Computing Science

CS4040: Impact of Meltdown & Spectre mitigations

Konrad Dryja

Signed:

Date: October 15, 2019

CS4040 Report

Department of Computing Science **October 2019**
University of Aberdeen
King's College
Aberdeen AB24 3UE

CS4040: Impact of Meltdown & Spectre mitigations

Konrad Dryja

Department of Computing Science
University of Aberdeen

October 15, 2019

Abstract: In my CS4040 report I'd like to evaluate the impact of the discovery of Side-Channel Attacks back in January 2018 with Spectre & Meltdown on the lead. As those were utilizing many of the performance reliant features - which had to be rolled back or squashed due to security concerns. The removal resulted in many calls about grossly deteriorated performance. With the patches already in place, I have gone ahead and removed them to compare before and after performance of one-core and single-core performance in different tasks, such as prime calculation or rendering. I will be aiming to diversify tested hardware and software, including platforms such as Windows, Linux and Intel, AMD.

1 Introduction

With speculative execution no longer considered safe, modern CPU manufacturers were forced to drop these features in favour of enhancing security. But at the same time, it resulted in sacrificing the performance of the chips. Probably the most harming aspect was that speculative execution was at the time an industry standard - after the processor designers hit a ceiling with potential clock speed, being hit by Moore's Law [9]. I will not be focusing on the origin and details of the vulnerabilities, but rather the patches that followed - although will overview the basics in the next section.

Personally, I found the topic the most interesting, as the aftermath is still haunting security researches to this day, since the fault was not the software, but inherent architecture was found to be flawed. Moreover, frequently we are hearing news how the newest vulnerability based on side-channel execution has been discovered - with the definite fix being complete hardware replacement with a chip produced after 2017. This forced Intel, AMD, ARM to release very aggressive patches, greatly hurting the benchmarks.

This has also raised ethical questions - since some of the affected machines suffered as much as 50% drop in performance. In the eyes of law, this could classify as false advertising which naturally was followed by many class-action lawsuits. As stated by Intel in their 2017 Annual report, as of February 2018, they were facing 30 customer faced suits along with two securities [1]. Intel perhaps is the company that was the most under fire, since Variant 1, also known as Meltdown, was mostly apparent in their chips, although Variant 2 & 2 were replicated in almost all consumer chips.

Computing community was faced with a difficult dilemma - how much of security are we willing to sacrifice in favour of increased performance?

2 Background and related work

When speaking about Spectre and Meltdown, it's very important to start from the very beginning - when in July 2017, a researcher Jann Horn from Google's Project Zero has discovered the vulnerability. Due to the severity and potential implications resulting from premature releasing of the findings, those were first communicated directly - on NDA basis - with manufacturers, hoping for an immediate fix. On January 2018, two papers were released by J. Horn et al. illustrating in-depth the vulnerabilities and how they could be replicated [6, 5]. The papers present a throughout overview of the potential attack - the exploitation here is based on **Branch Prediction** (BP) along with **Out of Order Execution** (OOE). Those are optimizations techniques used by almost all CPUs on the market:

- BP lets the processor "predict" direction where the program's execution flow will go towards without explicitly evaluating the condition. For example, in a situation where **if** statement was successful for 100 iterations, it can assume that 101st will be successful as well and thus prematurely execute the included code-block, storing the result in L-cache (high speed, low capacity memory located directly on the CPU)
- OOE, on the other hand will often reorder scheduled operations leaving the most time-consuming actions till the end, executing the ones containing required data present in CPU cache immediately - assuming that those do not depend on each other, e.g., it's a simple summation, not utilizing prior information.

Together they create a cheap and clever way to speed up the execution of binaries, but unbeknownst opened a pathway for side-channel attacks, exploiting the fact that CPU was executing the code that it wasn't meant to in the first place. Normally, the results are only stored in fast-access L-cache (which is only accessible from kernel-mode and is purely used to increase performance). A proof-of-concept presented by researches illustrated how it's possible to measure how long it takes to access particular variable in order to determine whether it's coming from RAM or L-cache and then make statistical assumption whether the piece of data was evaluated inside of block of code which was executed speculatively.

There is also a subtle difference between the nature of attack described by **Meltdown** (Variant 3) and **Spectre** (Variant 1, 2). The former allowed for any arbirtary process to access kernel-space memory, which contains restricted data such as password or even latest keystroke information. The latter - on the other hand - was capable of crossing process or sandbox boundaries, effectively nullyfying the protections provided by virtualised environment or JIT (just-in-time) compilers. This potentially might allow an attacker to deploy malicious binary onto VM cluster (run by a cloud provider such as GCP or AWD) and access information of other users.

Listing 1: Meltdown PoC

```
char testArray[256 * 64];
evictCache(testArray); // Clear L-cache
char x = * kernelSpace; // Will cause segmentation fault
testArray[x * 64]++; // Will be executed speculatively
for(int i=0; i<256; i++) {
  // Measure how long it takes to access the element
  if(is_cached(testArray[i * 64]) {
    // Cache hit! Found secret bit.
  }
}
```

Multiple patches have been created ever since to mitigate the negative effect, trying to minimize the impact on performance. M. Löw [7] provides an overview of created patches and affected hardware. The major and immediate patches which had the biggest performance effect included:

KAISER [3] - which stands for Kernel Page-Table Isolation. Meltdown is exploiting the fact that for the purposes of reduced access times, the entire kernel address page is mapped to every process. This is not directly accessible from user-mode and is protected by privileged bit. Unfortunately, with side-channel attacks, this data can be speculatively loaded and then later retrieved via a covert channel. KAISER aims to drop that mapping and leave only necessary signals.

