The Performance Cost of Software-based Security Mitigations

Lucy Bowen
lrbowen@calpoly.edu
California Polytechnic State University
San Luis Obispo, California, USA

Chris Lupo
clupo@calpoly.edu
California Polytechnic State University
San Luis Obispo, California, USA

ABSTRACT

Historically, performance has been the most important feature when optimizing computer hardware. Modern processors are so highly optimized that every cycle of computation time matters. However, this practice of optimizing for performance at all costs has been called into question by new microarchitectural attacks, e.g. Meltdown and Spectre. Microarchitectural attacks exploit the effects of microarchitectural components or optimizations in order to leak data to an attacker. These attacks have caused processor manufacturers to introduce performance impacting mitigations in both software and silicon.

To investigate the performance impact of the various mitigations, a test suite of forty-seven different tests was created. This suite was run on a series of virtual machines that tested both Ubuntu 16 and Ubuntu 18. These tests investigated the performance change across version updates and the performance impact of CPU core number vs. default microarchitectural mitigations. The testing proved that the performance impact of the microarchitectural mitigations is non-trivial, as the percent difference in performance can be as high as 200%.

ACM Reference Format:
Lucy Bowen and Chris Lupo. 2020. The Performance Cost of Software-based Security Mitigations. In *Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering (ICPE '20), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3358960.3379139

1 INTRODUCTION

Performance has always been an important factor in computer design. Over time the gap in performance between the memory and CPU has widened. In order to take advantage of the CPU speed, various microarchitectural optimizations were created. Unfortunately, recent research shows these optimizations can leak privileged information if exploited by savvy attackers [2]. Processor manufacturers and operating system programmers have created software based solutions to these exploits, but some exploits can only be solved with hardware changes [15]. Several software solutions are harmful to performance, but are necessary to secure the system until new hardware is designed.

ICPE '20, April 20–24, 2020, Edmonton, AB, Canada

ACM ISBN 978-1-4503-6991-6/20/04…$15.00
https://doi.org/10.1145/3358960.3379139

It is important to quantify how much performance has been lost for the sake of security. Users are allowed to disable these security mitigations, even if it is to their own detriment. Those with performance-intensive tasks: simulation, image creation / rendering, video encoding, etc., may desire to regain the previous performance. A user could choose to take the risk with removing the mitigations, or if they want to keep some security, instead use a dedicated machine for those tasks and ensure that it contains no sensitive information.

This paper is a comprehensive analysis of the performance impact from software-based security mitigations on different patch levels of Ubuntu 16 and Ubuntu 18, and the affect of allocating differing numbers of cores to the latest patch. By using a test-suite consisting of forty-six different tests, a quantitative summary of the performance impacts was created. Analysis of these tests shows that the mitigations had a quantifiable performance affect, with some being negligible but others by as much as three orders of magnitude difference in performance.

Section 2 provides background into the various microarchitectural optimizations and the attacks on them. The design and implementation of the experiments are covered in Sections 3 and 4, with an explanation of the tests in the test-suite in the former and an explanation of the software mitigations in the latter. Section 5 presents an analysis of both the individual test results and of the results when compared as a group. Future work and conclusions are presented in Sections 6 and 7, respectively.

2 BACKGROUND

Many of the microarchitectural optimizations that exist within modern processors were first created decades ago. These same optimizations that are critical to modern design and performance, have recently been proven to have significant, difficult to fix, security flaws. A dangerous cache-timing attack has been adapted to attack how these optimizations work. By doing this, attackers can access sensitive data without detection. This section covers how cache timing attacks work, and what the recent high profile microarchitectural attacks are.

2.1 Cache Attacks

Historically in computer security, solutions which used powerful encryption/decryption algorithms with cryptographic keys were considered secure. If a cryptographic algorithm is given a large enough key, brute force attacks become computationally infeasible. Attackers can instead target the physical implementation in hardware in order to take advantage of some physical information leaked by a cryptographic device. This is a side channel attack,

where the information is gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself [2].

A timing attack is an attack based on measuring how much time various computations take to perform. Cache attacks are a subset of timing attacks, which focus on exploiting the timing differences caused by the lower latency of CPU caches when compared to off-chip memory [2]. While other attacks exist, the three most well known are the Prime + Probe attack [21], the Evict + Time attack [21], and the Flush + Reload attack [42], with many of the other existing cache attacks being derivations or combinations of them.

2.2 Microarchitectural Attacks

Microarchitectural attacks are attacks which exploit the effects of microarchitectural components or optimizations. Since January 2018 the field has been in the news due to high profile, difficult-to-fix attacks, specifically those based on the Meltdown and Spectre attacks. Meltdown- and Spectre-type attacks are both speculative execution based attacks; the former attacks obtain data from instructions following a fault, while the latter attack prediction units [2]. Microarchitectural Data Sampling is a new attack type that instead targets small frequently overwritten buffers used in speculative execution. The microarchitectural attacks Nemesis [39], TLBleed [8], and Spoiler [11] use concepts from the above attack types but are not, themselves, members of them.

