Replicating and Mitigating Spectre Attacks on a Open Source RISC-V Microarchitecture

Abraham Gonzalez*
abe.gonzalez@berkeley.edu
University of California, Berkeley

Ben Korpan*
bkorpan@berkeley.edu
University of California, Berkeley

Jerry Zhao*
jzh@berkeley.edu
University of California, Berkeley

Ed Younis*
edyounis@berkeley.edu
University of California, Berkeley

Krste Asanović
krste@berkeley.edu
University of California, Berkeley

ABSTRACT

Recent revelations of new side-channel vulnerabilities in modern processors has made hardware security a first-order concern in processor design. We demonstrate how the Berkeley Out-of-Order Machine (BOOM), a generic open-source out-of-order RISC-V processor, is useful for studying the performance and security implications of microarchitectural mitigations for side-channel attacks. Two results are presented. First we replicate several basic variants of Spectre attacks which exploit the effects of speculative execution in the L1 data cache. We then implement a preliminary hardware mitigation for such attacks, demonstrate its effectiveness, and measure its impact on performance, and area. Compared to the baseline processor, our mitigation displays a 2% IPC improvement, a 2.5% area increase, and a 0.36% clock reduction in a 45nm process. To our knowledge, our work is the first available demonstration on an open-source RISC-V processor of speculative side-channel attacks and a potential hardware mitigation. Our methodology demonstrates the value of the open-source RISC-V hardware ecosystem for secure hardware research.

CCS CONCEPTS

• **Security and privacy → Security in hardware**.

KEYWORDS

security, RISC-V, out-of-order processor

ACM Reference Format:
Abraham Gonzalez, Ben Korpan, Jerry Zhao, Ed Younis, and Krste Asanović. 2019. Replicating and Mitigating Spectre Attacks on a Open Source RISC-V Microarchitecture. In *Proceedings of Third Workshop on Computer Architecture Research with RISC-V, June 22, 2019, Phoenix, AZ*. ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

In early 2018, the disclosure of the Spectre [10] and Meltdown [12] attacks demonstrated a common class of security vulnerabilities in modern general-purpose high-performance processors. Following these revelations, a wide range of similar attacks have surfaced. These attacks leverage processor behaviors including branch prediction [11, 13], caching [9], floating-point units [18], virtual addressing [20, 22], and speculative execution [14] to leak information across secure boundaries. Since these behaviors are necessary for performance in a modern processor, attacks exploiting these behaviors motivate computer architects to develop microarchitectural mitigations which preserve performance.

Existing research on mitigations has largely leveraged architectural simulation, instead of RTL implementation. Traditionally, architectural simulation has been favored over RTL implementation for computer architecture research due to the relative ease of modifying architectural simulators, and due to the sparsity of open-source RTL for high-performance processors. However, since architectural simulation generates only estimates on the power, performance, and area implications of new proposals, it provides only a partial evaluation of the trade-offs of a proposed hardware mitigation.

Rising trends in computer architecture present new methodologies for highly-productive architecture research which would enable hardware-security researchers to work at the RTL level. Uptake of the RISC-V open ISA provides a well-supported hardware and software ecosystem, with open-source tools and documentation. The trend towards agile hardware design and evaluation also provides an ecosystem of debugging and implementation tools, including Chisel [3], FireSim [7], and HAMMER [21]. This has encouraged the growth in quality of open-source hardware implementations of modern processors, including The Berkeley Out-of-Order Machine [4], Rocket Chip [2], PULPino [19], and riscy-OOO [26].

In this paper, we show the value of RISC-V for hardware security research by demonstrating a methodology for developing, implementing, and evaluating microarchitectural mitigations for security vulnerabilities in an out-of-order superscalar processor. Our study targets The Berkeley Out-of-Order Machine (BOOM), an open source RISC-V implementation of an out-of-order processor that exhibits many of the hardware features and execution behaviors which enable speculative execution attacks.

