Effect of Spectre and Meltdown Patches on System Performance

Trenton W. Ford, Ryan Karl, Rachel M. Krohn, and Nathan Vance
University of Notre Dame, Department of Computer Science and Engineering
CSE 60321: Advanced Computer Architecture
Email Addresses: {tford5, rkarl, rkrohn, nvance1}@nd.edu

***Abstract*—After benchmarking the system performance of personal use computers running Debian and applying patches to the Spectre and Meltdown vulnerabilities, our analysis indicates that while most aspects of system performance appear to be negligibly affected following the patch, the latency of certain services may be affected. Also, AMD manufactured processors seem experience less performance degradation than comparable Intel hardware after applying the patch.**

I. Introduction

Following the disclosure of the Spectre and Meltdown vulnerabilities on January 3, 2018 [3], public concern has increased dramatically. Since both vulnerabilities arise from a longstanding focus in computer architecture on maximizing system performance, several billion processors are affected [1]. Multiple patches have been released to mitigate security risks without sacrificing performance, but news outlets often report contradictory levels of slowdown without listing sources or the methodology used to gather the experimental results[3][1].

II. Objective and Motivation

Despite the widespread impact of Spectre and Meltdown, currently there is very little data on how existing patches impact performance, though several sources claim that patching the bugs will lead to severely decreased performance [7]. Our primary goal is to benchmark unpatched and patched machines to determine the effect of Spectre and Meltdown patches on overall system performance. Benchmark analysis will include an examination of the effect of the underlying architecture on patch impact. Since our focus is on consumer-level systems, our experiments target desktops and laptops from a variety of processor generations. We are particularly interested in analyzing performance differences between patched and unpatched kernels on Intel and AMD processors.

III. Background

A. Side-Channel Attacks

Cache side-channel attacks exploit cache operation timings to extract secrets that should not be available to an attacker process under normal execution. A common attack is Flush+Reload, in which an attacker flushes the cache, the victim loads shared memory, and the attacker times cache accesses to determine which location was loaded [12, 4]. Another attack is Prime+Probe, where an attacker fills the cache with known addresses, the victim evicts a known address, and the attacker times cache accesses to determine which location was evicted [13, 4]. Overall, the goal of these attacks is to extract private information from victim processes by exploiting architectural features of the cache system.

B. Meltdown

The Meltdown attack "exploits side effects of out-of-order execution on modern processors to read arbitrary kernel-memory locations including personal data and passwords" [5]. This allows for a complete kernel memory dump, which is more powerful than a typical side-channel attack that can only access a specific process's data. Meltdown exploits a singular privilege escalation vulnerability found in modern Intel processors: speculatively executed instructions can bypass memory protections, and any cache modifications caused by these instructions can be detected via cache side-channels [5].

Algorithm 1 Exploit Meltdown CPU Bug Example

```
1: Input: illegal_memory_address to read
2: Output: Contents of illegal_memory_address
3: flush_cache()
4: throw_exception()
5: // The next line will never commit
6: load(probe_array[*illegal_memory_address * 4096])

7: Exception handler:
8: loaded_cache_location = fastest_access(probe_array)
9: return loaded_cache_location
```

In the example given in Algorithm 1 (adapted from [5]), an unprivileged process uses Flush+Reload to read an arbitrary memory address. In an out-of-order instruction, the unprivileged process uses the contents of `illegal_memory_address` to load an index of an array into the cache after an exception is thrown, to avoid causing measurable changes. However, the Meltdown CPU bug allows the illegal access to load an index from `probe_array` into the cache. This index can be recovered in the exception handler, and the contents of `illegal_memory_address` can be extracted. This is a cross-core attack, and allows the spy and the victim to execute in parallel on different execution cores inside different processes.

C. Spectre

Spectre works differently than Meltdown, as it targets a specific victim process by inducing speculative operations to leak confidential information [4]. Processors with speculative execution will execute transient instructions to speed up execution; if the instructions become unreachable due to branching, the results are reverted. Which transient instructions to execute are carefully chosen to bypass security domains and leak information [4].