2.2.1 Meltdown-Type Attacks. Meltdown, once a single attack, has become the name of an overarching attack type where information is gained from transient executions following a faulting instruction. This attack type exploits out-of-order execution by extracting data from faulting instructions that are forwarded ahead in the pipeline. Meltdown attacks use transient execution to "melt down" architectural isolation barriers by computing on unauthorized results of faulting instructions, thus transiently bypassing hardware-enforced security policies in order to leak data that was architecturally inaccessible for the application. They reflect a failure of the CPU to respect hardware-level protection boundaries for transient instructions [4].

The original Meltdown [17], Foreshadow [38], Foreshadow-NG [41], Rogue System Register Read [10], and Lazy FP State Restore [36] are examples of Meltdown-type of attacks.

2.2.2 Spectre Type Attacks. Like Meltdown, the original Spectre attack, made up of Variants 1 and 2, has grown to encompass its own attack type. Spectre-type attacks exploit transient execution following control or dataflow misprediction, and rely on dedicated control or dataflow prediction machinery. In Spectre-type attacks, transient instructions only compute on data which the application is also allowed to access architecturally, which allows the attacks to transiently bypass software-defined security policies in order to access secrets out of the program's intended code/data paths [4].

Variant One [15], Variant Two [15], SGXPectre [5], BranchScope [7], Speculative Store Bypass [10], Bounds Check Bypass Store [14], Speculative Store Read-Only Overwrite [14], SpectreRSB and ret2spec [16, 18], NetSpectre [35], and SplitSpectre [19] are examples of Spectre-type attacks.

2.2.3 Microarchitectural Data Sampling. Microarchitectural Data Sampling (MDS) is a class of CPU vulnerability that does not rely on assumptions about memory layout, or depend on the processor cache state. This separates these attacks from Meltdown- and Spectre-like attacks. However, several MDS attacks are also counted as Meltdown-type attacks due to their use of fault or exception exploitation. Because of the lack of assumptions, MDS attacks are difficult to mitigate, though the structures involved are relatively small and are overwritten more frequently.

Fallout [20], Rogue In-Flight Data Load [40], Store-To-Leak Forwarding [34], and ZombieLoad [37] are examples of MDS attacks.

More detail on each of the attacks listed in this section can be found in [3].

3 DESIGN

In order to get a quantifiable impact of the software mitigations to the various microarchitectural attacks, the performance of many programs should be looked at. These programs should cover a large spectrum of possible users, and not simply be a series of micro-benchmarks, as those are often optimized by processor manufacturers. By using a wide variety of programs, different implementations and use cases can be investigated. Testing on virtual machines allows for additional control conditions to be added and tested without the need to perform the tests on additional, expensive, hardware. Using a known test-bench software suite allows for future comparisons to the same dataset.

3.1 Virtual Machines

Running the experiment on virtual machines allows for a comparison of both core count and vulnerability mitigation. The virtual machines in these experiments will run different Ubuntu versions. Additionally, by comparing against all of the older versions of a release, the performance loss or gain from Linux development can be seen. It should be noted that all versions of Ubuntu 18 have been released since the discovery of Meltdown and Spectre, while version 16.04.04 on-wards have been influenced by their discovery.

3.2 Phoronix Test Suite

The Phoronix Test Suite is a free, open-source benchmark software that supports Linux, Windows, Apple OS X, GNU Hurd, Solaris and BSD Operating Systems. It has access to more than 450 test profiles and over 100 test suites via OpenBenchmarking.org, and comes with built-in statistical result reporting. Tests and Test Suites inside Phoronix are built with eXtensible Markup Language.

3.3 Benchmarks

The following benchmarks were chosen on the basis of time, environment, and function. All of the chosen benchmarks can run on both Windows and Linux, so that in the future the results can be compared with Windows results.

3.3.1 PostgreSQL. pgbench is a simple program for running benchmark tests on PostgreSQL. It runs the same sequence of SQL commands over and over, possibly in multiple concurrent database sessions, and then calculates the average transaction rate (Transactions Per Second). We use the pre-created options provided by

Phoronix so these results can be easily compared against both future tests and other tester's work.

3.3.2 DaCapo Benchmarks. The DaCapo benchmarks were created in order to see the complex interactions between the architecture, compiler, virtual machine, memory management, and application. The DaCapo benchmarks used in this paper are: Eclipse, H2, Jython, and DayTrader [25].

3.3.3 SciMark. SciMark was developed by NIST and is widely used by the industry as a floating point benchmark. It is a benchmark for scientific and numerical computing. It consists of several subtests: Dense Lower-Upper Matrix Factorization, Fast Fourier Transform, Jacobi Successive Over-Relaxation, Monte Carlo, and Sparse Matrix Multiply. The SciMark benchmarks tested in this paper are Dense Lower-Upper Matrix Factorization, Fast Fourier Transform, Jacobi Successive Over-Relaxation, Monte Carlo, Sparse Matrix Multiply, and a Composite of all of these [22].

3.3.4 Encoding. Encoding is the process of putting a sequence of bytes into a specialized format for efficient transmission or storage. The encoding benchmarks used in this paper are the LAME MP3 Encoder [33], and x264 [32].

