Detecting and Patching Transient Execution Side Channels in an Out-of-Order RISC-V Core

Tobias Jauch[1] *, Alex Wezel[1], Mohammad Rahmani Fadiheh[1], Dominik Stoffel[1], Wolfgang Kunz[1]

[1]Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau

Abstract

Transient Execution Side Channels in modern hardware systems are particularly hard to detect and even harder to mitigate efficiently. Our case study on the open-source out-of-order processor SonicBOOM highlights some of the difficulties and pitfalls involved. We detected that a vulnerability to Meltdown has been re-introduced to the design by minor hardware updates and that closing this gap can easily leave certain channels open or even introduce new ones, e.g., channels that are based on speculative interference. Furthermore, we show how these channels can securely be mitigated with the help of a formal verification tool.

Introduction

Transient Execution Side Channel (TES) attacks, such as Spectre [1] and Meltdown [2], exploit side effects of instructions that are executed transiently, i.e., they are are executed speculatively and later discarded due to misspeculation or an earlier exception. This phenomenon is only visible at the cycle-accurate microarchitectural level, which exacerbates the challenge of detecting it in a hardware design.

In this work, we present our experiences with detecting and fixing Meltdown- and Spectre-type vulnerabilities in a complex out-of-order core. With our case study on the open-source BOOM design [3] we demonstrate the challenges of patching hardware against TES attacks. All of the vulnerabilities reported in this work are originally found using the formal security verification technique UPEC [4], highlighting the importance of formal verification for ensuring security.

Case Study on BOOM

Meltdown-Type Vulnerabilities

While the developers presented BOOMv2 as being secure against Meltdown, our formal analysis showed that seemingly innocuous design updates applied to the load-store unit (LSU) of BOOMv3 (SonicBOOM) have made the design vulnerable to Meltdown. The updated design conforms to the ISA specification and misaligned loads and page faults do raise exceptions as intended. However, these exceptions are only sent to the core itself (signal (1) in Fig. 1) and are handled when the corresponding load instruction arrives at the head of the reorder buffer. In the meantime, the corresponding read request may be sent to the data cache, thereby opening a timing side channel. This is due to a minor design update in SonicBOOM that removed the exception flag from the load queue and does not consider it when sending new read requests.

After this discovery, we informed the development team, who confirmed the gap. Fig. 1 shows the implemented patch that re-introduces the exception flag to the load queue

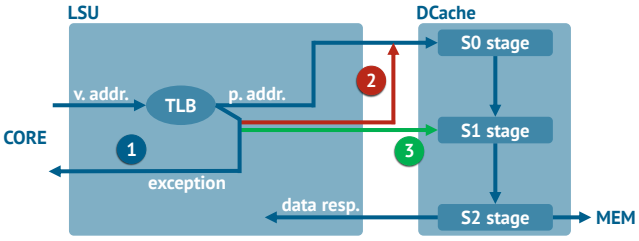

Figure 1: *Load-Store Unit in BOOM and implemented Meltdown mitigations*

and uses this flag to stop faulty loads from being sent to data cache in order to mitigate the vulnerability (signal (2) in Fig. 1). A designer might assume the vulnerability to be fixed now and no additional analysis were necessary. However, further formal analysis with UPEC showed that the design is still vulnerable to Meltdown-type attacks, due to the specific implementation of the LSU in SonicBOOM.

The first design patch ensures that the load instructions from the load queue with asserted exception flag can no longer reach the data cache. However, in SonicBOOM, in case the translation lookaside buffer (TLB) and data cache are available, the LSU prioritizes newly dispatched loads over the loads inside the load queue and directly sends them to the TLB. If address translation is successful, the newly dispatched (and now translated) load operation is sent to both the load queue and the data cache simultaneously. The first patch only blocks faulty loads in the load queue from being sent to the data cache. However, this does not cover the above scenario for newly dispatched loads. Implementing the same check for faulty loads that can reach the data cache directly from TLB closes this gap. This patch successfully secured SonicBOOM against Meltdown-type vulnerabilities, as verified by UPEC in our case study.

Spectre-Type Vulnerabilities

We also conducted a formal security analysis on SonicBOOM for speculative execution of privileged-mode software (e.g., kernel software executed in supervisor mode). Speculative execution in privileged mode allows an attacker to trick the core into speculatively accessing secret data

*Corresponding author: jauch@eit.uni-kl.de

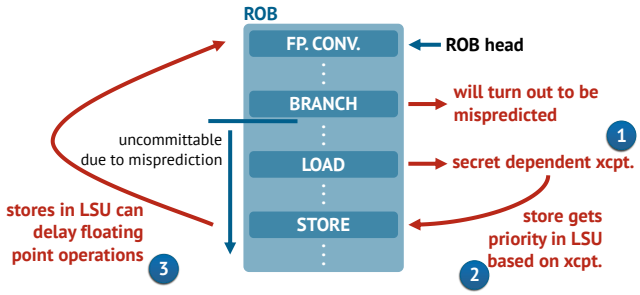

Figure 2: *Speculative Interference Vulnerability in FPIU*

within a privileged victim process, which is the basis for Spectre-type attacks.