We contribute the first demonstrated speculative execution attacks on the open-source BOOM processor. These attacks provide a framework for hardware security researchers to introspect on the behaviors of a modern high-performance processor during speculative execution. We also contribute a preliminary RTL mitigation for speculative execution attacks targeting the L1 data cache. Our mitigation is similar in spirit to the InvisiSpec [23] and SafeSpec [8] proposals, which suggest the addition of buffers to protect side-channels from speculative execution. Our evaluation of the mitigation's effect on performance and security demonstrates the

*These authors contributed equally to this paper.

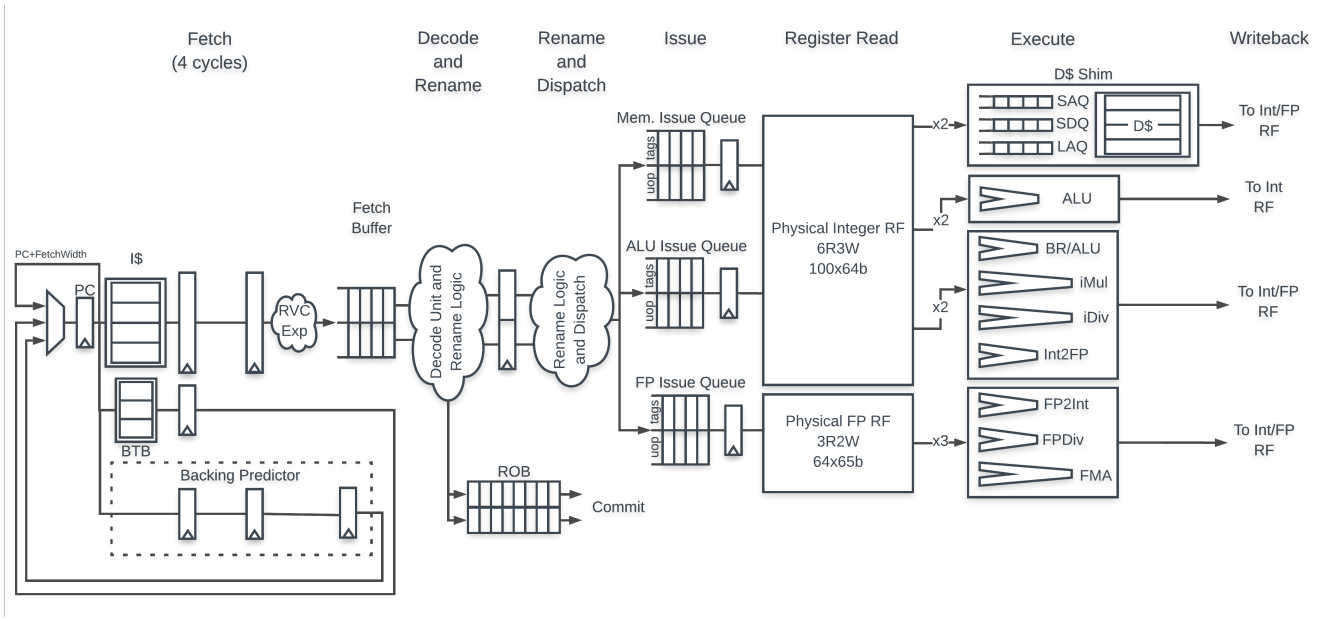

Figure 1: Overview of BOOM Pipeline

usefulness of the open-source RISC-V ecosystem for hardware security research.

The remainder of this paper is structured as follows: In Section 2 we demonstrate how disclosed attacks can be replicated on an open-source processor, specifically BOOM. In Section 3 we demonstrate the process of implementing a simple mitigation for our replicated attacks. In Section 4 we evaluate the performance and security implications of our implemented mitigation. In Section 5 we discuss future work and conclude in Section 6.

2 SPECULATIVE ATTACK REPLICATION

To our knowledge, we provide the first set of open-source implementations of speculative execution attacks on an open-source RISC-V processor, in our case BOOM. As a generic implementation of an out-of-order processor, BOOM provides all of the necessary microarchitectural components for a speculative execution attack to occur. Additionally, BOOM's open source RTL provides full visibility of microarchitectural behaviors during program execution.