3.3.5 Ray Tracing. Ray tracing is a technique for rendering three-dimensional graphics with very complex light interactions, e.g. reflection, refraction, scattering, and dispersion phenomena. The ray tracing bencmarks used in this paper are C-Ray [24], and the Sunflow Rendering System [30].

3.3.6 Compression. The compression benchmarks used in this paper are 7-zip [1], and Gzip [26].

3.3.7 Miscellaneous. These are tests that, while useful and informative, did not fit well into any of the previous categories.

Bork. Bork is a small, cross-platform file encryption utility. It is written in Java and uses a stream cipher with RC4, and is able to obfuscate filenames with SHA-1 hashing. This test measures the amount of time it takes to encrypt a sample file [23].

MAFFT. MAFFT is a free program used to create multiple sequence alignments of amino acid or nucleotide sequences. This test performs an alignment of 100 pyruvate decarboxylase sequences, with pairwise alignments computed with the Smith-Waterman algorithm and 20000 iterative refinement cycles performed [31].

R Benchmark. The R Benchmark in the Phoronix Test Suite downloads several R benchmarks; however, only R-benchmark-25, was used. The Phoronix Test Suite version is customized for the rbench driver [27]. R-benchmark-25 is a series of R benchmarks with three categories: matrix calculation, matrix functions, and programmation [9]. .

SQLite. SQLite is a software library that provides a relational database management system. This test profile measures the time to perform a pre-defined number of insertions on an indexed database [29].

Unigine - Sanctuary. Unigine is a proprietary cross-platform game engine, developed by Russian software company Unigine Corp. This test includes both a windowed and a fullscreen mode with a resolution of 800 x 600. [28].

Table 1: Virtual Machine Information

Operating System	Memory	Video Memory	Processors	Hard Drive Size
16.04.01	8192 MB	128 MB	2 Cores	25 GB
16.04.02	8192 MB	128 MB	2 Cores	25 GB
16.04.03	8192 MB	128 MB	2 Cores	25 GB
16.04.04	8192 MB	128 MB	2 Cores	25 GB
16.04.05	8192 MB	128 MB	2 Cores	25 GB
16.04.06	8192 MB	128 MB	Variable	50 GB
18.04.01	8192 MB	128 MB	2 Cores	35 GB
18.04.02	8192 MB	128 MB	Variable	150 GB

4 IMPLEMENTATION

To gather data for the analysis, all of the benchmarks from the previous section were run at least once with a minimum of three data points per run. Most benchmarks were run an additional time after the initial round of testing, in order to increase the number of data points per test. However, because of how long the pgbench benchmarks took (about 17 hours for Ubuntu 16 and about 28 hours for Ubuntu 18) a full second run of that benchmark was not completed. Tests were completed for the different versions of Ubuntu 16 and 18, allowing for a comparison of 32-bit to 64-bit. Additionally, comparing across the different version numbers demonstrates what performance loss is normal from a version update vs. the mitigations to speculative execution attacks. Using the latest versions of Ubuntu 16, 16.04.06, and Ubuntu 18, 18.04.02, the affect of having or not having the default mitigations was across several core counts. This allowed for a view into the performance affect of the mitigations on each core level and the affect of having more or less cores for each benchmark.

The tests were completed on virtual machines with an Intel Haswell processor. The mitigations were only tested in their default and "off" states, and mitigations that required a kernel recompile were ignored.

4.1 Virtual Machines

Oracle VM Virtual Box, version 6.0.4 r128413 (Qt5.6.2), was used to create and manage the virtual machines. Information about these machines can be seen in Table 1. Ubuntu 18 required a larger amount of hard drive space than Ubuntu 16 due to it being a default 64-bit operating system, unlike Ubuntu 16 which is a default 32-bit. Additionally, because of its 64-bit status, gcc-multilib had to be installed for some tests, meaning that some tests could natively use 64-bit while others could not. Ubuntu 18.04.02 required a large virtual drive; tests with smaller hard drive sizes (25GB, 35GB, 50GB, 75GB and 100GB) failed pgbench runs due to a size error. Ubuntu 16.04.06 had a similar issue, but the error resolved with a 50GB hard drive rather than 18.04.02's 150GB.

4.2 Host Machine

The machine used to host these tests is a Windows 10 Pro Desktop computer. It has 16GB of RAM and all of the virtual machines were run off of a 2TB hard drive. The CPU used in these tests is an Intel Haswell i5-4690K [12]. It has a clock speed of 3.50GHz and 4 cores.

4.3 Mitigations

The mitigations in this section are those that were visible when the testing was taking place. These are displayed in the `/sys/devices/system/cpu/vulnerabilities/` directory. Table 2 shows their default state and history for the two different operating systems. Other vulnerabilities and attacks either were not addressed yet by Linux or were fixed entirely by microcode updates and therefore were not visible as a specific mitigation. Testing was between the specific operating system's default state and a state where all mitigations that could be disabled without a kernel recompile, were. It is important to note that past versions of Ubuntu 16 and 18 also have some of the mitigations enabled.

