Systematic Analysis of Kernel Security Performance and Energy Costs

Fabian Rauscher
Graz University of Technology
Graz, Austria
fabian.rauscher@tugraz.at

Benedict Herzog
Ruhr-Universität Bochum
Bochum, Germany
benedict.herzog@rub.de

Timo Hönig
Ruhr-Universität Bochum
Bochum, Germany
timo.hoenig@rub.de

Daniel Gruss
Graz University of Technology
Graz, Austria
daniel.gruss@tugraz.at

Abstract

Security measures and patches typically come with a performance cost. While performance is a common metric to assess the practicality of security measures and patches, energy cost is typically ignored. However, it is unclear to what extent performance and energy costs of security correlate and to what extent they diverge.

In this paper, we present the first systematic analysis of the energy costs of CVE fixes and mitigations. We use the Linux kernel as a case study. We perform energy-performance delta benchmarks using Intel RAPL on the two software versions or configurations under test on our i7-6700K. We automatically attribute CVEs from an 8-year time frame to patch sets, automatically compile the corresponding source code versions, *i.e.*, pre-patch and post-patch, and automatically benchmark performance and energy consumption. Furthermore, we perform an evaluation of all kernel mitigations. We show that energy and performance costs diverge very clearly in some cases. One of these cases is the `retbleed IBPB` mitigation, with runtime increases of 0.4 % and energy consumption decreases of 7.1 % for the `apache` Phoronix benchmark. While not a fully indicative experiment, our work underscores the need for future security research to evaluate energy cost in addition to performance.

CCS Concepts

• **Hardware → Power estimation and optimization**; • **Security and privacy → Operating systems security**.

Keywords

CPU vulnerabilities, energy overhead, Meltdown, Spectre, CVE

ACM Reference Format:
Fabian Rauscher, Benedict Herzog, Timo Hönig, and Daniel Gruss. 2025. Systematic Analysis of Kernel Security Performance and Energy Costs. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3708821.3736197

ASIA CCS '25, Hanoi, Vietnam

ACM ISBN 979-8-4007-1410-8/25/08
https://doi.org/10.1145/3708821.3736197

1 Introduction

System security and efficiency are often seen in conflict. In particular, security measures and patches usually reduce performance. Security mechanisms and patches can be found on any layer of the system, *i.e.*, hardware, operating system, and application level. The standard metric to assess the cost of security mechanisms is measuring performance overhead over benchmarks. Well-known examples of the overheads of security patches include the patch against Meltdown [20, 44], with −5 % to 800 % overhead [19], and the mitigations for various Spectre attack variants, also with a wide range of reported overheads [7]. Some of these are implemented on an application level (e.g., site isolation [57]), on the system level (e.g., KPTI [19]), or on the hardware level (e.g., eIBRS [7]).

Performance overheads of new security mechanisms and mitigations are studied extensively across most corresponding publications. However, performance only indirectly covers energy overheads. Herzog et al. [24] found that for KPTI [19], some benchmarks show different overheads for energy than for performance. The reason is that the energy consumption is influenced significantly by the runtime and the performance mode the processor runs in. For instance, earlier stalling of the out-of-order execution can reduce energy consumption without affecting performance significantly. Conversely, adding memory traffic or on-core activity can increase energy consumption significantly without a significant effect on the performance. This lack of information about the energy overhead of security mechanisms, however, is in stark contrast to the importance of energy efficiency, especially in data centers [17].

A series of works on software energy costs [8, 10, 54], mostly focused on user-level software, consistently showed that software performance and energy costs are only weakly correlated and performance (*i.e.*, execution time) cannot be used to estimate energy costs accurately. Very few works analyzed the energy costs of security mechanisms besides Herzog et al. [24], e.g., Siavvas et al. [60] analyzed the costs of application-level security checks. This is a stark contrast across scientific communities, as our analysis of 1 257 publications from 2023 from 7 top-tier systems and system security conferences reveals. In some systems conferences, more focused on novel functionality and performance gains than security, about 70 % of the published papers mention energy and power consumption. However, only ∼1 % of published papers in top-tier security conferences provide energy and power consumption. This is particularly concerning as there is a clear relation between power

consumption and security, e.g., Rowhammer faults related to newer DRAM operating with lower power [35] and undervolting-related faults [30, 51]. Given frequent new mitigations and the performance of some recent CPUs being degraded to that of 3 years older predecessors [25], *i.e.*, about 15 % slower, we also have to ask:
What are the energy costs of system security? How far do energy and performance costs of system security diverge?

In this paper, we present a systematic analysis of the energy costs of CVE fixes and mitigations in the Linux kernel. The idea behind our systematic analysis is a differential energy measurement using Intel RAPL, a processor interface accurate enough to mount power analysis attacks [43]. By benchmarking the two corner cases of the software under test, *i.e.*, either the source code commits pre-patch and post-patch or the mitigation enabled and disabled, we can determine precise performance and energy overheads. For this purpose, our analysis framework automatically attributes CVEs to patch sets and locates them in the source-code versioning repository.

We evaluate kernel CVE fixes in a case study starting from Linux 4.0 over an 8-year time frame on an i7-6700K. From these, we were able to automatically identify 1 616 CVEs that we can automatically map to patch sets present in the source-code repository. We automatically compile the Linux kernel pre- and post-patch and benchmark it using the Stress-NG and Phoronix benchmark suits. We automatically analyze all fixes through automatic debugging, allowing us to filter for fixes that affect code executed by our benchmarks. We then collected performance and energy data for the 108 CVEs, for which we determined through this debugging that they affect our system. This reduces the number of CVEs for which a large amount of benchmark runs are necessary for statistically significant results while still analyzing all CVEs.

Overall, we obtain energy and performance data for 108 Linux CVEs that affect our system configuration and benchmarks. In this case study, we observe that the energy and performance costs are largely correlated with a few exceptions. We discuss these exceptions, as well as corner cases that reach the limitations of our approach. This work is **not** a cost-benefit analysis. Instead, we introduce a framework for automatically filtering through a large number of security patches and analyzing patches that potentially have large effects on energy consumption or runtime. Additionally, we analyze the correlation of energy and runtime overhead.

Beyond the automated quantitative analysis, we also qualitatively evaluate the performance and energy overheads of mitigations in the Linux kernel. While energy and performance costs are still correlated, we can see more clear corner cases here with clear energy overheads while the performance is largely unchanged and vice versa. This shows that energy cost and performance cost should both be evaluated in future security works.

Contributions. We make the following main contributions:

(1) We automatically run combined energy-performance benchmarks for security patches related to CVEs on our i7-6700K.
(2) Based on our approach, we provide the first large-scale systematic analysis of energy and performance costs of security patches and mitigations in the Linux kernel.
(3) We perform a large-scale analysis of energy and performance costs of kernel patches over an 8-year time frame, revealing significant outliers and divergences.
(4) We perform 14 qualitative case studies of energy and performance costs and use performance counters to reason why energy and performance counters diverge significantly.

Outline. In Section 2, we provide background. In Section 3, we describe our automated performance-energy benchmarking of security patches and mitigations. In Section 4, we present our large-scale analysis of Linux patches over an 8-year time frame. In Section 5, we present our analysis of Linux mitigations. We discuss limitations in Section 6 and related work in Section 7. We conclude in Section 8.

Ethics Considerations. There are no new vulnerabilities discovered or published in this work. We analyzed public CVE patches and mitigations present in the Linux kernel.

2 Background

In this section, we provide background on OS kernel security, the performance overheads of security measures, and CPU energy measurement interfaces for benchmarking and side-channel analysis.

2.1 Kernel Security

Since modern systems run code from arbitrary sources, containerization and process isolation using operating system kernel support have become key elements of system security. The operating system kernel is typically considered part of the trusted computing base (TCB). For decades, the academic community has explored pathways to minimize a system's TCB, e.g., by moving components out of the kernel or only allowing verified code to run in the kernel [11, 48], or entirely switching to a micro-kernel approach [36].

The most widely used open-source kernel today is the Linux kernel, which is also integrated into the Android operating system [1]. Some parts of the kernel source code are continuously improved and extended [23], especially drivers or modules. Furthermore, there are more developers involved in driver or module code, potentially increasing the risk of bugs in these parts of the kernel [61]. Given the large code base of the Linux kernel, it is natural that numerous bugs are introduced and discovered [59]. Furthermore, as the Linux kernel is part of the TCB, any exploitable bug must be considered a security issue that requires a patch [62]. Prior work studied Linux kernel vulnerabilities and found typical C program bug classes to be most prevalent, e.g., buffer errors and invalid dereferences [59]. Consequently, there is a constant stream of security patches for the kernel [32]. Not patching a security issue in the kernel (in time) is a significant security risk [9]. These severe consequences imply that patching security issues is of utmost importance, and the attached costs are an unavoidable side effect.