2.1 Speculative Execution Attack Components

We now describe the microarchitectural components which enable speculative execution attacks on modern processors, and show how BOOM demonstrates these features.

2.1.1 Branch Predictor Unit. In a modern high-performance processor, the branch predictor lets the processor execute instructions past a unresolved branch, substantially improving performance. Many recently disclosed speculative execution attacks exploit this optimization by training the branch predictors to misdirect the PC during execution of victim code.

BOOM's branch predictor, as shown in Figure 1, is split into a simple two-cycle "next-line predictor" (NLP) and a complex four-cycle "backing predictor". The NLP contains the Branch Target Buffer (BTB) where the PCs and targets of recent branches are cached. The NLP also contains the Return Stack Buffer (RSB) which holds a stack of targets from `ret` instructions. The "backing predictor" is a TAGE [16] or GShare predictor [15, 25] which makes a more accurate prediction based on a global history of branch activity. We designed attacks targeting a GShare predictor since the current GShare predictor implementation performs more reliably than the TAGE predictor implementation.

2.1.2 Speculative Execution. In a modern high-performance processor, the branch predictor instructs the fetch stages to provide a predicted instruction stream to the execution backend. As a result, mispredicted branches might invalidate previously executed instructions, marking them as misspeculated. Register renaming and reorder structures enable recovery from these misspeculations to maintain overall program correctness, while still allowing instructions to execute out-of-order. However, misspeculated instructions may leave behind visible microarchitectural state in the processor, forming side-channels from which attacks can extract information about the results of misspeculated instructions.

BOOM follows the conventional design paradigm of modern out-of-order processors, as seen in Figure 1. The reorder buffer, renaming stages, and issue queues coordinate to enable speculative execution while guaranteeing program correctness.

2.1.3 Caching. In modern processors, multi-level cache hierarchies allow the processor to exploit locality in its memory accesses. These cache hierarchies also present a side-channel for speculative execution attacks. To reduce noise, cache side-channel attacks generally target a large last level cache.

Our configuration of BOOM has a two-level memory hierarchy, with a non-blocking L1 data cache, and an outer memory set to the

latency of L2. While demonstrated attacks have generally exploited the L2 or L3, we show that highly-precise attacks designed with full knowledge of the microarchitecture can still target the L1. Demonstrated attacks have also used lower-level cache misses to expand their speculation window. Since our memory latency is bounded by hit latency to L2, we approximate long-latency memory-bound misspeculations using a `fdiv` dependency chain.

2.1.4 Cache Manipulation. Existing demonstrated attacks have so far exclusively targeted the x86 and ARM ISAs, where cache manipulation instructions allow the attacker to gain control over cache contents [1, 6]. For example, x86 has `clflush` to flush a cache line from every level of the cache hierarchy.

However, RISC-V does not have an equivalent instruction. Therefore, we implement a similar L1 cache flush function, which flushes the an entire cache set, instead of a specific line in the set. The L1 cache flush function evicts the entire set by accessing extra array addresses corresponding to the set `N * L1_WAY` times, where `N` is large enough to account for the random cache replacement policy. We choose `N = 4` in order to have a 99% probability of evicting a specific cache line from L1. While this function does provide a method for the attacker to control L1 cache state, it also degrades the performance of all code on the machine due to extra cache misses.

2.2 Replicated Attacks

Leveraging the characteristics listed in Section 2.1, we implement both Spectre-v1 and v2, otherwise known as the Bounds Check Bypass and Branch Target Injection attacks for BOOM[1].

In the Bounds Check Bypass attack, an "attacker array" is first flushed from the L1 cache using the custom cache flush function. Then a bounds check conditional branch within the victim code is trained to enter a code block that accesses secret information. Next, an attack round gives a value to the victim that fails the bounds check, but during speculation is used to retrieve the value from the cache. This allows the "attacker array" to be scanned and timed to determine the secret value based on the timing of cache hits. This attack pattern is otherwise known as a FLUSH + RELOAD attack [24].