4.3.1 GRUB. GRUB is a boot loader for Linux that allows a user to select a specific kernel configuration. The options for GRUB are found in `/etc/default/grub`. `GRUB_CMDLINE_LINUX_DEFAULT` is used to change which mitigations are enabled. When this file is changed, the system can be commanded to update grub and reboot, and the resulting machine will have the default mitigations disabled.

4.3.2 Meltdown. The Meltdown vulnerability (CVE-2017-5754) can be completely mitigated in software [17]. This mitigation is kernel page-table isolation (KPTI), previously called KAISER.

KPTI can partially be disabled with the *nopti* kernel boot option. Ubuntu 16.04.06 does not have KPTI enabled by default, while Ubuntu 18.04.02 does.

4.3.3 Foreshadow. Intel has classified all of the Foreshadow vulnerabilities (CVE-2018-3615, CVE-2018-3620, CVE-2018-3646) as Level-One Terminal Fault, L1TF, and they can be partially mitigated in software. The most basic mitigation is Page Table Entry (PTE) Inversion. By inverting all of the bits in a PTE when it is not marked as present the PTE will point to a nonexistent region of memory [6]. This is enabled by default and cannot be turned off without a kernel recompile.

The two additional, non-default, mitigations are only if Kernel Virtual Machine (KVM) is enabled. The first turns off simultaneous multithreading (SMT), or hyper-threading on Intel machines. Disabling SMT can have a significant performance impact, and therefore must be weighted against the impact of other mitigation solutions like confining guests to dedicated cores. The other is to have the hypervisor flush the L1 Data Cache (L1D) before entering the guest. Flushing the L1D evicts both guest data and any data that should not be accessed by malicious guests. However, this also has a performance impact as the processor has to bring the flushed guest data back into the L1D [13]. This paper uses Ubuntu as the Guest machine, and not the host, so the virtual machines do not have KVM enabled. Thus, these two mitigations can be ignored.

4.3.4 Spectre Variant 1. Spectre Variant 1 (CVE-2017-5753), can be mitigated in software only by patching code sequences found to be vulnerable. To patch a vulnerable section, a barrier instruction, LFENCE, is inserted in order to stop speculation. Alternatively, all instructions can be serialized in order to stop younger instructions from executing, even speculatively, before older instructions have retired, but LFENCE is a better performance solution. An LFENCE instruction inserted after a bounds check will prevent younger operations from executing before the bound check retires. However, it must be used carefully; if used too often, performance is compromised. Developers are using static analysis to find these vulnerable sections, but if even one section in a codebase is skipped the entire codebase is still vulnerable [10]. This mitigation is enabled by default on operating systems that have it, because it is compiled into the kernel. If a user decompiles and recompiles the kernel it can be turned off, but for this paper it was not.

4.3.5 Spectre Variant 2. Google developed the Full Generic Retpoline mitigation for Spectre Variant 2 (CVE 2017-5715). A retpoline is a "return trampoline", where the software replaces indirect near jump and call instructions with a code sequence that includes pushing the target of the branch in question onto the stack and then executing a return instruction to jump to that location, as return instructions can be protected using this method [10].

4.3.6 Speculative Store Bypass. The mitigation for Speculative Store Bypass (CVE 2018-3639) is not on by default on either 16.04.06 or 18.04.02. This is because Intel recommends enabling this mitigation only for managed runtimes or other situations that use language-based security to guard against attacks within an address space. If this mitigation is enabled the system sets the Speculative Store Bypass Disable bit in order to prevent loads from executing before all older store addresses are known. Software that does not use language-based security should instead carefully insert LFENCE instructions, insert additional register dependencies between vulnerable loads and stores, or isolate secrets into a separate address space from code that is relying on language-based security [10]. For this paper, this mitigation was ignored because it was not on by default.

5 RESULTS

Most benchmarks behaved as expected, with test runs with more cores outperforming those with less, and tests with mitigations disabled outperforming those with them enabled. While some tests demonstrate outliers these are all within testing runs that have otherwise consistent data.

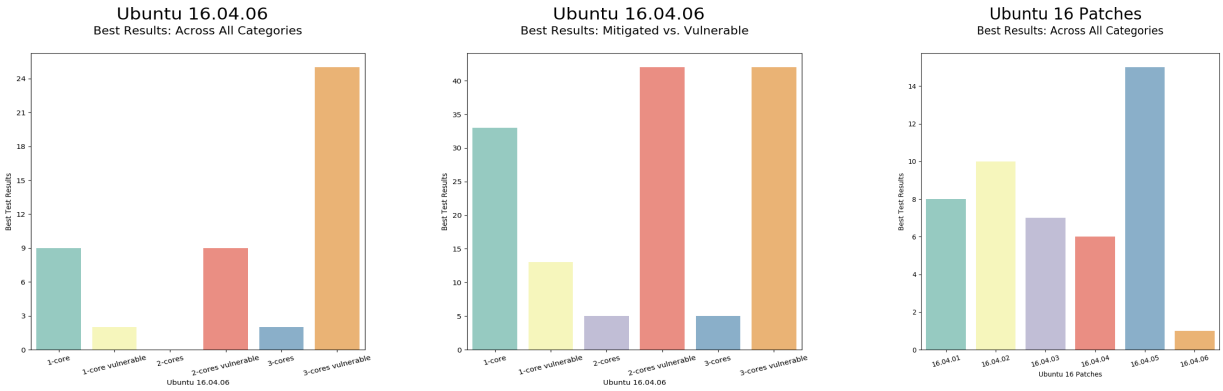
5.1 Average Performance

Looking at the average performance of the tests prevents outliers from skewing the data. Table 3 compares the average percent difference for vulnerable versus mitigated tests within the same core count, showing that the average variance between tests was low. Additionally, tests were compared across both patch and core number to see which had the best results, with best being determined by each benchmark because for some benchmarks higher numbers are better and others, lower.