Algorithm 2 Exploit Spectre CPU Bug Example

```
1: Victim Code:
2: Input: x
3: if 0 <= x < array1_size then
4:     y = array2[array1[x] * 256]
5: end if

6: Attacker Code:
7: Input: victim_memory_address
8: Output: Contents of victim_memory_address
9: // Train branch predictor
10: exec_victim_code(legal_x_val)
11: // Prime cache
12: fill_cache()
13: // Victim evicts cache block dependent on
14: // the contents of victim_memory_address
15: exec_victim_code(victim_memory_address)
16: // Probe cache
17: evicted_cache_location = slowest(probe_cache())
18: return  evicted_cache_location
```

In example Algorithm 2 (see [4]), an attacker has analyzed victim code to determine which instructions can leak secrets. This utility code is referred to as a *gadget*. First, the attacker trains the branch predictor to execute the gadget, i.e. a memory access following a bounds check. The attack then uses Prime+Probe to read victim memory. Then the gadget is executed again, this time with the attacker's target memory address. The victim speculatively executes the gadget, which causes a cache eviction. The attacker can deduce which cache block was evicted by timing cache accesses, and determine the contents of the targeted memory. Note, this is a cross-core, cross-VM attack, and allows the spy and the victim to execute in parallel on different execution cores, inside different processes on different VMs.

D. Spectre vs. Meltdown

Spectre and Meltdown both exploit speculative and out-of-order execution to access privileged information, there are key differences between the attacks. Table I highlights the main differences, including vulnerable hardware, attack method, etc. While Meltdown is only known to impact Intel processors [5], Spectre can potentially effect any CPU with speculative execution, including Intel, AMD, and ARM processors [4].

In practice, Meltdown exploits a vulnerability in Intel's memory protections to read protected memory, while Spectre uses speculative execution to create detectable changes in the cache. Spectre uses branch prediction to induce speculative execution, while Meltdown uses instructions aborted after a trap [4]. The KAISER patch helps to mitigate Meltdown, but does mitigate against Spectre, while the RETPOLINE patch claims to mitigate both without severely impacting performance[10].

E. Vulnerability Patches

There are two existing patches for Spectre and Meltdown, KAISER and RETPOLINE, each based on different protection principles.

1) KAISER: KAISER (Kernel Address Isolation to have Side-channels Efficiently Removed) prevents side-channel attacks by unloading information regarding kernel addresses when running user processes, to achieve Kernel Address Space Layout Randomization (KASLR)[2]. This ensures that hardware has no knowledge of kernel addresses, and is claimed to reduce runtime overhead to roughly 10% [2].

2) RETPOLINE: RETPOLINE is a technique to isolate indirect branches from speculative execution at compile-time to prevent side-channel attacks on the protected code [11]. This prevents branch-target-injection by using an infinite loop that is never executed to prevent the CPU from speculating on the target of an indirect jump, and is claimed to reduce runtime overhead to roughly 2%[11].

IV. Team Effort Distribution

This section presents a brief summary of the tasks performed by each team member. Some tasks, such as analyzing the data, were a group effort and are not listed.

Trenton

- Create and manage project repository.
- Run benchmarks on personal devices.
- Scrape data from benchmark log files.

Ryan

- Write first draft of project proposal.
- Conduct literature review.
- Research, test, and select Phoronix benchmarks.
- Run benchmarks on personal devices.
- Write first draft of presentation slides.

Rachel

- Research, test, and select Phoronix benchmarks.
- Run benchmarks on personal devices.
- Create plots of experimental results.
- Write first draft of final paper.

Nathan

- Acquire patched and unpatched Linux kernels.
- Create live-boot Debian USB for benchmark tests.
- Run benchmarks on personal devices.
- Execute exploit for live demo during presentation.

