

```
1  ### general concept
2
3      goal: UI for the emulator with visualisation of CPU/ memory components
    and their contents
```

```
4
5       in terminal
6           maybe comparison to existing analysis tool/ debugger gdb
7
8       triggers/ controls the actual emulator
9           overview features, e.g. breakpoints, step-by-step and stepback
10
11      maybe which tools/ libraries were used?
12
13  ### features in more detail and their didactic purpose
14
15      breakpoints, step-by-step, stepback and more
16
17      challenges/ design choices during the implementation
```

2.7 Demonstration/ evaluation

```
1   what kind of system do we need/ did we use to run this?
2
3   ## general demonstration
4
5       brief example program showing all the features in a "normal" execution,
         e.g. adding stuff
6
7   ### Meltdown und Spectre demonstration
8
9       #### Example Program Meltdown
10
11          show example program
12
13          maybe compare to example program for real life architecture from
        SCA or literature (Gruss) if available
14
15          explain which Meltdown variant it implements
16
17          briefly highlight which components (we expect to) interact to make
        it work
18
19          how well does it work?
20
21      #### maybe example program spectre
22
23          same as Meltdown
24
25      #### mitigations
26
27          what is possible in our program as is
28
```

```
        how effective are these
            in real life
            in our program

        what would be the necessary steps/ changes to the program for further mitigations
            compare to changes in hardware by the manufacturers
```

2.8 Conclusion

```
recap goals

did we reach the goal of implementing a CPU emulator where a user can perform a Meltdown attack

how many Meltdown attacks are possible?

is Spectre possible?

value to students:
    how easy to use and convenient do we think our program is?
    do we think this will be a good tool for teaching?
    how accessible is it wrt different host architectures?

further work
    more (detailed/ realistic) functionality for more Meltdown and Spectre variants
    more mitigations
    maybe nice to haves wrt to the visualisation/ general UI functionality/ ISA?
```

Statement of Authorship

I hereby confirm that the work presented in this bachelor thesis has been performed and interpreted solely by myself except where explicitly identified to the contrary. I declare that I have used no other sources and aids other than those indicated. This work has not been submitted elsewhere in any other form for the fulfilment of any other degree or qualification.

TODO

TODO