2.2 Performance Overheads of Security

The performance costs of security patches and security mechanisms are well-studied. Kernel developers are required to keep the performance costs for new security mechanisms to a minimum and justify any overheads [4, 13, 14]. Consequently, the performance of the core functionality of Linux remains consistently fast and does not continuously slow down due to new functionality [58]. An exception to this rule can be mitigations for known vulnerabilities to ensure the integrity and security of the operating system. Nevertheless, such mitigations are continuously analyzed and improved to further reduce the overhead where possible [3, 5, 24, 26, 46]. This applies, in particular, to hardware vulnerabilities, where the vulnerability cannot be remedied directly, but software mitigations must remain permanently active. Besides analyzing the performance costs, additional studies analyze the timeliness and effectiveness of security patches, e.g., in a large study consisting of 4 000 security patches [42]. The energy overhead of security patches, however, has received only little attention so far, with only few works analyzing the overhead of specific mitigations [24, 46].

As a consequence of these considerable performance overheads, the Linux kernel offers many boot-time configuration options to enable or disable specific security measures. Hence, system administrators can select which measures are required in their specific setup. However, also as a consequence of the performance overheads, users are disabling mitigations regardless of their individual exposure to attacks to restore the performance without the patches or mitigations [12, 41]. Even though mitigations can even improve the performance for specific workloads [12]. This already shows that users do not necessarily share the view of the scientific community that the mitigation of security issues is of utmost importance.

2.3 CPU Energy Measurement Interfaces

Both Intel and AMD introduced a mechanism for controlling thermal and power constraints from software [18]. On Intel processors, the Running Average Power Limit (RAPL) mechanism allows software to adjust the CPU frequency and voltage, as well as the power limits. RAPL provides different energy domains for different parts of the processor [21]. AMD provides an energy measurement interface that is essentially compatible with RAPL [2]. The accuracy of energy measurements is typically on the scale of milliseconds or microseconds. Because of its accuracy and precision, RAPL has been used for benchmarking works in the past [24, 34]. RAPL can be accessed through a set of model-specific registers (MSRs), which are available to the kernel. To make them accessible to regular software, Linux provides a driver that allows user-space software to directly read the information provided by RAPL [53]. These integrated energy measurement interfaces significantly lower the barrier for energy measurements compared to measurements with external devices. In particular, they are an enabler for systematic analyses, such as the one presented in this work, as well as for energy overhead measurements for new security mechanisms.

3 Methodology

In this section, we discuss our measurement methodology and test setup. Furthermore, we propose a prefiltering method allowing us to evaluate a large number of CVEs in a short amount of time.

3.1 Measurement Methodology

All our measurements are conducted on an i7-6700K CPU. The benchmarks are executed on an isolated core inside a virtual machine running under KVM with an unmodified Debian 11 using the ext4 filesystem on an Ubuntu 20.04 LTS. An overview of our setup is provided in Figure 1. We run the tested kernels inside a virtual machine similar to the way they would be used in the cloud. All Linux kernel versions are compiled with the default KVM configuration. The energy measurements are done on the host using Intel RAPL. We use the PKG RAPL domain, which provides the energy consumption of the whole CPU. As RAPL does not distinguish between code executed inside a VM and code on the host, energy measurements include the power consumption of the benchmarks running in the VM. We sample the RAPL model-specific register (MSR) in 1 ms intervals throughout all benchmark executions. For runtime measurements, we use a millisecond-accurate timer.

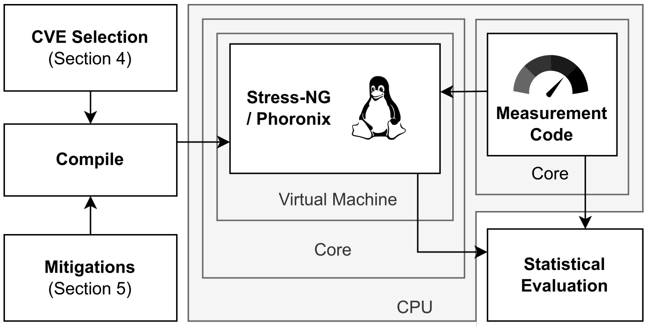

Figure 1: Overview of our analysis framework and experimental setup. The kernels are first compiled. The tested kernel is launched through KVM on its own core and executes the benchmarks. The measurement code runs on its own core, tracking energy consumption through RAPL and the runtime of each benchmark. Once a sufficient number of measurements are completed, the results are evaluated.

Our setup offers fast drop-in replacement of the kernel without requiring a system restart, adapting the measurement code for different kernel versions, relying on the stability of the tested kernels and driver availability in the case of older kernels. This setup allows us to test a wide range of kernel versions and CVE fixes in a short amount of time. We evaluate all Linux CVEs for which an explicit fix commit exists from Linux 4.0 to Linux 6.2, covering an 8 year range of Linux releases. We further evaluate existing command line controlled mitigations in Linux 6.2 and their configurations. To avoid confusion, we refer to mitigations or fixes to software bugs in the Linux kernel as `fixes` or `patches` and mitigations to hardware vulnerabilities, which can be enabled or disabled at boot time as `mitigations`. To find the commits, we use a publicly available database that logs Linux CVEs and fixes [47].

For CVEs, our baseline is the kernel version before the commit marked as the fix is applied. We compare this baseline with the kernel version after the fix is applied. In our evaluated 8 year time period, there are 1 616 CVEs according to [47]. We only evaluate CVEs where an explicit fix commit exists, leaving us with 1 604 CVEs. Of the 1 604 CVEs, 46 did not compile or boot on our system, resulting in 1 545 testable CVEs. Build problems for some commits are to be expected as we do not build Linux releases but specific commits. We run multiple benchmarks to evaluate the impact of CVE fixes on performance and energy consumption. Our benchmark set consists of 14 benchmarks from the Stress-NG suite and 5 benchmarks of the Phoronix Test Suite. The Stress-NG suite benchmarks used are `icache`, `fork`, `pthread`, `context`, `pipe`, `io`, `sock`, `udp`, `futex`, `aio`, `switch`, `sctp`, `signal`, and `cpu`. The Phoronix Test Suite benchmarks are `network-loopback`, `mutex`, `osbench`,

Table 1: Discussions and Evaluations of Performance- and Energy-related Metrics in Top Publications.

Papers discussing	ISCA	MICRO	ASPLOS	ACM CCS	IEEE S&P	NDSS	USENIX Sec.
Overall	151	229	195	69	101	90	422
Performance	55	41	38	20	33	18	84
Energy	37	20	13	47	68	7	30
Perf. & Energy mentioned	25	8	10	17	29	5	19
Performance & Energy	13	5	8	15	21	2	9
Energy measured, estimated, or discussed	8	2	3	11	16	1	3

`apache`, and `pmbench`. We chose our benchmarks to cover a wide range of use cases while keeping the number of benchmarks low. While more benchmarks would be beneficial, it would result in significantly longer measurement times, as many kernels have to be tested with each one. The low number of benchmarks allows us to test a wide range of Linux versions in a reasonable amount of time. We further prefilter benchmarks for each CVE to only test benchmarks that change code executed during a benchmark run. Furthermore, we do not explicitly enable or disable any kernel mitigations when testing CVE fixes.

For Linux kernel mitigations, we run all benchmarks with all available command line options for each mitigation on Linux 6.2. Our baseline for the mitigation measurements is the kernel with the tested mitigation disabled. At the time of testing the available mitigations according to the official documentation for our x86 system are `spectre_v2`, `spectre_v1`, `spec_store_bypass_disable`, `pti`, `l1tf`, `mds`, `tsx_async_abort`, `retbleed`, `mmio_stale_data`, `l1d_flush`, and `kvm.nx_huge_pages` [33]. We do not apply prefiltering for the command line options, as the number of mitigations that can be enabled this way is manageable.

3.2 Methodologies across Communities

To understand the current situation of energy measurements in the security community, we performed a comparative study with the systems community. We want to determine to which extent energy costs are discussed in systems and system security publications and identify methodological discrepancies between the communities.

We selected the top system security venues (ACM CCS, IEEE S&P, NDSS, and USENIX Security) as well as the top systems venues (ASPLOS, ISCA, and MICRO). We used a semi-automated approach to retrieve the publications for all 7 conferences in 2023 from the publisher websites. While this worked perfectly for some conferences, one can notice that the number of publications is just slightly below the number of accepted papers for some conferences due to missing publications[1], or repeatedly failed download attempts from the publisher. Thus, overall, we base our evaluation on 1 257 unique papers published at top systems and system security venues. Most publications are in the same length range, given the tight and similar page limits. Despite the differences in their respective topics, both software and hardware mechanisms have been published in each of these 7 conferences. In systems conferences, both publications focused on novel functionality and performance gains, and publications proposing new security mechanisms can be found.