V. Approach and Methodology

To assess the performance of various systems in patched and unpatched environments, we required a stable, consistent test platform for use on a variety of machines. We used a lightweight live-boot USB, containing both patched and unpatched kernels as a transferable testing platform. A robust open-source benchmarking suite was selected which supported a comprehensive evaluation of system performance.

TABLE I: Key differences between Spectre and Meltdown attacks.

	Meltdown	Spectre
Hardware	Intel processors	Any CPU with speculative execution
Vulnerability	Instructions executed out of order bypass Intel memory protections	Speculative execution modifies the cache, which is detected via a side-channel attack
Attack	Write an unreachable instruction to break memory isolation, and read data through a side-channel	Use branch misprediction to execute code that would not otherwise be executed to leak information
Mitigation	KAISER partially mitigates, RETPOLINE isolates kernel memory in Linux	Disable speculative execution, or use RETPOLINE patch

A. Experiment Setup

We chose the Debian 9 Linux distribution as our test OS platform. This included both an unpatched kernel (4.9.51-1 released October 2017 [8]), and a kernel patched against Spectre and Meltdown (4.9.82-1 released March 2018 [9]). Debian uses the Linux kernel and relies on the RETPOLINE patch to mitigate Spectre and Meltdown.

To allow for an identical test environment on all test machines, without requiring direct installation of an unpatched OS, both kernels were loaded onto a single live-boot USB. Benchmarks were selected which focused on system performance. For each machine in the test sample, the full set of benchmarks were run on the unpatched kernel, and then the patched kernel.

B. Phoronix Benchmarks

Due to the varying capabilities of our test machines, we selected the Phoronix Test Suite (PTS), a lightweight, cross-platform, comprehensive benchmarking platform, that offers hundreds of distinct tests [6] as our benchmarking platform, to avoid overly taxing the systems. We selected the five benchmarks given in Table II, due to there emphasis on performance. A lower metric score is better for all tests but RamSpeed. Final analysis normalizes and combines all five tests into a single overall performance metric.

TABLE II: PTS tests used for performance benchmarking.

Test	Focus	Metric Units
AOBench	real-world floating point performance	seconds
Cyclictest	system thread latency in response to stimulus	milliseconds
Primesieve	L1 and L2 cache performance	seconds
PyBench	average overall Python performance	milliseconds
RamSpeed	cache and memory performance	MB/s

C. Test Machines

To assess changes in performance to consumer-level computers as a result of Spectre and Meltdown patches, all systems tested were laptops. The sample set included eight Intel and three AMD CPU's manufactured across 10 years, but the set is limited to team member owned machines, given the potential security risks of running unpatched systems. System specifications are listed in Table III.

VI. Experimental Results and Evaluation

To remove machine-specific nuances from our results, we compared the speedup of patched kernels over unpatched kernels. A speedup less than 1 indicates degraded performance following the patch, while speedup greater than 1 indicates improvement. Results in this section rely on the speedup as a point of comparison across machines, benchmark tests, cache sizes, and processor type. Each individual test was run 3-5 times based on its time to completion, and the average time is reported.

A. Individual Test Results

Five different benchmarks were run on each of the 11 test machines. Then, the speedup of the patched kernel over the unpatched kernel for each benchmark was computed. Figure 1 plots the speedup of each benchmark on each machine.

Most benchmarks report a performance change within roughly %5 of the unpatched kernel, except the Cyclictest benchmark, which reported significant slowdown. This is likely because the benchmark returns millisecond times as small, whole integers, so there is less variance in the speedups than other benchmarks. This could also be Cyclictest reporting significant slowdown after applying the Spectre and Meltdown patch, since this test's goal is to repeatedly measure the difference between a thread's intended wake-up time and the time at which it actually wakes up to provide accurate system latency statistics.

Most benchmark results are not consistent across machines; on the same test some machines experience a performance degradation post-patch, while others see an improvement. Results for each machine are also inconsistent; some benchmarks yield speedup while others yield slowdown. The exception is the Cyclictest benchmark, which reports speedup less than or equal to 1 for all machines tested, but this may be because the test's precision is too low to detect small performance changes.