In the Branch Target Injection attack, we instead exploit the outcome of a `jalr` instruction that uses the BTB to predict its destination. In this attack, the attacker trains a BTB entry for a `jalr` in the victim code to point at a function that leaks a secret value through speculation. Similar to the Bounds Check Bypass attack, this attack follows a FLUSH + RELOAD pattern in which an "attacker array" is initially flushed then can be scanned and timed to extract leaked information.

3 SPECULATION BUFFER

The side-channel which underlies many Spectre variants, including those we replicated, is the modification of data cache state during speculative execution. This is due to a behavior common in modern out-of-order processors, in which speculative load misses will still be written into the tag and data arrays. The attack variants described in Section 2.2, carefully measure the execution times of repeated loads allowing attacker code to inspect the state of the cache and infer the destination addresses of misspeculated loads by the victim code.

In our mitigation, we consider the tag and data arrays as part of the architectural state of the machine, since their contents will affect the architectural results of timing measurements performed by attacker code. Similar to how architectural register state is managed in the execution pipelines of out-of-order machines, a secure processor must only allow correctly speculated, committing instructions to modify the cache state. We implement a small "L0 speculation buffer" that holds refill data from speculating load misses, and flushes the data when the load is resolved as misspeculated. This prevents misspeculated loads from affecting the state of the cache, while still allowing correctly speculated loads to broadcast their data into the rest of the machine as soon as possible, maintaining performance.

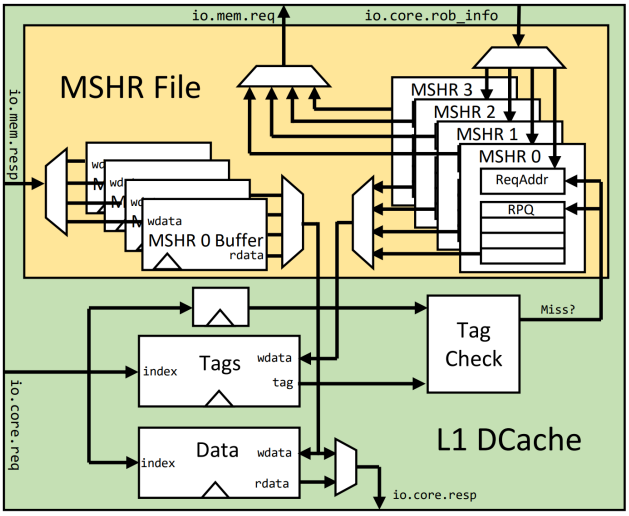

Figure 2: Overview of modified BOOM L1 data cache

3.1 Miss Status Holding Registers

We implement our buffer as part of the Miss Status Holding Registers (MSHRs) in BOOM's L1 data cache. The MSHRs hold the status of inflight memory requests made by the L1 cache to the L2 memory bus. In the original data cache, L2 cache refills would write the refill data into the tag and data arrays before waking up the corresponding MSHRs to return the load data to the processor. To implement the speculation buffer, we modified cache refills to instead write the refill data into per-MSHR cache line buffers, depicted in Figure 2.

An old cache line is evicted and a new cache line is written into the cache only after the load which allocated the MSHR entry has committed. On misspeculation, the speculatively allocated MSHR entry is flushed along with any load data that has been returned. This prevents misspeculated loads from altering the state of the cache.

We enable bypassing of refill data out of the buffer, before the data is committed. Consequently, the service time for a cache miss

[1]The source code for these replicated attacks is available at https://github.com/riscv-boom/boom-attacks.

is slightly reduced compared to the original behavior; data can be forwarded out prior to the updating of the cache data and tags. However, the total MSHR allocation time is longer than with the original behavior.