By-hand evaluating all 1 257 papers is a prohibitive amount of work, even when involving multiple experts for the evaluation. Therefore, we pre-filtered the papers based on an automated keyword search. To identify papers discussing performance-related aspects, we used the expression `(performance|run.?time|execution.time|CPU.time).(cost|consumption|overhead|increase)`. To identify papers discussing energy-related aspects, we used the expression `(energy|power)(.consumption)?.(cost|consumption|overhead|increase)`. It is clear that this approach may have missed some papers, but based on our manual checks, this filter is representative of the entire set of 1 257 papers. To filter papers discussing performance-related aspects further, we used the expression `(energy|power.consumption)` and refined it to also use the expression for energy-related aspects above. All resulting publications were manually evaluated by an expert.

As shown in Table 1, our automated keyword analysis yields that 152 out of 321 systems publications from 2023 mention energy efficiency or costs or power consumption. For ASPLOS, this number is around 25 %, whereas for ISCA and MICRO, it is closer to 70 %. We can also see that for those papers that mention performance costs or overheads, more than 85 % of ISCA and MICRO papers also mention energy, and about 45 % of ASPLOS papers. For the system security venues, the numbers look more devastating: Only 63 out of 846 system security publications in our evaluation mention energy or power consumption, corresponding to only 7 %. This is a significant discrepancy to the systems community.

This difference does not exist to the same level for performance metrics, where all conferences are in the range of 18 % to 36 %. This discrepancy can be explained by attack papers that do not present a mitigation in detail and, hence, also no performance evaluation.

To narrow down the set for manual analysis by experts further, we combined the two initial filters, resulting in a selection of 73 publications that possibly discuss performance- and energy-related aspects. We manually evaluated these 73 publications. The manual analysis should identify how many of these 73 publications provide energy costs through measurements, estimations, or at least a discussion of energy costs in a wider sense. The result of the manual analysis shows that only 44 of the 73 publications (1 257 overall) our search identified actually discuss energy costs. For systems conferences, an overall number of 35 out of 49 publications (321 overall) were identified to discuss energy costs, whereas, for system security, it is only 9 out of 24 (936 overall). However, taking into account that the remaining publications already did not match the keywords in our automated search, our manual analysis provides estimates for the overall ratio of publications that discuss energy additionally to performance: For system security, we can estimate the ratio to ≈1 % that discuss energy and power consumption; for systems publications, we can estimate the ratio to ≈11 %.

This discrepancy indicates methodological differences that need to be addressed. During our manual analysis, we discovered numerous works that could have provided energy costs or estimates, for instance, by using CPU energy power measurements (e.g., using RAPL and equivalent features [2, 18]), or software-based estimates (e.g., using CACTI [50]).

[1] Instead of the paper PDF, the publisher provided a PDF stating that the corresponding paper is not available.

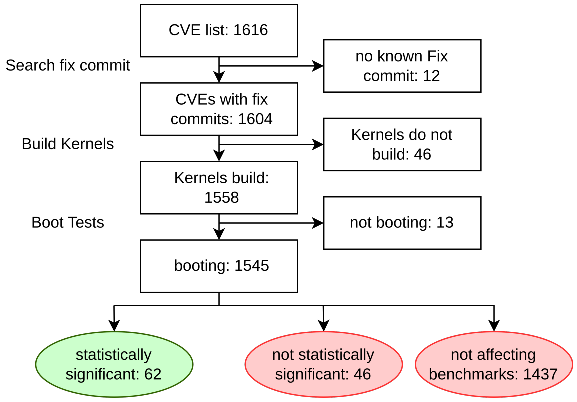

Figure 2: Statistics on how many CVEs exist for the tested time frame, how they were processed in our benchmarking approach, and which led to statistically significant results.

4 Analysis of 8 Years of Kernel CVEs

In this section, we analyze the measurement results of our benchmarks on 1 616 CVEs, which were introduced to the Linux kernel between Linux 4.0 and Linux 6.2.

4.1 Benchmark Filtering

A high-level overview of our filtering process is provided in Figure 2. For our evaluation period of Linux 4.0 to Linux 6.2, we found 1 616 CVEs. For 12 of the CVEs, there is no explicit commit marked as a fix [47]. For further 46 CVEs, the unpatched or the patched kernel does not compile. Of the remaining 1 558 CVEs, 13 had at least one kernel that did not boot. This results in 1 545 testable CVEs.

With 1 545 CVEs, a complete run with all benchmarks for each CVE would take over two weeks on our test system. For a high enough sample size, our system would have to run for at least a year. Such a long runtime is not practical. As we do not want to reduce the number of CVEs or benchmarks, we prefilter the benchmarks for each CVE. While there are 1 545 testable CVEs, only a fraction affect the code that our benchmarks run. These CVEs might only affect Android, ChromeOS, other architectures, or drivers of devices and file systems not present in our test system. By filtering CVEs and benchmarks for CVEs that affect code that is not executed, we significantly reduce the number of overall benchmark runs.

For filtering, we leverage automatic debugging in combination with the ability to debug kernels in QEMU. First, we find all code lines changed by the CVE patch. Second, we set a breakpoint for each changed line. Third, we execute a benchmark and log each breakpoint hit. We use the resulting information to remove CVEs that do not affect any benchmarks and only run benchmarks that execute changed code. Furthermore, we do not execute benchmarks for CVEs if breakpoints were only hit less than 100 times, as the changes are not often executed. A low number of breakpoint hits can be the result of the changed code executed once during setup, which will only marginally impact the benchmark. By removing all CVEs that do not affect code executed by our benchmarks, we reduce the 1 545 testable CVEs to 108 CVEs. This number can be further increased through more benchmarks and testing on different architectures and hardware configurations. The prefiltering is automated and does not require manual intervention.

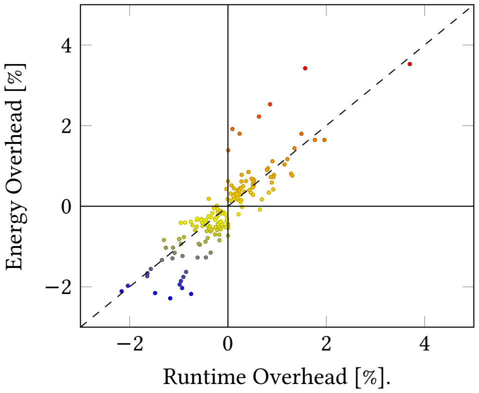

Figure 3: Energy overhead and runtime overhead scatter plot of benchmark runs from 108 Linux kernel CVE fixes.

4.2 High-Level Analysis

Figure 3 provides an overview of the benchmark results for all 108 tested CVEs. Each dot in the scatter plot corresponds to the result of a single benchmark run. Each benchmark was executed 92 times, and the results are averaged. Black lines separate the four quadrants, and the expected runtime-to-energy correlation of 1:1 is shown as a dotted line. Of the 108 CVEs, 46 have no benchmark runs where either the runtime or energy change show statistical significance according to the Mann-Whitney-U-Test ($p \geq 5\%$). This means that 62 out of the 108 measured CVE fixes have a measurable impact on our benchmark suite’s runtime or energy consumption. 42.2 % of benchmark runs are in the first quadrant, meaning positive runtime and energy overhead. 53.0 % of benchmark runs are in the third quadrant, meaning negative runtime and energy overhead. Results in the first and third quadrants follow the assumption that energy overhead can be roughly estimated by runtime overhead. 2.4 % of benchmark runs fall into the fourth quadrant, meaning positive runtime overhead and negative energy overhead. For these CVEs, the corresponding benchmarks took longer to execute but did so in an overall more energy-efficient way than without the patch. 2.4 % of benchmark runs fall into the second quadrant, meaning negative runtime overhead and positive energy overhead. Therefore, energy and runtime are largely correlated. Contrary to what is expected, there is a large number of patches that improve energy consumption and runtime. This can be the result of code simplification but also more complex changes, such as fewer branch predictions and more energy-efficient stalls instead of mispredictions.

47.6 % of benchmark runs with statistically significant results have either a runtime or energy overhead change that is statistically significant, while the corresponding other metric is statistically insignificant. This means that the patch affected either energy consumption or runtime overhead, but not both. Following the assumption that energy and runtime follow a 1:1 relationship, faster code should be more energy-efficient. While energy and runtime seem to roughly correlate (Figure 3), the correlation is far from perfect, with the measured overheads scattered around the expected correlation. For optimizations of fixes, typically, only runtime overhead is considered, while energy overhead is ignored. This one-sided optimization results in an energy overhead that can vary wildly from the runtime overhead. The Pearson correlation coefficient of energy and runtime overhead is 0.84 with a coefficient of determination (r^2) of 0.71, where 1 would be a perfect linear correlation. Our measurements show that runtime is a rough estimate for energy consumption, but it is not precise and can vary significantly.