B. Overall Benchmark Results

We aggregated all five benchmarks into a single average speedup for each individual test machine, and report results with standard error in Figure 2.

All but one test machine, the AMD Athlon, have a final average speedup less than or equal to 1, indicating a performance degradation post-patch. The four low Intel speedups correspond to the low Cyclictest results, which skews the

TABLE III: Test Machine Specifications

CPU Type	Processor	Release Date	Base Frequency	L1 Cache	L2 Cache	Cores	Threads
AMD	Turion II M520	Q2 2010	2300 MHz	128 KB	1 MB	2	2
AMD	C-50	Q4 2010	1000 MHz	128 KB	256 KB	2	2
AMD	Athlon 7310	Q1 2015	2400 MHz	256 KB	2 MB	4	4
Intel	Pentium T3400	Q4 2008	2160 MHz	N/A	1 MB	2	2
Intel	i5-450M	Q2 2010	2400-2660 MHz	128 KB	512 KB	2	4
Intel	i5-2520M	Q1 2011	2500-3200 MHz	128 KB	512 KB	2	4
Intel	i3-2330M	Q2 2011	2200-2400 MHz	128 KB	512 KB	2	4
Intel	i7-3632QM	Q3 2012	2200-3200 MHz	256 KB	1 MB	4	8
Intel	i5-5200U	Q1 2015	2200-2700 MHz	128 KB	512 KB	2	4
Intel	i5-7500T	Q1 2017	3400-3800 MHz	256 KB	1 MB	4	4
Intel	I5-8250U	Q3 2017	1600-3400 MHz	256 KB	1 MB	4	8

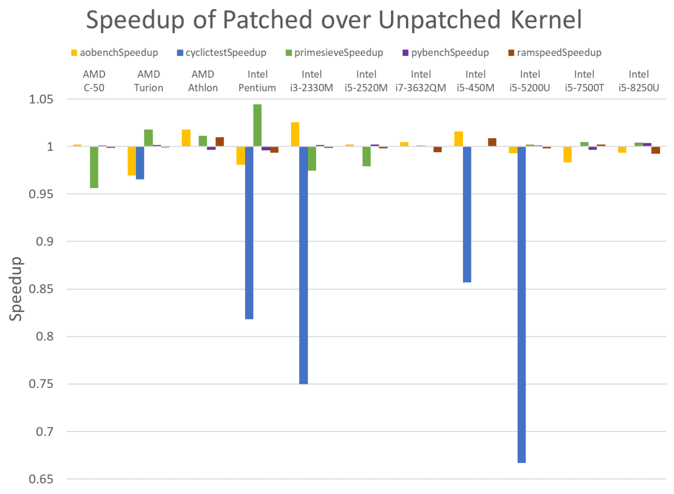

Fig. 1: Benchmark speedup results by test and machine.

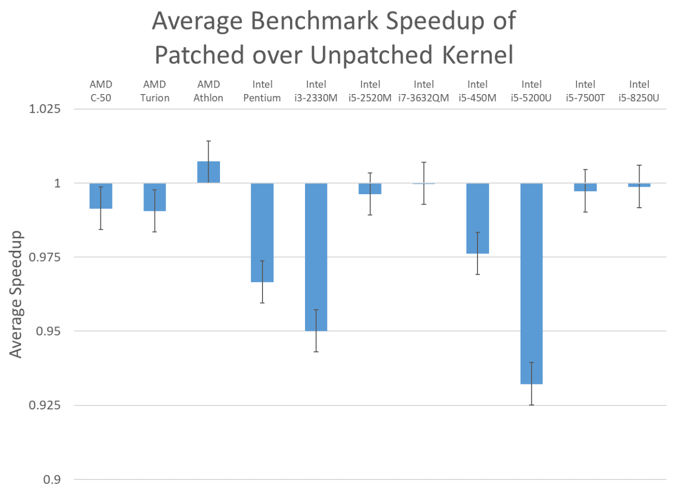

Fig. 2: Average benchmark speedup results for each machine.

average. Interestingly, all processors that experience significant slowdown during Cyclictest have 128 KB L1 and 256 KB L2 caches, and this seems to be a suboptimal design post-patch, perhaps because the small difference in size between the two caches leads to more traffic when operating on nonsequential data. L2 caches of unaffected processors are typically much larger than their L1 caches, so this may support faster nonsequential memory access.