Another subtlety of the MSHRs is the per-MSHR replay queues. These queues hold consecutive (secondary) load misses to the same cache line while the initial (primary) miss is being handled. Following the acquisition of the requested cache line, our modified MSHRs will eagerly empty the replay queues until a store miss reaches the head. Since stores issued to the memory system are always non-speculative, the cache line can be immediately committed at this point.

A final subtlety is the MSHR eviction policy which prevents deadlock conditions. Consider a branch which depends on a load for resolution. This critical load, and many loads which follow the branch, will miss to different cache lines. The loads following the branch are then issued speculatively before the critical load, filling the MSHR file. Once the miss data for each of these loads is returned and buffered, the MSHRs will wait for their speculative status to resolve. As a free MSHR entry is required to complete the critical load which the branch depends on, this scenario would result in deadlock. To solve this, we allow an MSHR to be evicted if still marked as speculative after receiving its refill data and draining its replay queue. If the branch proved to be speculated correctly, then any following loads to the same cache line would miss, causing a performance decrease.

3.2 Point of No Return

In the base implementation of the speculation buffer, entries in the buffer are only deallocated when their corresponding instructions are reached by the head of the reorder buffer (ROB). However, this can result in heavy MSHR utilization, as many instructions may reside between a waiting load and the commit head. However, we observe that many of those instructions, while still inflight, can be marked as guaranteed to commit.

This informs the concept of a "point-of-no-return" (PNR) in the ROB, in addition to the commit head. While the commit head tracks the next instruction which will update the committed architectural state, the PNR tracks the oldest instruction which may cause misspeculated execution, such as a branch which has not yet executed. We observe that refills can be committed to the cache as soon as the PNR passes the instruction which triggered them, since the PNR guarantees that this instruction will eventually commit. This reduces pressure on MSHR resources and prevents backpressure on incoming cache requests.

To reduce the performance impacts of our buffer, we implemented a PNR in the ROB. Two versions were implemented. The first "simple-PNR" will mark at most one ROB row per cycle as "guaranteed to commit". We also implemented a more complex "fast-PNR" that can mark an arbitrary number of rows per cycle, essentially "jumping over" groups of safe or completed instructions to the oldest incomplete unsafe instruction. These are illustrated in Figure 3, which depicts a ROB organized into 2 columns as in our BOOM configuration. The position of the PNR pointer on the subsequent cycle is shown for both the simple and fast versions.

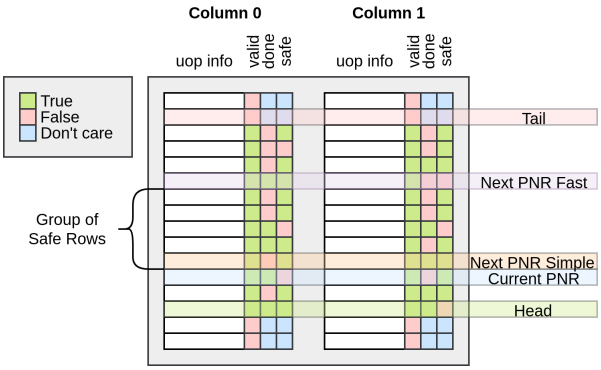

Figure 3: Overview of PNR in ROB

Table 1: BOOM Processor Parameters

Parameter	Value
L1 Sets	64
L1 Ways	8
L1 Linesize	64 bytes
MSHR File Entries	4
BTB Sets	512
BTB Banks	2
BTB Ways	4
GShare History Bits	23
GShare Counter Table Entries	4096

4 EVALUATIONS

4.1 BOOM Processor Parameters

The two attacks were replicated on the BOOM processor with the parameters given in Table 1 and 2. Note that our configuration employs 4 MSHRs and thus contains a 4-entry "L0 speculation buffer". We have used the "simple" version of the PNR for the results (Section 3.2), as the fast version has not yet provided a measurable performance increase. All attacks were measured using FireSim, an open-source cycle-accurate, FPGA-accelerated multi-scale simulation platform [7].