Figure 1 compares the results of each benchmark for all of the possible core counts, mitigation impact, and patch levels for Ubuntu 16. Figure 2 shows similar results for Ubuntu 18. The data show that for Ubuntu 16, 3 cores vulnerable is almost always the best, while

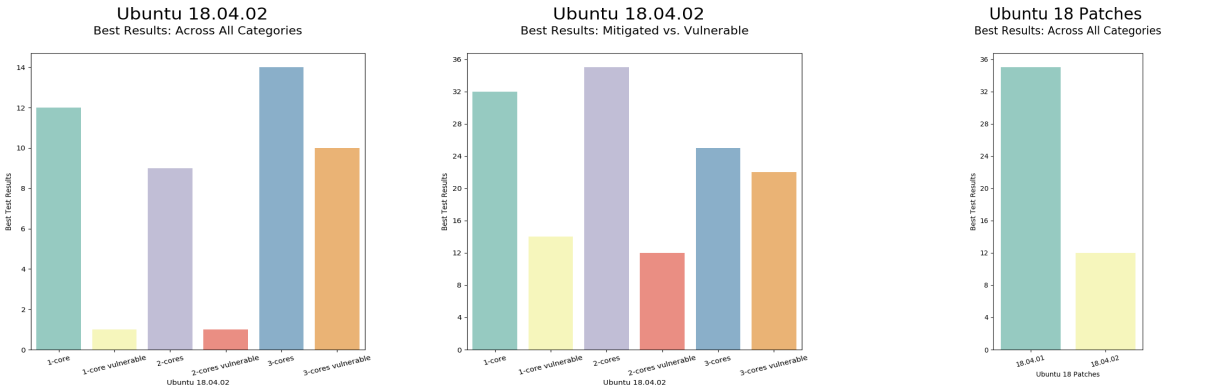
Table 2: Vulnerabilities and Mitigations

Vulnerability	16.04.04	16.04.05	16.04.06	18.04.01	18.04.02
Meltdown	Default Off	Default Off	Default Off	Default On	Default On
Foreshadow	Not Available	Always On	Always On	Always On	Always On
Spectre v1	Always On	Always On	Always On	Always On	Always On
Spectre v2	Default On	Default On	Default On	Default On	Default On
Speculative Store Bypass	Not Available	Default Off	Default Off	Default Off	Default Off



(a) Best Results: All Categories **(b) Best Results: Mitigated vs. Vulnerable** **(c) Best Results: All Patch Levels**

Figure 1: Ubuntu 16 Results



(a) Best Results: All Categories **(b) Best Results: Mitigated vs. Vulnerable** **(c) Best Results: All Patch Levels**

Figure 2: Ubuntu 18 Results

Table 3: Average % Difference Per Core Count

Core Number	16.04.02 Core	18.04.02 Core
1 core	3.40%	2.54%
2 cores	3.07%	4.93%
3 cores	3.47%	3.68%

for Ubuntu 18 the results are more spread out. The Ubuntu 18 results may be affected by the patch to the host machine, and on the benchmarks where this looks possible it is called out. It is interesting to note for Ubuntu 16, when there is more than one core the vulnerable version is almost always better. This is likely because many of the mitigations involve both multiple physical and logical cores; enabling those mitigations makes those tests slower. Thus, the mitigations don't significantly change performance when tested on only one core. The Ubuntu 18 results are spread out across all categories. Notably no vulnerable category outperforms its default counterpart, potentially due to the patch to the host machine.

In Figure 3 the result of all tests compared against each other can be seen. Ubuntu 18 is shown to be superior, and interestingly 18.04.01 can almost compete with 18.04.02 even when it is at one less core. This emphasizes the impact of the 18.04.02 micro-op changes.

The number of tests that do better without mitigations is nearly equal to the number that do better with mitigations. This indicates

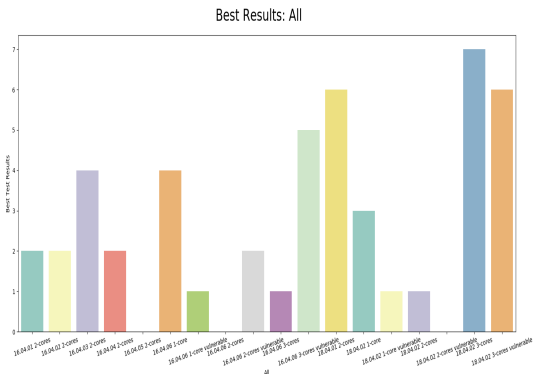


Figure 3: Best Results: All

that while many tests are affected by the software mitigations, it is not the overwhelming majority.