For all CVEs that we further analyzed, the performance and energy changes can be explained by code changes or by changes in performance counter values. Given that some results significantly deviate from the expected 1:1 correlation between energy and runtime shown in Figure 3, it is crucial to not only measure runtime overhead but also energy overhead when testing CVE patches.

4.3 Case Studies

In this section, we discuss interesting results of our measurements. We discuss a selection of CVEs and corresponding benchmarks with a significant change in runtime or energy consumption according to the Mann-Whitney-U-Test ($p \geq 5\%$). For each case study, we first compare the code changes with the benchmark results. Second, we look at the change of performance counter values between the unpatched and patched kernels to gain further insights. In particular, we track stalled cycles, dTLB-load and dTLB-store misses, iTLB-load and iTLB-store misses, LLC-load and LLC-store misses, L1-dcache loads and stores, branch loads, branch misses, instructions executed, mispredicted branches retired, uOPs issued, stalled cycles to recover from a misprediction, and the number of times the front end is resteered, mainly when the branch predictor cannot provide a correct prediction (BACLEARS). We repeated the measurements for the CVEs discussed in this section to collect the performance counter-values. We executed the benchmarks 65 times while tracking the performance counters. The newly collected results are consistent with our first measurements, where we collected results from all CVEs. This further proves the repeatability and consistency of our results. For all results discussed, we provide the average overhead in percent, the standard error of the mean $\sigma_{\bar{x}}$ and the sample size n. To improve readability, numbers that increase with the patch applied are additionally colored green and numbers that decrease red.

CVE-2017-1000112. This CVE belongs to an exploitable memory corruption in the UDP Fragmentation Offload (UFO) implementation in the Linux kernel [47]. The bug was fixed by adding checks to four if conditions in the UDP, IPv4, and IPv6 implementations. The fix results in a runtime overhead of 1.5 % ($\sigma_{\bar{x}} = 0.25\,\%$, $n = 65$) and an energy overhead of 1.4 % ($\sigma_{\bar{x}} = 0.042\,\%$, $n = 65$) for the Stress-NG udp benchmark. Despite this significant change in runtime and energy consumption, the performance counters indicate no increase in the number of instructions executed. The increase stems from an increase in stalled cycles by 2.4 % ($\sigma_{\bar{x}} = 0.7\,\%$, $n = 65$) and an increase of stalled cycles used to recover from an earlier branch misprediction of machine clear event by 12.8 % ($\sigma_{\bar{x}} = 3.1\,\%$, $n = 65$). Furthermore, we detected an increase of L1-dcache loads and stores by 2.3 % ($\sigma_{\bar{x}} = 1.23\,\%$, $n = 65$) and 3.0 % ($\sigma_{\bar{x}} = 1.2\,\%$, $n = 65$) respectively. The udp benchmark is the only benchmark for this CVE with a statistically significant runtime or energy overhead, perfectly matching the fix that changes the UDP implementation.

CVE-2020-29534. This CVE belongs to a bug where `io_uring` takes a non-recounted reference to the file struct that submitted a request [47]. For Stress-NG pipe the runtime overhead decreases by 0.6 % ($\sigma_{\bar{x}} = 0.46\,\%$, $n = 65$) and the energy overhead by 0.9 % ($\sigma_{\bar{x}} = 0.10\,\%$, $n = 65$). The dTLB store misses decrease by 4.6 % ($\sigma_{\bar{x}} = 0.6\,\%$, $n = 65$), the branch misses by 0.92 % ($\sigma_{\bar{x}} = 0.68\,\%$, $n = 65$), and the number of cycles stalled due to recovery from a branch miss by 2.1 % ($\sigma_{\bar{x}} = 1.36\,\%$, $n = 65$). Contrary to that, the BACLEARS increase by 3.6 % ($\sigma_{\bar{x}} = 1.54\,\%$, $n = 65$) and the LLC loads increase by 3.9 % ($\sigma_{\bar{x}} = 1.50\,\%$, $n = 65$). From these results, we assume that the decrease in branch misses and the cycles used to recover from branch misses are the reasons for the improvements.

For Stress-NG sock the runtime overhead decreases by 0.39 % ($\sigma_{\bar{x}} = 0.08\,\%$, $n = 65$) and the energy overhead decreases by 0.43 % ($\sigma_{\bar{x}} = 0.02\,\%$, $n = 65$). The branch misses decrease by 1.5 % ($\sigma_{\bar{x}} = 0.50\,\%$, $n = 65$), the L1-dcache loads decrease by 3.0 % ($\sigma_{\bar{x}} = 1.31\,\%$, $n = 65$). Similarly to the pipe benchmark, the improvements for the sock benchmark are most likely due to fewer branch misses.

Insight 1. By optimizing for branch prediction, patches can, even with more complex code, significantly decrease energy and runtime overhead.

For Stress-NG udp, runtime overhead decreases by 1.3 % ($\sigma_{\bar{x}} = 0.19\,\%$, $n = 65$) and energy overhead by 1.3 % ($\sigma_{\bar{x}} = 0.04\,\%$, $n = 65$). The dTLB store misses decrease by 7.5 % ($\sigma_{\bar{x}} = 0.63\,\%$, $n = 65$), L1-dcache stores by 1.5 % ($\sigma_{\bar{x}} = 0.47\,\%$, $n = 65$), LLC loads by 6.7 % ($\sigma_{\bar{x}} = 0.72\,\%$, $n = 65$), instructions executed by 1.2 % ($\sigma_{\bar{x}} = 0.45\,\%$, $n = 65$), and BACLEARS by 6.0 % ($\sigma_{\bar{x}} = 0.70\,\%$, $n = 65$). However, cycles stalled due to misprediction increase by 21.4 % ($\sigma_{\bar{x}} = 4.2\,\%$, $n = 65$). Energy consumption and runtime most likely improved due to fewer instructions executed and lower cache pressure.

CVE-2020-12114. This CVE is a race condition in `fs/namespace.c` allowing users to cause a denial of service [47]. The fix results in a runtime overhead of 1.9 % ($\sigma_{\bar{x}} = 0.35\,\%$, $n = 65$) and an energy overhead of 2.0 % ($\sigma_{\bar{x}} = 0.044\,\%$, $n = 65$) for Stress-NG pipe. The instructions executed increase by 2.1 % ($\sigma_{\bar{x}} = 0.5\,\%$, $n = 65$), uOPs issued by 3.0 % ($\sigma_{\bar{x}} = 1.03\,\%$, $n = 65$), dTLB load misses by 2.4 % ($\sigma_{\bar{x}} = 0.50\,\%$, $n = 65$), dTLB store misses by 3.3 % ($\sigma_{\bar{x}} = 0.83\,\%$, $n = 65$), and L1-dcache stores by 1.4 % ($\sigma_{\bar{x}} = 0.64\,\%$, $n = 65$). The stalled cycles from misprediction decreased by 5.9 % ($\sigma_{\bar{x}} = 1.50\,\%$, $n = 65$), while the overall stalled cycles stalled increased by 2.6 % ($\sigma_{\bar{x}} = 1.010\,\%$, $n = 65$). Therefore, the runtime and energy overhead changes are most likely due to more code being executed.

For Stress-NG udp, the runtime increases by 0.75 % ($\sigma_{\bar{x}} = 0.19\,\%$, $n = 65$) and the energy consumption by 0.87 % ($\sigma_{\bar{x}} = 0.02\,\%$, $n = 65$). The amount of stall cycles increased by 2.6 % ($\sigma_{\bar{x}} = 1.09\,\%$, $n = 65$) while the instructions executed only slightly increased by 0.68 % ($\sigma_{\bar{x}} = 0.48\,\%$, $n = 65$). Similar to the pipe benchmark, the stalled cycles due to recovery from a misprediction decreased by 14.6 % ($\sigma_{\bar{x}} = 1.51\,\%$, $n = 65$). We did not observe an increase in TLB or LLC misses. We did observe an increase in BACLEARS by 1.4 % ($\sigma_{\bar{x}} = 0.70\,\%$, $n = 65$), which, together with other factors that we did not track, likely resulted in the overall increase in stalled cycles.

Therefore, the change in runtime and energy overhead is likely due to the overall increase in stalls and the reason behind them.