4.2 Replicating Speculative Attacks Results

Overall, the attacks chosen were replicated successfully on the BOOM microarchitecture. Table 3 shows the results of the two attacks in reading a single secret byte or overall speed in bytes per second. These measurements take into account the clearing of the tally array before each run, the multiple rounds of training for the BPU, the single attack run on the victim, and the time to measure out the secret from the attacker array. The cycle counts are close to each other because the code shares a similar structure. The differences stem from the setup of the `fdiv` manipulation and additional arithmetic in the Branch Target Injection attack.

Table 2: Attack Parameters

Parameter	Value
Cache Hit Threshold	50 cycles
Number of runs on same byte	10 rounds
Training rounds for BPU	6 training rounds
Cache flush hits on same set	4 * L1_WAYS
GShare Counter Table Entries	4096

Table 3: Speculative Attack Results

Attack	Cycles per Secret Byte	Bytes per Second 100 MHz	Bytes per Second 3.2 GHz
Bounds Check Bypass	884485	113 B/s	3618 B/s
Branch Target Injection	876602	114 B/s	3650 B/s

4.3 Speculation Buffer Results

We evaluated our speculation buffer implementation using Dhrystone, the replicated attacks, and three small microbenchmarks. Our microbenchmarks stressed the MSHR blocking and eviction conditions that slow down the machine while Dhrystone was chosen as an initial general performance test. The speculation buffer shows an expected small decrease in performance in the microbenchmark tests while Dhrystone shows a small improvement all while providing a defense against the replicated attacks demonstrated. A table of results is shown at Table 4.

4.3.1 Microbenchmark Results. The first microbenchmark named "Non-speculative load misses to same sets" was used to test how efficient non-speculative loads were when the MSHR was unable to allocate entries for each new load. This microbenchmark is a series of loads that had different tags but the same index, which would all request one specific MSHR in BOOM's data cache. When running with our speculation buffer, the refill data must first fill the MSHR buffer before the cache. This causes a small performance decrease for each load miss. Since our benchmark has 16 loads, this results in a performance decrease of around `16 loads * (CACHE_LINE_SZ / DATABUS_WIDTH)` or `16 * (64B / 8B) = 128 cycles`.

The second microbenchmark named "Non-speculative load misses to different sets" is similar to the first except that each subsequent load is accessing a different set in the cache. In the normal version of BOOM, the MSHRs would fill up completely but then stall when full until a prior MSHR entry would be freed (when a previous load is completed). When using the speculation buffer, there is a higher performance penalty because the data must fill the MSHR buffer with the cache data before filling the cache. Thus, there is an extra overhead of around 8 cycles for every 4 loads since the MSHR's are allocated waves of 4 and the fill latency to the MSHR buffer is `CACHE_LINE_SZ / DATABUS_WIDTH` or `64B / 8B = 8 cycles`.

The final microbenchmark named "MSHR evicted speculative load miss" is used to test the impact when a load has to be evicted from an MSHR due to a potential deadlock condition mentioned in Section 3.1. This microbenchmark shows the increased latency of a load to the evicted cache line following this scenario. The large performance penalty associated with this scenario is not surprising, since the evicted load needs to completely reissue through the load-store unit.

Table 4: Speculation Buffer Results

Benchmark	Version of BOOM: normal	Version of BOOM: w. spec. buffer	% Diff.
Non-spec. LD misses to same sets	540 cycles	640 cycles	-19%
Non-spec. LD misses to diff. sets	264 cycles	297 cycles	-11%
MSHR evicted spec. LD miss	48 cycles	67 cycles	-40%
Dhrystone	2176 Dhrys./s	2216 Dhrys./s	+2%

4.3.2 Dhrystone Results. Enabling the speculation buffer granted a 2% increase in Dhrystone performance, as described in Table 4. There are several factors which may contribute to this small performance improvement. The speculation buffer delays the eviction of old cache lines until the refill is known to commit, allowing hits on the old cache line in the intervening cycles. Additionally, the speculation buffer decreases the latency of missed loads: this is because refill requests can be sent to the bus earlier, and the returned data can be forwarded out of the buffer as soon as it is available, rather than waiting for the update of cache metadata.