5.2 PostgreSQL

The pgbench Benchmarks have the most diversity when looking at the impact of both patch and core differences. They are almost universally better with a higher core number and with mitigations disabled. Additionally, these benchmarks are very much impacted by patches to the host machine. Because of this disparity in performance, data from before and after the patch has been separated.

Table 4: Buffer Tests Benchmark % Difference of Average

Benchmark	16 Patch	16.04.02 Core	18 Patch	18.04.02 Core
Heavy - RO	8.02	97.96	19.67	104.96
Heavy - RW	43.98	59.12	5.32	74.58
Normal - RO	12.55	98.41	18.55	106.23
Normal - RW	43.71	65.77	2.44	73.66
1 Thread - RO	27.89	57.21	7.09	61.08
1 Thread - RW	17.25	10.52	11.88	3.34

5.2.1 Buffer Test. Table 4 shows large discrepancies between the best and the worst for each category.

Table 5: RAM Tests Benchmark % Difference of Average

Benchmark	16 Patch	16.04.02 Core	18 Patch	18.04.02 Core
Heavy - RO	7.09	9.94	199.77	14.31
Heavy - RW	3.24	18.26	199.71	21.14
Normal - RO	8.81	41.19	199.86	45.58
Normal - RW	5.09	24.70	199.88	27.68
1 Thread - RO	38.03	11.51	199.97	10.54
1 Thread - RW	17.99	6.86	199.98	20.16

5.2.2 Mostly RAM. Although Table 5 would indicate that the percent difference between core levels is large, these tests share the relationship of more cores performing markedly better.

5.2.3 Mostly RAM - Host Patch. The Mostly RAM test is the most dramatically affected by the application of the patch to the host system. Bizarrely, according to the documentation of the patch, this should have affected only VIA-based systems, which the host machine is not. Further research into whether Meltdown and Spectre mitigation patches for other chip types are inadvertently impacting unrelated machines is needed.

Table 6: DaCapo Benchmark % Difference of Average

Benchmark	16 Patch	16.04.02 Core	18 Patch	18.04.02 Core
Eclipse	7.18	61.24	1.90	63.02
H2	32.53	34.19	3.13	19.37
Jython	4.42	72.65	2.23	68.41
Tradebeans	4.52	76.93	0.18	76.16
Tradesoap	12.75	86.21	2.35	87.69

5.3 DaCapo

Table 6 shows that all of the DaCapo benchmarks are affected by core count, as expected. Some benchmarks, Eclipse, H2, and Jython, do poorly with one core but the performance with two and three cores is similar. The other DaCapo benchmarks, Tradebeans and Tradesoap, are more sensitive to having two than three cores. The table additionally shows that within patch levels there is little percent difference. H2 has a significant variance when looking at Ubuntu 16 patches.

Table 7: SciMark: ANSI C Benchmark % Difference of Avg

Benchmark	16 Patch	16.04.02 Core	18 Patch	18.04.02 Core
Dense LU	1.84	2.39	1.65	3.36
FFT	6.75	1.97	1.08	2.30
Jacobi SOR	1.29	2.36	1.00	2.53
Monte Carlo	2.18	3.29	1.19	3.94
Sparse MM	1.22	1.93	1.77	3.96
Composite	1.22	1.67	1.37	3.07

5.4 SciMark

There is little difference obserserved between either patch levels or core number for either ANSI C or Java SciMark. This is likely because these are fairly small math benchmarks that both processor and operating system creators use to optimize their product before it is released. It is still useful to investigate them to ensure nothing has gone catastrophically wrong, considering how many scientific simulations use thousands of these operations, but their lack of a meaningful winning category for any given SciMark test is both unsurprising and comforting.

5.5 Encoding

Table 8 shows patch level having a negligible impact. This benchmark is another interesting potential case where the host patch may have affected the Ubuntu 18 vulnerable versus default tests. While this behaves as expected, the audio encoding benchmark, LAME MP3, does not. Table 8 shows that LAME MP3 is not affected by either core number or patch level.

Table 8: Encoding % Difference of Average

Benchmark	16 Patch	16.04.02 Core	18 Patch	18.04.02 Core
LAME MP3	2.71	2.89	1.28	1.93
x264	2.14	81.67	1.61	84.35

5.6 Ray Tracing

Table 9: Ray Tracing % Difference of Average

Benchmark	16 Patch	16.04.02 Core	18 Patch	18.04.02 Core
C-Ray	1.20	98.44	0.70	97.54
Sunflow	7.08	35.32	4.57	35.28

The ray tracing benchmarks both perform as expected. There is little variance in patch levels according to Table 9.