For the network-loopback benchmark, the runtime increased by 0.25 % ($\sigma_{\bar{x}} = 0.06\,\%$, $n = 65$) with no change in energy consumption. This is contrary to benchmark runs form other CVEs, where runtime and energy overhead change at a similar rate. From all tracked performance counters, only stalled cycles due to misprediction recovery increase by 3.2 % ($\sigma_{\bar{x}} = 1.70\,\%$, $n = 65$). The change in runtime is likely due to very subtle changes in performance counter values, which would require a higher sample size to detect, or due to reasons that we do not track. The difference between energy overhead and runtime overhead could stem from a more efficient way of stalling than stalling from misprediction recovery.

For all affected benchmarks, the changed code is heavily executed. The network-loopback benchmark, in particular, executes changed code lines 159 040 times in its execution.

CVE-2018-1108. This CVE belongs to a weakness in the generation of random seed data. The weakness allows programs to use the random seed before it was sufficiently generated [47]. The fix results in a runtime overhead of 0.72 % ($\sigma_{\bar{x}} = 0.07\,\%$, $n = 65$) and an energy overhead of 1.7 % ($\sigma_{\bar{x}} = 0.24\,\%$, $n = 65$) for network-loopback. While this fix is intended for early boot-time random number generation by updating four if-conditions in the random number generator code, it does affect programs even after that. Some of the changed conditions are frequently executed during random number generation. The network-loopback benchmark, in particular, executed the changed code over 2 600 times per run. Due to this, Branch misses increased by 4.6 % ($\sigma_{\bar{x}} = 1.49\,\%$, $n = 65$) and dTLB load and store misses by 2.1 % ($\sigma_{\bar{x}} = 0.69\,\%$, $n = 65$) and 1.5 % ($\sigma_{\bar{x}} = 0.7\,\%$, $n = 65$) respectively. This fix results in an energy overhead that is significantly higher than the runtime overhead for this benchmark. The result of the overhead appears to stem from an increase in mispredictions due to changed if-conditions.

Insight 2. Changes that should only be executed rarely, e.g., at boot time, can have a significant impact on regular runtime through misprediction of added branches that are rarely taken.

CVE-2015-8839. This CVE belongs to multiple race conditions in the ext4 implementation [47]. For the pipe Stress-NG benchmark the runtime increases by 1.9 % ($\sigma_{\bar{x}} = 0.19\,\%$, $n = 65$) and the energy consumption by 1.5 % ($\sigma_{\bar{x}} = 0.03\,\%$, $n = 65$). For this benchmark the BACLEARS increase by 7.2 % ($\sigma_{\bar{x}} = 2.86\,\%$, $n = 65$), branch misses by 2.6 % ($\sigma_{\bar{x}} = 0.59\,\%$, $n = 65$), LLC loads by 8.4 % ($\sigma_{\bar{x}} = 2.7\,\%$, $n = 65$), L1-dcache stores by 2.9 % ($\sigma_{\bar{x}} = 1.74\,\%$, $n = 65$), and instructions executed by 1.4 % ($\sigma_{\bar{x}} = 0.54\,\%$, $n = 65$).

For Stress-NG aio, runtime increases by 2.9 % ($\sigma_{\bar{x}} = 0.58\,\%$, $n = 65$) and energy consumption by 2.4 % ($\sigma_{\bar{x}} = 0.13\,\%$, $n = 65$). For this benchmark, BACLEARS increase by 5.0 % ($\sigma_{\bar{x}} = 1.80\,\%$, $n = 65$), dTLB store misses by 21.4 % ($\sigma_{\bar{x}} = 4.23\,\%$, $n = 65$), dTLB load misses by 15.8 % ($\sigma_{\bar{x}} = 5.20\,\%$, $n = 65$), cycles stalled by 5.8 % ($\sigma_{\bar{x}} = 1.75\,\%$, $n = 65$), and instructions executed by 2.0 % ($\sigma_{\bar{x}} = 1.52\,\%$, $n = 65$).

For Stress-NG udp the runtime increases by 3.2 % ($\sigma_{\bar{x}} = 0.26\,\%$, $n = 65$) and the energy consumption by 2.7 % ($\sigma_{\bar{x}} = 0.15\,\%$, $n = 65$). For this benchmark BACLEARS increase by 2.0 % ($\sigma_{\bar{x}} = 0.64\,\%$, $n = 65$), branch misses by 4.1 % ($\sigma_{\bar{x}} = 0.95\,\%$, $n = 65$), LLC loads by 1.7 % ($\sigma_{\bar{x}} = 0.64\,\%$, $n = 65$), LLC stores by 6.9 % ($\sigma_{\bar{x}} = 1.37\,\%$, $n = 65$), L1-dcache stores by 4.5 % ($\sigma_{\bar{x}} = 0.67\,\%$, $n = 65$), and instructions executed by 3.0 % ($\sigma_{\bar{x}} = 0.91\,\%$, $n = 65$). Furthermore, branch misses increase by 3.4 % ($\sigma_{\bar{x}} = 1.14\,\%$, $n = 65$).

For the network-loopback benchmark the runtime increases by 2.4 % ($\sigma_{\bar{x}} = 0.15\,\%$, $n = 65$) and the energy consumption by 2.4 % ($\sigma_{\bar{x}} = 0.15\,\%$, $n = 65$). For this benchmark BACLEARS increased by 7.2 % ($\sigma_{\bar{x}} = 2.86\,\%$, $n = 65$), and LLC loads by 3.4 % ($\sigma_{\bar{x}} = 2.7\,\%$, $n = 65$). During a run of the network-loopback benchmark, the VM executes over 700 000 times code modified by the CVE fix.

As our VM uses ext4 as its filesystem, a change in the ext4 implementation affects a large number of benchmarks. These results show that minor changes in code frequently executed by most applications can significantly impact runtime and energy consumption.

Insight 3. Unoptimized branches in CVE patches in code important for regular OS operation can have drastic negative impacts on a wide range of applications.

CVE-2016-5696. This CVE belongs to a bug in the IPv4 TCP stack that results in improper ACK segment rate determination, simplifying TCP hijacking through a blind in-window attack [47]. The fix decreases the runtime and energy consumption of Stress-NG sock by 0.9 % ($\sigma_{\bar{x}} = 0.16\,\%$, $n = 65$) and 1.0 % ($\sigma_{\bar{x}} = 0.046\,\%$, $n = 65$) respectively. Furthermore, retired branches decrease by 4.8 % ($\sigma_{\bar{x}} = 1.78\,\%$, $n = 65$), and stall cycles to recover from earlier mispredictions by 16.0 % ($\sigma_{\bar{x}} = 2.53\,\%$, $n = 65$). This benchmark is affected by this fix, as it tests socket performance by setting up a server and client that transmit packages. The fix appears to decrease the mispredictions when executing Stress-NG sock, decreasing performance and energy consumption.

CVE-2017-7495. This CVE belongs to a bug in fs/ext4/inode.c of the ext4 implementation. The implementation mishandles a needs-flushing-before-commit list with data=ordered mode, allowing users to obtain sensitive information from other users' files [47]. For the icache benchmark runtime decreases by 2.5 % ($\sigma_{\bar{x}} = 0.25\,\%$, $n = 65$) and energy overhead by 2.4 % ($\sigma_{\bar{x}} = 0.03\,\%$, $n = 65$). The reason for these decreases is a decrease in instructions executed by 2.8 % ($\sigma_{\bar{x}} = 0.34\,\%$, $n = 65$).

For the network-loopback benchmark the runtime decreases by 0.5 % ($\sigma_{\bar{x}} = 0.05\,\%$, $n = 65$) and the energy overhead by 1.1 % ($\sigma_{\bar{x}} = 0.03\,\%$, $n = 65$). BACLEARS decrease by 2.7 % ($\sigma_{\bar{x}} = 0.93\,\%$, $n = 65$), branch misses by 3.2 % ($\sigma_{\bar{x}} = 1.49\,\%$, $n = 65$), and L1-dcache stores by 3.5 % ($\sigma_{\bar{x}} = 1.76\,\%$, $n = 65$). The lower runtime and energy overhead likely stems from fewer branch misses, resulting in efficient code execution.

5 Analysis of Mitigations

In this section, we discuss our results of all available Linux mitigations changeable by command line options. We run the same benchmarks as in Section 4. A wide variety of mitigations have been introduced over the past years. We benchmark all mitigation options available in Linux kernel 6.2 on an x86 system separately and evaluate the runtime and energy consumption changes introduced by them. We list the available mitigation options and their parameters in Table 2. For each mitigation, we benchmark all available options and compare them with the deactivated mitigation. While most mitigations can only be turned on or off, some can

Table 2: Tested mitigation options on Linux Kernel 6.2.