C. Architecture Considerations

Given the wide variation in test machine specifications, we also investigated the impact of cache size on patch speedups. Figure 3 gives the average speedup of each benchmark for the range of L2 cache sizes tested. This corresponds to 256 KB caches, 512 KB caches, 1024 KB caches, and 2048 KB caches.

The fluctuation in Cyclictest's average is also influenced by the four Intel processors that experienced noticeable performance decrease, but if we remove this test as an outlier, there are no strong correlations between cache size and speedup.

D. Impact of Processor Type: Intel vs. AMD

Since Intel is affected by Spectre and Meltdown, while AMD is only vulnerable to Spectre, we wanted to investigate potential performance differences between each vendor's

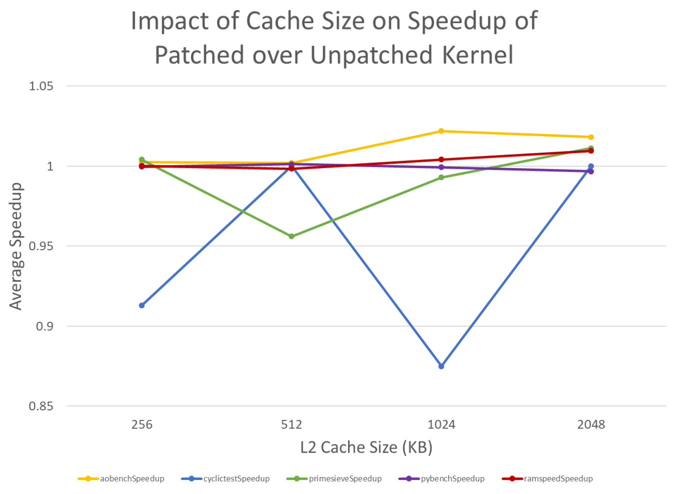

Fig. 3: Individual benchmark speedups as a function of L2 cache size.

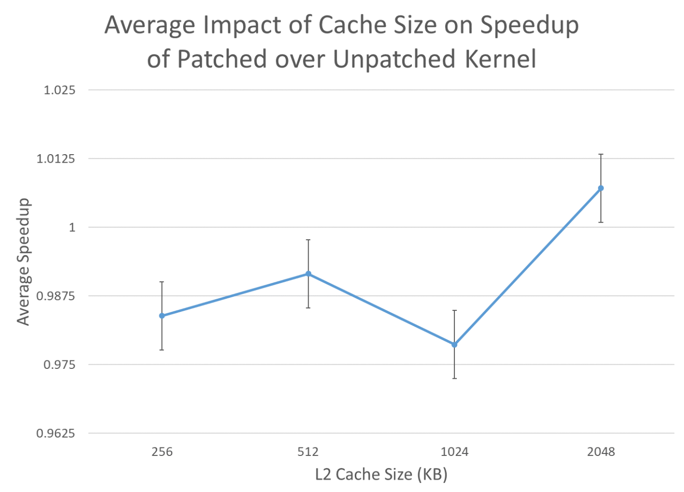

Fig. 4: Overall benchmark speedups as a function of L2 cache size.