Table 10: Compression % Difference of Average

Benchmark	16 Patch	16.04.02 Core	18 Patch	18.04.02 Core
7-zip	3.96	64.36	0.09	66.61
Gzip	1.14	5.20	3.90	6.33

5.7 Compression

The compression tests perform as expected. It is interesting to see that for both tests Ubuntu 18.04.01 was the best. The 7-Zip benchmark has a slightly unusual bimodal distribution and seems to be able to use two cores at most, while the Gzip tests do not display the same distribution. 7-Zip's bimodal distribution is likely caused by caching; the test is compressing a file three times, the first time will always be the worst and the next test runs will better because the data will be in one of the cache levels. If this test was improved it would compress the file multiple times each run and average the time, or compress a file that was much larger than the last level cache. Table 10 shows that there is little variance with different patch levels for either test. Lastly, the gzip Ubuntu 18 tests have a better average speed than their Ubuntu 16 counterparts, meaning that the `tar` function in Ubuntu 18 is better optimized.

5.8 Miscellaneous

Table 11: Miscellaneous % Difference of Average

Benchmark	16 Patch	16.04.02 Core	18 Patch	18.04.02 Core
Bork	38.62	39.53	1.43	57.32
MAFFT	8.42	87.75	0.10	84.84
R	5.28	4.45	0.38	3.00
SQLite	41.23	33.77	3.33	25.56
Uengine - Fullscreen	3.99	94.56	0.75	93.64
Uengine - Windowed	4.21	94.87	1.61	94.03

5.8.1 Bork. The Bork test is unusual as an outlier in this regard, but this is likely caused by its larger variance when compared to the other tests in this section. When comparing the percent difference in the testing runs, seen in Table 11, the Ubuntu 18 Patches have smallest percent difference, confirming how similar the data is between the two runs. In contrast the different core levels of 18 have the largest variance, but this is likely due the extreme anomalous behavior shown in 2-cores vulnerable.

5.8.2 MAFFT. The tests for MAAFT behaved as expected. Table 11 shows a small variance between patches.

5.8.3 R. Table 11 shows that the percent difference between runs for the R benchmark is small, so it seems affected by the mitigations, but without more data from other sources it is hard to know if it is just a coincidence.

5.8.4 SQLite. The SQLite tests are unusual in that they do not perform better for having more cores. This is likely because, as a Lite model, it is designed without the heavy threading that would take advantage of the higher core number. Additionally, these tests have a high degree of variance, Table 11, which is likely caused by some of the tests having extreme outliers.

5.8.5 Unigine. Both tests for Unigine behaved as expected, with the number of cores being the most important aspect.

6 FUTURE WORK

Recording the total impact from the mitigations to stop Meltdown and Spectre is a large task. The number of potential combinations is enormous, and the computation time for the tests in this paper is non-trivial. Considering how quickly new speculative execution attacks are being discovered, it is very likely that a new attack will be discovered and mitigated before the testing is complete.

The benchmarks demonstrated in this paper take 20 hours to complete, for a single run. Total experiment time is a function of test time, hardware, mitigation, virtual machine software, host os, and guest os. The experiment time for a reasonable set of combinations quickly grows to thousands of years of computation time.

7 CONCLUSION

The result of the tests were not surprising. As expected, almost all benchmarks benefited greatly from an increased core count. These benchmarks were either the newer or more complex benchmarks whose developers optimized for the additional performance the increase in core number can give. Simpler benchmarks were affected less by either core count or having mitigations turned off. Surprisingly, even though Ubuntu 16 did not have the Meltdown mitigation, kpti, enabled by default, it did not outperform Ubuntu 18. This shows that being 64-bit native is more of a performance impact than the kpti mitigation. The biggest surprise was the impact of the accidental patch to the host, which according to Windows, should have only affected VIA-based systems. Although many benchmarks showed affect, the degree of change was mostly small.

These tests showed that currently, the number of benchmarks that performed better without mitigations was almost equal to the number that performed better with the mitigations enabled. Thus, while many benchmarks are affected by the software mitigations, it

is not the overwhelming majority. Therefore, users seeking performance gain must research their individual usecase, as there does not seem to be a blanket gain by disabling mitigations.

Some microarchitectural exploits can only be fixed in hardware. Processor manufacturers are attempting to create new designs that are not vulnerable to Meltdown- and Spectre-type exploits However, even as these designs are created new exploits are discovered that need to be protected against, causing additional temporary software mitigations. Consumers seeking top performance will need to continue to follow the news about these exploits and mitigations.

The problem of performance versus security will continue as long as computers exist, if not longer. The flaws exposed by Meltdown and Spectre are causing security researchers to tear apart hardware diagrams and undocumented optimizations. Eventually they will be fixed, and new optimizations will be added that will in turn be targeted, starting the cycle anew.

REFERENCES

[1] 7-Zip. 2019. 7-Zip Website. https://www.7-zip.org

[2] Swarup Bhunia and Mark Tehranipoor. 2019. Chapter 16 - System Level Attacks & Countermeasures. In *Hardware Security*, Swarup Bhunia and Mark Tehranipoor (Eds.). Morgan Kaufmann, 419 – 448. https://doi.org/10.1016/B978-0-12-812477-2.00021-6

[3] Lucy Bowen. 2019. *The Cost of Security.* Master's thesis. California Polytechnic State University.