4.3.3 Synthesis Results. Trial synthesis of BOOM in TSMC 45nm was performed with HAMMER, a framework for running synthesis and place-and-route [21]. The addition of our 4-entry speculation buffer resulted in a 2.5% increase in area and a 0.36% decrease in clock frequency. These differences are largely attributable to the 4 512-bit register arrays used to buffer incoming cache lines within the MSHR file.

5 FUTURE WORK

5.1 Attack Improvements

The replicated attacks are only a subset of speculative execution attacks that out-of-order microarchitectures are susceptible to. In future work, we plan on both improving the efficiency of our replicated attacks and demonstrating more variants of these exploits.

A simple improvement based on the microarchitecture may be to tweak the attack parameters to improve performance. This includes the attack parameters listed in Table 2. Potentially, the number of rounds on the same byte and training rounds can be reduced to improve the speed of the attack. These can be adjusted in conjunction with the cache flush hits on the same set. Even though the random cache replacement policy may not clear all the ways of the set, the number of rounds on the same byte might remove the false hits when the specified line was not evicted.

5.2 Other Attacks and Configurations

BOOM is susceptible to a variety of other speculative style attacks. One such example is the Return Stack Buffer (RSB) attack. In the

future, we plan on fixing BOOM's RSB so we can demonstrate this exploit. Another improvement would be to train the attacks on different branch predictors. Specifically, we would like to evaluate the efficiency of these attacks on a high-performance TAGE predictor.

5.3 Further Evaluations

We plan to perform a more thorough evaluation of the performance and security implications of the speculation with more workloads, specifically the SPEC2017 benchmark suite [17] and the EEMBC CoreMark benchmark [5]. We plan to configure multiple design points of the speculation buffer with multiple implementations of the PNR head and refill/replay system, and compare the trade-offs of each implementation for security and performance.

5.4 Mitigation Improvements

From our evaluation of the implemented speculation buffer, we have identified several areas for future improvements to this mitigation strategy.

5.4.1 Multi-level Cache Hierarchy. The current speculation buffer addresses only a single level cache hierarchy. BOOM is currently configured with an L1 data cache and a large all-encompassing L2. In the future, it would be valuable to configure BOOM with a more realistic cache hierarchy, including a large L2 and L3. The technique described in this paper could be potentially extended to protect lower-level caches from speculative execution.

5.4.2 MSHR File as a Side-Channel. Two key limitations affect our current MSHR file implementation: MSHRs are not always immediately deallocated after being killed by misspeculation, and only a single miss may be be inflight to a particular cache set at once. These limitations open up more side-channels through which Spectre style attacks could extract information. For instance, when these limitations are combined, the following attack surfaces: an attacker could perform their malicious call on a victim function repeatedly, following each call immediately with a single load which is known to miss, rather than an inspection of an entire attack array. If one of these loads took longer than expected, the attacker could deduce that a killed miss to the same set was being held by the MSHR, waiting for a response from the data bus. From this, the attacker infers part of the address used in the victim's secret-dependent load. This attack may be more difficult to perform than the standard attack, as there is a limited time window in which the MSHR will remain allocated after being killed. Additionally, this attack would be far slower as the victim call and probing sequence needs to be performed an average of $(num\_sets + 1)/2$ times to read out $log_2(num\_sets)$ bits.

6 CONCLUSION

We have replicated Spectre on BOOM, an open source RISC-V processor, showing that it is useful as a baseline for hardware security research. Additionally, a speculative buffer was demonstrated as a basic mitigation for these cache side-channel based speculative attacks. With security as a first order concern in processor design, this work demonstrates how RISC-V and BOOM enables researchers to productively study hardware security vulnerabilities and mitigations in modern processors.

REFERENCES

[1] ARM. 2015. Cache maintenance. *ARM Cortex-A Series Programmer's Guide for ARMv8-A* 1 (2015). infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0024a/BABJDBHI.html

[2] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. 2016. The rocket chip generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17* (2016).