Mitigation	Options
spectre_v2	off; retpoline; retpoline,generic; retpoline,lfence; eibrs; eibrs,retpoline; eibrs,lfence; ibrs; on
spectre_v1	nospectre_v1; spectre_v1
pti	off; on
spec_store_bypass_disable	off; on
l1tf	off; flush; full
mds	off; full,nosmt; full
tsx_async_abort	off; full
retbleed	off; unret; ibpb
mmio_stale_data	off; full
l1d_flush	on
kvm.nx_huge_pages	off; force

be configured to include a specific mitigation strategy. One of the most configurable mitigations are the Spectre V2 mitigations, with 8 different specific mitigation options.

5.1 Overview

In this section, we present the results of our measurements using 54 benchmark runs, with all not tested mitigation disabled, which are provided in Table 3 (runtime) and Table 4 (energy). All numbers are rounded to one decimal place if they are <10 and >−10 and to the next integer otherwise. There are multiple runs with overheads listed as 0, which result from very small changes in runtime or energy consumption (<0.1 %). Measurements with no statistically significant overhead according to the Mann-Whitney-U-Test ($p \geq 5\%$) are represented by *. `l1tf`, `l1d_flush`, and `kvm.nx_huge_pages` show (almost) no statistically significant change in runtime or energy consumption. This is as expected, as these mitigations target either SGX or the execution of VMs. While our tested kernel runs inside of a VM, it itself does not manage any VMs in any of our benchmarks. The only exceptions for runtime are the `pthread` (p-value: 3.4 %) and `pipe` (p-value: 1.43 %) benchmarks for `l1tf=flush` and the `cpu` benchmark (p-value: 1.45 %) for `l1d_flush`. The only exceptions for energy overhead are the `pthread` (p-value: 3.76 %), `pipe` (p-value: 0.46 %), and `mutex` (p-value: 1.45 %) benchmark for `l1tf=flush` as well as the `pipe` (p-value: 1.84 %), and `mutex` (p-value: 4.86 %) benchmark for `l1tf=full`. We reran these benchmarks and could not find any statistical significance in the newly collected data. Therefore, we conclude that these were false positives. A small number of false positives are expected due to the high number of benchmarks executed and the low sample size.

Overall, energy consumption and runtime correlate well with a few exceptions discussed later, as shown in Figure 4 with a zoomed-in view in Figure 5. Each point corresponds to one benchmark execution where at least the energy overhead or runtime overhead is statistically significant. We highlight the cardinal axes using solid lines and the expected 1:1 correlation with a dashed line. When linearly approximating energy overhead through runtime, we computed the polynomial $e = 0.84r + 1.14$, where e is the energy overhead and r the runtime overhead, with an $r^2 = 0.992$ (1 would be a perfect representation), indicating that this is a very accurate approximation. The polynomial indicates that energy overhead is typically lower than runtime overhead, as shown by the 0.84 coefficient. Calculating the r^2 score for the samples with a small runtime and energy overhead in the range of $[-10, 10]$ shown in Figure 5, still containing roughly half of the measurements, results in $r^2 = 0.52$. While this linear model works well for large overheads, it does not provide accurate results for smaller overheads.

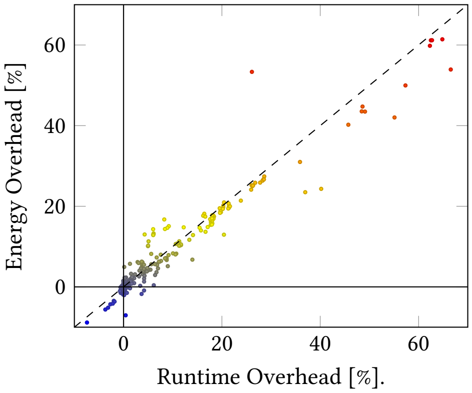

Figure 4: Linux kernel hardware mitigation energy and runtime overhead.

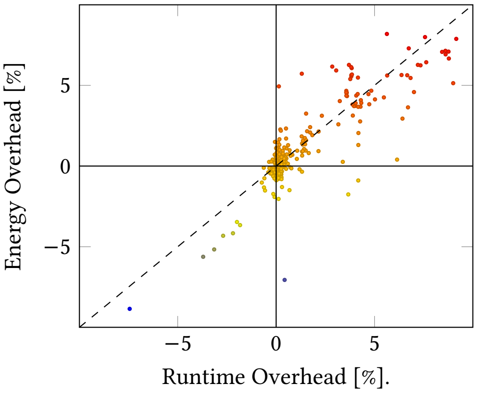

Figure 5: Linux kernel hardware mitigation energy and runtime overhead zoomed-in with an x-range of $[-10, 10]$ and a y-range of $[-10, 10]$.

5.2 High-Level Analysis

While mitigations have a significant impact on the runtime and energy consumption of micro benchmarks, the impact on macro benchmarks is limited, as shown in Table 3 (runtime) and Table 4 (energy). For macro benchmarks, Phoronix `network-loopback` performs the worst, as the network stack is handled by the kernel, followed by Phoronix `osbench files`, which continuously creates files. Some benchmarks are only minimally affected by the mitigations, such as Stress-NG `cpu` and Phoronix `pmbench`. As the tested mitigations only change kernel behavior, minimizing kernel time through fewer syscalls minimizes the overhead. As the `cpu` and `pmbench` benchmarks focus on computations in user space and

Table 3: Runtime overhead of all measured software mitigations in percent. Values are rounded to one decimal point for changes <10 % and to the next integer for larger changes. Overheads that are not statistically significant are replaced by a ∗-symbol.

	Mitigation																			
Benchmark	spectre_v2=retpoline	spectre_v2=retpoline,generic	spectre_v2=retpoline,lfence	spectre_v2=eibrs	spectre_v2=eibrs,retpoline	spectre_v2=eibrs,lfence	spectre_v2=ibrs	spectre_v2=on	spec_store_bypass_disable=on	spectre_v1	pti=on	l1tf=flush	l1tf=full	mds=full	tsx_async_abort=full	retbleed=unret	retbleed=ibpb	mmio_stale_data=full	l1d_flush=on	kvm.nx_huge_pages=force
icache	4.3	4.3	*	3.9	4	4.3	71	4.7	*	*	17	*	*	16	16	6.4	238	16	*	*
fork	1.8	1.3	0.7	1.3	1.5	1.4	35	11	2.2	*	6.8	*	*	5	4.8	5.6	66	4.7	*	*
pthread	−7.4	*	*	*	*	*	36	*	−41	*	*	5.5	*	23	27	−9.9	379	25	*	*
context	3.6	3.6	4.3	3.6	3.6	3.6	322	3.6	0.7	*	87	*	*	96	96	5.6	972	96	*	*
pipe	21	21	13	21	21	21	81	55	1.5	*	19	0.5	*	19	20	10	261	19	*	*
io	8.8	8.8	7	10	8.6	10	40	8.6	13	*	6.7	*	*	7.3	6.4	7.6	75	9	*	*
sock	17	17	10	17	17	18	177	16	*	*	45	*	*	48	48	7.2	536	49	*	*
udp	26	26	14	26	26	26	109	313	1.4	*	15	*	*	18	18	15	347	17	*	*
futex	11	11	9.1	11	11	11	112	174	64	*	20	*	*	9.2	8.9	6.7	214	8.4	*	*
aio	17	18	8.4	17	17	18	354	18	1.1	*	84	*	*	94	94	19	1,147	94	*	*
switch	28	28	17	28	28	28	125	404	1.4	*	21	*	*	20	20	10	387	20	*	*
sctp	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	78	*	*	*
signal	4.1	4	3.2	4.2	4	3.9	233	4.1	1.3	*	57	*	*	62	62	7.6	734	62	*	*
cpu	*	0.1	*	0.1	*	*	1.7	0.1	6.7	*	*	*	*	0.1	*	*	3.2	*	−0.2	*
network-loopback	6.1	6	2.8	6.1	6	6	26	55	0.3	*	8.3	*	*	5.1	5	3.1	89	5	*	*
mutex	−0.1	−0.1	−0.1	−0.1	−0.1	0.1	0.5	−0.1	−0.1	*	0.2	*	*	0.2	−0.1	0.2	1.5	0.2	*	*
osbench files	3.8	3.8	2.3	3.8	3.8	3.9	4.2	3.8	0.4	*	1.5	*	*	1.6	1.7	3.7	12	1.5	*	*
osbench processes	0.1	0.1	0	0.1	0.1	0.1	1.3	0.2	0.1	*	0.3	*	*	0.3	0.3	0.2	4.2	0.3	*	*
osbench threads	0.1	0.1	0.1	0.1	0.1	0.1	1.3	0.2	0.1	*	0.3	*	*	0.3	0.3	0.2	4.2	0.3	*	*
osbench programs	0.1	0.1	0.1	0.2	0.1	0.1	1.4	0.2	0.1	*	0.3	*	*	0.3	0.3	0.2	4.2	0.3	*	*
osbench allocations	0.2	0.2	0.1	0.2	0.2	0.2	1.3	0.3	0.1	*	0.2	*	*	0.2	0.2	0.3	4.3	0.2	*	*
apache	0	0	*	0	0	0	0.1	0	*	*	0	*	*	0	0	0	0.4	0	*	*
pmbench	0	0	0	0	0	0	0.2	0	0	*	0	*	*	0	0.1	0	0.7	0	*	*

memory accesses, respectively, they are only minimally affected. Despite this, they are still affected by forced switches to kernel mode, e.g., through interrupts. The Spectre V1 mitigation shows that security does not have to come at a cost, as it is optimized enough that we did not detect any significant overhead.