Retpoline [10] - retroactive trampoline, it's a mitigation aiming to catch Branch Predictor in an infinite loop, effectively never speculatively executing sensitive code. It is also important to mention that software mitigation can only be helpful to a specific point. From month to month more ways of performing side-channel attacks are discovered, bypassing completely the mitigations, thus manufacturers are force to deploy more and more restrictive patches, decreasing the benefits of BP and OOE.

3 Research question

Having introduced the basics of side-channel attacks in previous sections, we are now ready to tackle the implications of the released patches. The research question that would be formed following the aforementioned considerations would be **"How significant the performance degradation is after mitigations for Side-Channel Attacks were deployed and are they worth the gained security"**.

Fortunately, majority of the flaws were already patched with immediate effect, meaning that the vulnerability cannot be replicated on up-to-date software. This leaves us with necessity of either simulating the behaviour via software or rolling back the updates. To this day, Dell claims that no malicious use has been spotted to this day [4], which introduces question of how severe the incident was and whether the sacrifice was worth it. We can speculate that this state and others pushed the Linux team to introduce an option for kernel versions starting from 5.2 allowing the users to disable the mitigations, regaining the performance but opening the system for potential attacks

Listing 2: Disabled mitigations

```
% lscpu
[...]
Model name:                      Intel(R) Core(TM) i5-6500 CPU
[...]
Vulnerability L1tf:              Mitigation; PTE Inversion;
Vulnerability Mds:               Vulnerable; SMT disabled
Vulnerability Meltdown:          Vulnerable
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Vulnerable:
Vulnerability Spectre v2:        Vulnerable, IBPB: disabled
```

For Windows, on the other hand, the mitigations are slightly harder to disable, although it can be achieved by tweaking Windows Registry [8]. While the impact on consumer Windows version can be negligible (according to Microsoft), Server edition of their Operating System was particularly susceptible to slowdowns, especially during Input/Output operations. Microsoft decided to leave the decision of enabling the patch up to system administrators, leaving the option enabled by default. Alas, mitigations for CVE-2017-5753 (Spectre Variant 1) cannot be disabled for Windows at the moment. To work around this, I will install Windows 10 Version 1607 (From April 2016) and perform all experiments in that environment - with no internet connection to avoid automatic updates. Although this might also cause inconsistency, as newer revision might also include unrelated performance fixes - thus the test will have to be performed on both older revision of Windows 10 and Windows Server with only disabled mitigations.

At this point, with a vulnerable system, we have several ways to obtain before-after comparison of performance, as mentioned: each platform offering different solutions. Linux has open access to compiled byte-code with an option of disabling optimizers potentially spoiling the results. Usually, to benchmark the performance of a CPU, we would be looking at how quickly it can solve complex operations. Example the being prime numbers calculations or matrices multiplication. There are two open-source packages which wrap around those operations, producing a concise report on performed CPU Cycles, page faults etc - stress-ng [11] and sysbench [2]

With Windows, it is slightly more difficult to access direct CPU registers from the OS - additionally, all the compiled code is subject to many optimizations, which might introduce variance and inconsistencies in the results. The most efficient way would be to utilize closed-source, proprietary software created specifically for benchmarking, which can extract raw performance data. Most popular choice nowadays is Cinebench along with 3DMark which at the same time could determine whether the slowdowns in CPU could also be affecting external GPU.

4 Experimental Design

What are your hypotheses? How are you going to test them? What is your target population? What are your datasets; i.e. your sample of the target population. What are the dependent and independent variables?

Guide length: 500 words.

5 Discussion

What do the results say? What have you learned from the experiments? Have you identified a correlation between variables, or causation? What are the limitations of what you've done? What further experiments might be of benefit?

Guide length: 400 words.

6 Conclusion

What have you done and why? What have you shown through your experiments?

Guide length: 100 words.

References

[1] Intel Corporation, 2017 Annual Report. `https://bit.ly/2pkjeqq`, February 2018.

[2] akopytov. sysbench, 2019. `https://github.com/akopytov/sysbench`, Last accessed on 2019-10-11.

[3] Jonathan Corbet. The current state of kernel page-table isolation, 2017.

[4] Dell. Microprocessor side-channel vulnerabilities (cve-2017-5715, cve-2017-5753, cve-2017-5754): Impact on dell products, 2018. `https://www.dell.com/support/article/uk/en/ukdhs1/sln308587`, Last accessed on 2019-10-11.

[5] 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. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[6] 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. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[7] Marc Löw. Overview of meltdown and spectre patches and their impacts. *Advanced Microkernel Operating Systems*, page 53, 2018.

[8] Microsoft. Windows server guidance to protect against speculative execution side-channel vulnerabilities, 2019. `https://support.microsoft.com/en-us/help/4072698/windows-server-speculative-execution-side-channel-vulnerabilities-prot`, Last accessed on 2019-10-11.

[9] Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[10] Paul Turner. Retpoline: a software construct for preventing branch-target-injection. *URL https://support. google. com/faqs/answer/7625886*, 2018.

[11] Ubuntu. stress-ng, 2019. `https://kernel.ubuntu.com/git/cking/stress-ng.git/`, Last accessed on 2019-10-11.