[3] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a scala embedded language. In *DAC Design Automation Conference 2012*. IEEE, 1212–1221.

[4] Christopher Celio, David A. Patterson, and Krste Asanović. 2015. The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor.

[5] Embedded Microprocessor Benchmark Consortium. 2018. Coremark: An EEMBC Benchmark. https://www.eembc.org/coremark/

[6] Intel. 2011. Intel® 64 and IA-32 Architectures Software Developer's Manual. *Volume 3B: System programming Guide, Part 2* (2011).

[7] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, et al. 2018. Firesim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 29–42.

[8] Khaled N Khasawneh, Esmaeil Mohammadian Koruyeh, Chengyu Song, Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2018. Safespec: Banishing the spectre of a meltdown with leakage-free speculation. *arXiv preprint arXiv:1806.05179* (2018).

[9] Vladimir Kiriansky, Ilia A. Lebedev, Saman P. Amarasinghe, Srinivas Devadas, and Joel S. Emer. 2018. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2018), 974–987.

[10] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *CoRR* abs/1801.01203 (2018). arXiv:1801.01203 http://arxiv.org/abs/1801.01203

[11] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. 2018. Spectre Returns! Speculation Attacks using the Return Stack Buffer. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. USENIX Association, Baltimore, MD. https://www.usenix.org/conference/woot18/presentation/koruyeh

[12] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *CoRR* abs/1801.01207 (2018). arXiv:1801.01207 http://arxiv.org/abs/1801.01207

[13] Giorgi Maisuradze and Christian Rossow. 2018. ret2spec: Speculative execution using return stack buffers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2109–2122.

[14] Giorgi Maisuradze and Christian Rossow. 2018. Speculose: Analyzing the security implications of speculative execution in CPUs. *arXiv preprint arXiv:1801.04084* (2018).

[15] Scott McFarling. 1993. *Combining branch predictors*. Technical Report. Technical Report TN-36, Digital Western Research Laboratory.

[16] André Seznec and Pierre Michaud. 2006. A case for (partially) TAgged GEometric history length branch prediction. *Journal of Instruction-level Parallelism - JILP* 8 (01 2006).

[17] Standard Performance Evaluation Corporation (SPEC). 2017. SPEC CPU 2017. https://www.spec.org/cpu2017/Docs/overview.html

[18] Julian Stecklina and Thomas Prescher. 2018. Lazyfp: Leaking fpu register state using microarchitectural side-channels. *arXiv preprint arXiv:1806.07480* (2018).

[19] Andreas Traber, Florian Zaruba, Sven Stucki, Antonio Pullini, Germain Haugou, Eric Flamand, Frank K Gurkaynak, and Luca Benini. 2016. PULPino: A small single-core RISC-V SoC. In *3rd RISCV Workshop*.

[20] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 991–1008.

[21] Edward Wang, Adam Izraelevitz, Colin Schmidt, Borivoje Nikolic, Elad Alon, and Jonathan Bachrach. 2018. Hammer: Enabling Reusable Physical Design. (2018).

[22] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F Wenisch, and Yuval Yarom. 2018. *Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution*. Technical Report. Technical report.

[23] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher Fletcher, and Josep Torrellas. 2018. InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy. In *2018 51st Annual IEEE/ACM International*

Symposium on Microarchitecture (MICRO). 428–441. https://doi.org/10.1109/MICRO.2018.00042

[24] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 719–732.

[25] Tse-Yu Yeh and Yale N. Patt. 1991. Two-level Adaptive Training Branch Prediction. In *Proceedings of the 24th Annual International Symposium on Microarchitecture (MICRO 24)*. ACM, New York, NY, USA, 51–61. https://doi.org/10.1145/123465.123475

[26] Sizhuo Zhang, Andrew Wright, Thomas Bourgeat, and Arvind Arvind. 2018. Composable Building Blocks to Open up Processor Design. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 68–81.