Similar to CVE patches (Section 4), energy and runtime are largely correlated, especially for larger overheads as shown in Figure 4. Despite this, for smaller overheads, the correlation is significantly weaker (Figure 5), and in some cases, the two metrics are completely different. Due to this deviation, similar to CVE patches, it is crucial to not only measure runtime but also energy overhead when testing mitigations.

Our measurements also show an untapped way of optimizing energy consumption. Two examples of this are shown in the Phoronix apache benchmark for the Spectre V2 ibrs and the Retbleed ibpb mitigations. While the runtime only marginally increases with these mitigations, the energy consumed drastically decreases. This is the result of more cycles stalled instead of misspeculation. There seem to be code parts in the kernel where stalling saves a significant amount of energy. As energy consumption and runtime deviate in this case, forcing stalls instead of predictions could optimize energy consumption without impacting runtime.

5.3 Case Studies

In this section, we discuss interesting results for specific mitigations. We rerun the benchmarks for these mitigations while tracking performance counters. We track stalled cycles, dTLB-load and dTLB-store misses, iTLB-load and iTLB-store misses, LLC-load and LLC-store misses, L1-dcache loads and stores, branch loads and misses, instructions executed, mispredicted branches retired, uOPs issued, and stalled cycles due to misprediction recovery. We use these performance counters to determine possible reasons for our results. For all results discussed, we provide the average overhead in percent, the standard error of the mean $\sigma_{\bar{x}}$ and the sample size n. To improve readability, numbers that increase with the mitigation active are colored green and numbers that decrease red.

Retbleed IBPB. The retbleed indirect branch prediction barrier (IBPB) mitigation results in the most significant performance hit. This mitigation introduces a barrier that prevents code executed before it from affecting future branches [27]. We observe overheads of 1 147 % ($\sigma_{\bar{x}} = 2.46\,\%$, $n = 54$) (runtime) and 959 % ($\sigma_{\bar{x}} = 0.62\,\%$, $n = 54$) (energy) in the case of aio, and 972 % ($\sigma_{\bar{x}} = 0.20\,\%$, $n = 54$) (runtime) and 841 % ($\sigma_{\bar{x}} = 0.15\,\%$, $n = 54$) (energy) in case of context. This is due to IBPB introducing barriers that reset the branch predictor, resulting in more stalls at conditional branches after the barrier [27].

For all benchmarks, the IBPB mitigation increases runtime. The energy overhead is almost fully in line with the runtime, with two exceptions. For apache, the energy consumption decreases by 7.1 % ($\sigma_{\bar{x}} = 0.03\,\%$, $n = 54$), despite a runtime increase. This means that with IBPB enabled, the apache benchmark consumes less energy than with the mitigation disabled despite running slightly longer. Branch misses decrease by 6.8 % ($\sigma_{\bar{x}} = 1.45\,\%$, $n = 90$) and cycles

Table 4: Energy overhead of all measured software mitigations in percent. Values are rounded to one decimal point for changes <10 % and to the next integer for larger changes. Overheads that are not statistically significant are replaced by a ∗-symbol.

Benchmark	spectre_v2=retpoline	spectre_v2=retpoline,generic	spectre_v2=retpoline,lfence	spectre_v2=eibrs	spectre_v2=eibrs,retpoline	spectre_v2=eibrs,lfence	spectre_v2=ibrs	spectre_v2=on	spec_store_bypass_disable=on	spectre_v1	pti=on	l1tf=flush	l1tf=full	mds=full	tsx_async_abort=full	retbleed=unret	retbleed=ibpb	mmio_stale_data=full	l1d_flush=on	kvm.nx_huge_pages=force
icache	3.7	4.1	*	3.1	3.7	3.9	64	4.5	*	*	15	*	*	18	17	5.6	211	17	*	*
fork	1.9	1.4	*	0.7	1	1.1	30	10	2.1	*	5.5	*	*	4.1	4.7	5.6	53	3.8	*	*
pthread	−8.8	*	*	*	*	*	23	*	−46	*	*	4.3	*	21	25	−10	313	24	*	*
context	4.6	4.3	2.8	4.7	4.5	4.4	289	4.6	0.8	*	79	*	*	117	118	8.2	841	118	*	*
pipe	19	19	11	19	20	19	69	42	1.7	*	18	0.7	0.5	18	19	10	209	18	*	*
io	6.7	7.1	4.6	8.2	6.9	8.3	24	7.1	6.8	*	3.6	*	*	6.2	2.9	6.4	41	5.1	*	*
sock	14	15	8	16	16	15	151	13	1.3	*	40	*	*	44	43	6.3	425	43	*	*
udp	25	25	12	25	24	25	98	269	1.6	*	13	*	*	17	17	14	282	16	*	*
futex	10	10	7.9	10	11	10	87	149	61	*	12	*	*	15	14	7.3	169	14	*	*
aio	16	17	7.1	16	16	16	308	16	1	*	79	*	*	92	91	19	959	92	*	*
switch	27	26	16	26	26	26	108	332	1.3	*	20	*	*	19	20	10	316	20	*	*
sctp	*	*	*	*	*	*	0.4	*	*	*	*	*	*	*	*	*	73	*	*	*
signal	4.4	3.9	2.6	3.7	3.7	3.8	192	4.5	1.8	*	49	*	*	61	61	8	600	59	*	*
cpu	*	*	*	*	*	*	2.1	0.2	5.6	*	0.4	*	*	*	*	*	4	0.2	*	*
network-loopback	13	14	6.2	13	12	12	53	109	0.9	*	16	*	*	11	10	5.9	166	10	*	*
mutex	*	*	*	*	*	*	*	*	1.5	*	*	0.3	−0.1	*	0.2	*	1.3	*	*	*
osbench files	6.1	5.4	3.1	5.6	5.6	5.7	5.5	6.1	*	*	1.6	*	*	2.2	2.4	6.3	14	3.2	*	*
osbench processes	0.8	0.3	*	0.3	0.4	0.3	0.9	*	*	*	*	*	*	*	0.2	0.7	2	*	*	*
osbench threads	*	*	−0.8	*	*	*	−0.4	*	1.1	*	−0.5	*	*	*	*	1.3	−0.9	*	*	*
osbench programs	0.4	*	*	0.2	*	*	0.9	*	0	*	*	*	*	0.7	0	0.2	1.3	0.1	*	*
osbench allocations	0.2	*	*	0.4	0.3	0.2	5.7	0.6	4.9	*	1.7	*	*	2.3	1	0.5	13	2.2	*	*
apache	0.5	0.4	0.7	0.4	0.6	0.5	−2	−0.6	*	*	*	*	*	0.8	1.1	0.4	−7.1	1.1	*	*
pmbench	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	0.6	*	*	*

Mitigation

stalled increase by 4.2 % ($\sigma_{\bar{x}} = 0.59\,\%, n = 90$). Due to IBPB, there are fewer mispredictions in the apache benchmark, counteracted by more stalls. This leads to a slight runtime increase. As stalls are more efficient than mispredictions and due to almost no runtime change, the energy consumption decreases.

Insight 4. Stalling instead of speculating difficult to predict branches can lead to significant energy savings at almost no performance cost.

Speculative Store Bypass Disable (SSBD) & Retbleed Unret. Speculative store bypass allows the CPU to speculatively execute a load after a store with potentially overlapping addresses if the CPU predicts that they do not overlap. SSBD forces the CPU to wait until the addresses of all previous stores are known before executing any following loads [27]. Executing the pthread benchmark with SSBD enabled decreases runtime by 41 % ($\sigma_{\bar{x}} = 1.97\,\%, n = 54$) and the energy consumption by 46 % ($\sigma_{\bar{x}} = 0.99\,\%, n = 54$). The instructions executed decrease by 63 % ($\sigma_{\bar{x}} = 1.08\,\%, n = 90$), while the amount of stalled cycles increase by 46 % ($\sigma_{\bar{x}} = 0.82\,\%, n = 90$). Furthermore, L1-dcache stores and loads, branch loads, branch misses, and mispredicted retired branches decrease by 60 %. The decrease in runtime and energy overhead seem to stem from fewer instructions executed and the decrease in mispredictions and cache misses when spawning a large number of threads. The benchmark seems to regularly trigger speculative store bypass, leading to mispredictions. Enabling SSBD, therefore, reduces mispredictions and improves performance for this benchmark. The retbleed unret mitigation behaves similar to SSBD for Stress-NG pthread with an energy and runtime decrease of 10 % ($\sigma_{\bar{x}} = 0.85\,\%, n = 54$) while instructions executed, and retired mispredicted branches decreased by 11 %.