[4] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2018. A Systematic Evaluation of Transient Execution Attacks and Defenses. *CoRR* abs/1811.05441 (2018). arXiv:1811.05441 http://arxiv.org/abs/1811.05441

[5] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2018. SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution. (02 2018).

[6] Jonathan Corbet. 2018. Meltdown strikes back: the L1 terminal fault vulnerability. (2018). https://lwn.net/Articles/762570/

[7] Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18)*. ACM, New York, NY, USA, 693–707. https://doi.org/10.1145/3173162.3173204

[8] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2018. Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with {TLB} Attacks. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 955–972.

[9] Philippe Grosjean and Stephan Steinhaus. 2008. R Benchmarks. https://mac.r-project.org/benchmarks/

[10] Intel. 2018. Intel Analysis of Speculative Execution Side Channels. http://kib.kiev.ua/x86docs/SDMs/336983-004.pdf

[11] Saad Islam, Ahmad Moghimi, Ida Bruhns, Moritz Krebbel, Berk Gulmezoglu, Thomas Eisenbarth, and Berk Sunar. 2019. SPOILER: Speculative Load Hazards Boost Rowhammer and Cache Attacks. *CoRR* abs/1903.00446 (2019).

[12] Tarush Jain and Tanmay Agrawal. [n.d.]. The haswell microarchitecture-4th generation processor. ([n. d.]).

[13] The kernel development community. [n.d.]. L1TF - L1 Terminal Fault. https://www.kernel.org/doc/html/latest/admin-guide/l1tf.html

[14] Vladimir Kiriansky and Carl Waldspurger. 2018. Speculative Buffer Overflows: Attacks and Defenses. *CoRR* abs/1807.03757 (2018).

[15] 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. 2019. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*.

[16] 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)*.

[17] 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. 2018. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*.

[18] 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.

[19] Andrea Mambretti, Matthias Neugschwandtner, Alessandro Sorniotti, Engin Kirda, William Robertson, and Anil Kurmus. 2018. Let's Not Speculate: Discovering and Analyzing Speculative Execution Attacks. *IBM Research* (2018).

[20] Marina Minkin, Daniel Moghimi, Moritz Lipp, Michael Schwarz, Jo Van Bulck, Daniel Genkin, Daniel Gruss, Berk Sunar, Frank Piessens, and Yuval Yarom. 2019. Fallout: Reading Kernel Writes From User Space. (2019).

[21] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache Attacks and Countermeasures: The Case of AES. In *Topics in Cryptology – CT-RSA 2006*, David Pointcheval (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–20.

[22] Roldan Pozo and Bruce R Miller. [n.d.]. SciMark 2.0. https://math.nist.gov/scimark2/about.html

[23] Open Benchmarking. 2016. Bork File Encrypter. https://openbenchmarking.org/test/pts/bork

[24] Open Benchmarking. 2018. C-Ray. https://openbenchmarking.org/test/pts/c-ray-1.2.0

[25] Open Benchmarking. 2018. DaCapo Benchmark. https://openbenchmarking.org/test/pts/dacapobench-1.0.0

[26] Open Benchmarking. 2018. Gzip Compression. https://openbenchmarking.org/test/pts/compress-gzip-1.2.0

[27] Open Benchmarking. 2018. R Benchmark. https://openbenchmarking.org/test/pts/rbenchmark-1.0.3

[28] Open Benchmarking. 2018. Sanctuary Demo. https://openbenchmarking.org/test/pts/unigine-sanctuary

[29] Open Benchmarking. 2018. SQLite. https://openbenchmarking.org/test/pts/sqlite

[30] Open Benchmarking. 2018. Sunflow Rendering System. https://openbenchmarking.org/test/pts/sunflow-1.1.2

[31] Open Benchmarking. 2018. Timed MAFFT Alignment. https://openbenchmarking.org/test/pts/mafft

[32] Open Benchmarking. 2018. x264. https://openbenchmarking.org/test/pts/x264-2.5.0

[33] Open Benchmarking. 2019. LAME MP3 Encoding. https://openbenchmarking.org/test/pts/encode-mp3-1.7.3

[34] Michael Schwarz, Claudio Canella, Lukas Giner, and Daniel Gruss. 2019. Store-to-Leak Forwarding: Leaking Data on Meltdown-resistant CPUs.

[35] Michael Schwarz, Martin Schwarzl, Moritz Lipp, and Daniel Gruss. 2018. Netspectre: Read arbitrary memory over network. *arXiv preprint arXiv:1807.10535* (2018).

[36] Julian Stecklina and Thomas Prescher. 2018. LazyFP: Leaking FPU Register State using Microarchitectural Side-Channels. *CoRR* abs/1806.07480 (2018).

[37] Phillip Tracy. 2019. ZombieLoad Attacks May Affect All Intel CPUs Since 2011: What to Do Now. https://www.tomsguide.com/us/zombieload-attack-intel-what-to-do,news-30082.html

[38] 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 *Proceedings of the 27th USENIX Security Symposium*. USENIX Association.

[39] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 178–195.

[40] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2019. RIDL: Rogue In-flight Data Load. In *S&P*.

[41] 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* (2018). See also USENIX Security paper Foreshadow [38].

[42] 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.