Beside classic Spectre attacks, we also discovered a new variant exploiting vulnerabilities similar to speculative interference [5] and Spectre-STC [4]. Speculative interference attacks use transient, secret-dependent operations to influence the timing of older operations that will definitely commit.

Formal security analysis in SonicBOOM by UPEC found an execution trace in which a *speculative secret-dependent load* can delay a non-transient floating point operation at the head of the ROB. This was made possible by a design flaw in one of BOOM's functional units. The floating point to integer unit (FPIU) is responsible for both floating point to integer conversion instructions and floating point store instructions. In this unit, the ready bit of the interface is set only if both of these tasks can be performed, i.e., if the FPIU is not already busy with another operation and the LSU is ready to receive store operations. UPEC showed an interesting Spectre variant exploiting this seemingly innocent design flaw.

Let us assume a speculative attack scenario as depicted in Fig. 2. Within a (mis-)speculated sequence there is a secret-dependent load (i.e., a load instruction that uses speculatively accessed secret as its address operand) from the load queue and a newly dispatched store that both need to access the TLB. The load instruction has the higher priority. However, if the accessed address is misaligned (1), the load is aborted and the store is prioritized (2). Since the target address of the load is a function of the secret, the priority of the store also depends on the secret. The store is now able to block and delay a non-transient floating point to integer conversion because the FPIU does not accept requests until the store queue in the LSU becomes available again (3).

This example shows how small design decisions in different modules can lead to potential covert channels that are almost impossible to detect without formal methods. We decided to tackle this vulnerability with a dedicated fix in the FPIU, so that conversions now can be executed independently of floating point stores.

New Side Channels in the Patched Design

Although the FPIU vulnerability was caused by an implementation detail inside the FPIU, the patched SonicBOOM design suffers from a more global speculative interference problem inside the load and store queue of the LSU. UPEC counterexamples showed that preventing faulty loads from being sent to the data cache interface (S0 stage in Fig. 1) and choosing a different operation to be processed creates a secret-dependent timing behavior in the system. Speculative secret-dependent loads can influence the timing of non-transient operations by creating contention over different resources in the memory subsystem. We solve this problem by shifting the intervention of our Meltdown mitigation mechanism by one cycle. Instead of being prevented from reaching the data cache, faulty memory requests are now killed by signal (3) in Fig. 1 before propagating to the last stage inside the data cache. With this mechanism, given that the cache interface is fully pipelined, faulty loads do not affect the scheduling of memory requests to the data cache. A faulty load occupies the data cache port for one clock cycle, but in the following cycle the port is available again regardless of whether or not the load was aborted in the S1 stage. Meltdown attacks are still prevented since faulty loads can never access the data array inside the data cache (S2 stage).

Conclusion

The findings of this case study clearly show the challenges of implementing mitigations to transient execution attacks in modern out-of-order cores. When it comes to complex designs, there is a high chance of missing leakage paths or even introducing new ones. Being aware of and giving consideration to all possible consequences is a very demanding task for designers. At this point, the capabilities of formal tools like UPEC can provide a significant advantage over standard design flows and testing, as they provide the opportunity to iteratively patch and verify the design. This way, vulnerabilities cannot only be discovered but it is also ensured that implemented mitigations do not lead to new weaknesses.

References

[1] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. "Spectre Attacks: Exploiting Speculative Execution". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019.

[2] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. "Meltdown: Reading Kernel Memory from User Space". In: *27th USENIX Security Symposium (USENIX Security 18)* (2018), pp. 973–990.

[3] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. "SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine". In: *Fourth Workshop on Computer Architecture Research with RISC-V* (2020).

[4] Mohammad Rahmani Fadiheh, Alex Wezel, Johannes Müller, Jörg Bormann, Sayak Ray, Jason M. Fung, Subhasish Mitra, Dominik Stoffel, and Wolfgang Kunz. "An Exhaustive Approach to Detecting Transient Execution Side Channels in RTL Designs of Processors". In: *IEEE Transactions on Computers* 72.1 (2023), pp. 222–235.

[5] Mohammad Behnia, Prateek Sahu, Riccardo Paccagnella, Jiyong Yu, Zirui Neil Zhao, Xiang Zou, Thomas Unterluggauer, Josep Torrellas, Carlos Rozas, Adam Morrison, et al. "Speculative Interference Attacks: Breaking Invisible Speculation Schemes". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '21. 2021, pp. 1046–1060.