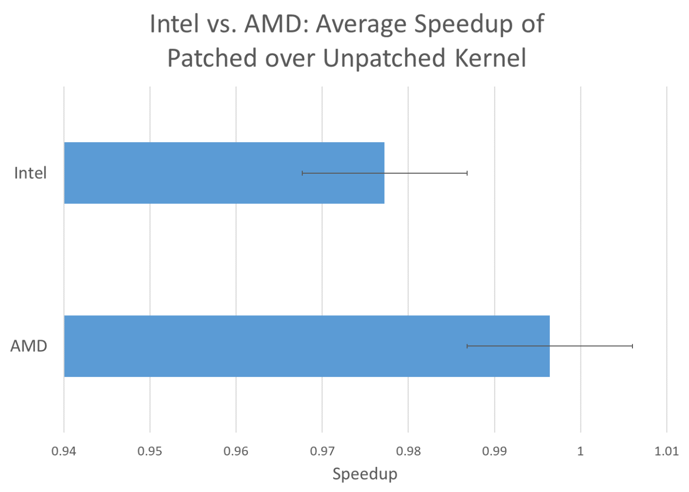

Fig. 5: Average speedup of Intel vs. AMD processors.

processors. For each processor type, we computed an overall average speedup of all test machines, reported in Figure 5.

The Intel processors' average is much lower than that of AMD processors, although the performance degradation for both is small (roughly 2.5%), even including the Cyclictest results in the averages. This suggests that Intel processors experience more performance degradation post-patch than AMD processors. It is interesting that AMD machines never have more threads than processors, and the Intel machines that experience the most performance degradation generally have more threads than processors. Perhaps the overhead of managing these extra threads is increased post-patch, and this decreases system performance.

VII. Conclusion

The Phoronix benchmarks used in our study indicate system performance of PCs is in general negligibly affected after patching for the Spectre and Meltdown bugs. However, the results of the Cyclictest benchmark indicate that the latency of certain services depend on responsive thread management or L2 cache size may be more severely affected. To optimize performance post-patch we suggest increasing the L2 cache relative to L1 and minimizing thread count to not greater than the core count. AMD processors seem to experience less performance degradation than comparable Intel hardware after post-patch, perhaps due to differences in thread management or cache organization.

VIII. Future Work

More testing with different benchmarks and machines is needed to validate our analysis. Also, we have primarily been testing the performance changes induced by the RETPOLINE patch, but further performance analysis after applying the KAISER patch would be useful for developing an efficient long-term patch. Also, running similar tests with different operating systems will yield insights into which key features are most affected by patches.

References

[1] Martin Giles. "At Least Three Billion Computer Chips Have the Spectre Security Hole". In: *MIT Technology Review* (2018).

[2] Daniel Gruss et al. "KASLR is dead: long live KASLR". In: *International Symposium on Engineering Secure Software and Systems*. Springer. 2017, pp. 161–176.

[3] Jann Horn. *Reading privileged memory with a side-channel*. Project Zero Blog. Jan. 2018. (Visited on 03/27/2018).

[4] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *arXiv preprint arXiv:1801.01203* (2018).

[5] Moritz Lipp et al. "Meltdown". In: *arXiv preprint arXiv:1801.01207* (2018).

[6] Phoronix Media. *Phoronix Test Suite Features*. 2018. (Visited on 04/01/2018).

[7] Metz and Perlroth. *Researchers Discover Major Flaws in the World's Computers*. New York Times. 2018.

[8] Debian Kernel Team. *linux 4.9.51-1 source package in Debian*. launchpad.net. Oct. 2017.

[9] Debian Kernel Team. *linux 4.9.82-1 source package in Debian*. launchpad.net. Mar. 2018.

[10] Liam Tung. *Google: Our brilliant Spectre fix dodges performance hit*. ZDNet. 2018. (Visited on 03/24/2018).

[11] Paul Turner. *Retpoline: a software construct for preventing branch-target-injection*. Google. 2018. (Visited on 04/09/2018).

[12] Yuval Yarom and Katrina Falkner. "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack." In: *USENIX Security Symposium*. 2014, pp. 719–732.

[13] Yinqian Zhang et al. "Cross-VM side channels and their use to extract private keys". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 305–316.