Spectre V1. Contrary to all other mitigations that target code tested, we did not observe any overheads for the spectre_v1 mitigation. To rule out a problem with our setup, we validated that the mitigation is correctly applied by debugging the kernel. While unexpected, this result is not unreasonable as the spectre_v1 mitigations only consist of lfence and swapgs barriers for selected user-copy functions, as well as explicit pointer sanitation, on a case-by-case basis. These defenses are lightweight and might not be encountered a significant number of times by our benchmarks.

Insight 5. Some mitigations can be lightweight enough to not result in a statistically significant overhead, making it reasonable to always enabled them.

Spectre V2 IBRS. Indirect branch restricted speculation (IBRS) introduces a bit in the IA32_SPEC_CTRL MSR. When this bit is set after a switch to a higher privilege level, e.g., user mode to kernel mode, branches executed in the higher privilege mode can not be controlled by software executed in the lower privilege mode. IBRS is extremely expensive, which is why some CPUs received a more optimized version called enhanced IBRS (eIBRS) [27]. Contrary to the previously discussed IBPB retbleed mitigation, IBRS takes effect only when switching to a higher privilege level. The IBRS mitigation increases the runtime of apache by 0.14 % ($\sigma_{\bar{x}} = 0.01\,\%, n = 54$) and decreases energy consumption by 2.2 % ($\sigma_{\bar{x}} = 0.034\,\%, n = 54$). The cycles stalled increase by 28.4 % ($\sigma_{\bar{x}}$ = 0.47 %, n = 166), presumably due to an increase in branch misses by 7.4 % ($\sigma_{\bar{x}}$ = 0.62 %, n = 166). Despite this and the low runtime change, instructions executed increased by 11.7 % ($\sigma_{\bar{x}}$ = 0.82 %, n = 166). LLC-store misses decreased by 15.3 % ($\sigma_{\bar{x}}$ = 3.93 %, n = 166), L1-dcache loads by 8.0 % ($\sigma_{\bar{x}}$ = 1.37 %, n = 166), and L1-dcache stores by 3.6 % ($\sigma_{\bar{x}}$ = 1.40 %, n = 166). The increase in branch misses and instructions executed seems to be counteracted by the drastic decrease in cache misses, resulting in only a minor runtime change. We conclude from these results that the almost unchanged runtime, combined with the increase in stalls, resulted in this decrease in energy consumption.

PTI. Page table isolation (PTI) unmaps the whole kernel except for trampoline code and data structures, which always have to be accessible while in user space. While this mitigation was initially proposed to protect against a wide range of side-channel attacks that break kernel address space layout randomization (KASLR) [20], it was introduced into the kernel as a software mitigation against the Meltdown attack [19, 44]. PTI has the largest runtime and energy overheads with 87 % ($\sigma_{\bar{x}}$ = 0.06 %, n = 54) and 79 % ($\sigma_{\bar{x}}$ = 1.06 %, n = 54) respectively when running the `context` Stress-NG benchmark. The `context` benchmark continuously performs user-level context switches. While the user can manage the contexts and trigger context switches to them, each context switch triggers a syscall and, therefore, a switch to kernel mode. This results in a high number of switches between kernel and user mode, which is the worst case scenario for PTI. The stalled cycles increase by 91.8 % ($\sigma_{\bar{x}}$ = 14.71 %, n = 43), the dTLB-store misses by 67.2 % ($\sigma_{\bar{x}}$ = 8.07 %, n = 43), the iTLB-load misses by 35.3 % ($\sigma_{\bar{x}}$ = 6.90 %, n = 43), the branch misses by 102.7 % ($\sigma_{\bar{x}}$ = 3.04 %, n = 43), uOPs issued by 431.2 % ($\sigma_{\bar{x}}$ = 47.8 %, n = 43). The increased pressure on the TLB is the result of the regular unmapping and remapping of the kernel memory. Benchmarks that do not induce a significant amount of privilege changes are only marginally affected by PTI. The `cpu` Stress-NG benchmark, in particular, which performs calculations purely in user space, has an energy consumption increase of 0.4 % ($\sigma_{\bar{x}}$ = 0.12 %, n = 54) and no statistically significant runtime overhead. Contrary to the expected behavior, the OSBench `threads` benchmark from the Phoronix test suit has a runtime increase of 0.28 % ($\sigma_{\bar{x}}$ = 0.01 %, n = 54) and an energy consumption decrease of 0.5 % ($\sigma_{\bar{x}}$ = 0.13 %, n = 54). None of our tracked performance counters indicate why the energy consumption decreased for this benchmark, but the result is consistent even after multiple reruns.

MDS & TAA. Microarchitectural data sampling (MDS) and TSX async abort (TAA) mitigations behave almost identically. According to the documentation, Linux uses the same mechanism to mitigate both MDS and TAA [31], which is also suggested by Intel in their advisory [28]. The slight differences in some of the measurements for the two can be attributed to noise and the low sample size.

6 Discussion of Limitations & Robustness

In this Section, we discuss the limitations to our work and the robustness of our results. While there are limitations to our work, we discuss why they do not affect our main insights.

Setup. CVEs affect specific systems in virtualized or native environments. In particular, we only focused on Linux kernels in a virtual machine. There may be vulnerabilities that affect only kernels in virtual machines or only kernels outside of virtual machines (home computer scenario), or other kernels than Linux. Future work has to determine whether the relation between performance and energy is similar for scenarios and kernels we did not study.

We measure the energy consumption with RAPL of the whole CPU package on a single system for comparability of the measurements. The measurements, thus, include the energy consumption of host programs and the hypervisor. The host and hypervisor are the same for all test runs and, hence, add a noisy baseline to the measurements but the absolute overheads stays the same. To compensate for the noise, we perform >60 measurements.

Patch Commit Choice. There is no generic way to find all CVE fix commits. We benchmark only the commit that marks the CVE as fixed. This strategy is sufficient for smaller and, therefore, most CVEs, as they are usually fixed by one commit. Larger CVEs might have multiple commits with other commits in between. For major vulnerabilities, such as Meltdown and Spectre, that were under embargo, the fix commit might only change the names of defines and does not directly fix the vulnerability. We cannot test these fixes using our automatic approach. To incorporate mitigations to major hardware vulnerabilities into our evaluation, we individually benchmark each mitigation that can be activated and deactivated through command line options on the recent Linux 6.2 kernel.

Prefiltering. The benchmark prefiltering (Section 4.1) is fundamental to efficiently test a wide range of CVEs on a wide range of benchmarks in a short amount of time. It allows us to run benchmarks only on CVEs that use the affected code. Using breakpoints, while completely automatic, can lead to false positives and false negatives, e.g., if the code change is in a preprocessor macro. Furthermore, our prefiltering does not account for changes in the binary layout due to the fix. This may affect performance and energy consumption due to caches and other CPU internal buffers and optimizations. Accounting for such changes is infeasible as they can also occur from a compiler version or build system change and can be negated entirely by unrelated code changes.

Measurement Accuracy. Systematic energy analyses require accurate energy measurements from software, e.g., RAPL. However, in response to attacks via RAPL [43], Intel limited RAPL's accuracy in certain cases [29]. For this work, the mitigation is inactive, and unfiltered energy measurements are reported by RAPL.

Benchmark Choice. While we chose a broad set of benchmarks, covering a wide range of the Linux kernel's functionality, our results are inherently limited by our selection. This is an inherent issue with benchmarking. Therefore, it is not possible to determine the energy or runtime overhead of CVEs on kernel code parts that are not covered by our benchmarks, e.g., CVEs that affect other architectures, not used drivers, not tested kernel interfaces.

Hardware Choice & Measurement Interface. In this work, we execute all our measurements on an i7-6700K. Depending on factors such as microarchitecture and core frequency, these results can differ on other CPUs, limiting their general applicability. Despite this, our experiments can show a general trend between energy overhead and runtime overhead, as well as unexpected results such as energy and runtime overhead differing from each other. Furthermore, the framework described in this work can be easily applied to kernel versions running on a wide range of CPUs.

For our experiments, we use the Intel RAPL interface. As RAPL only measures CPU energy consumption, our determined overheads are CPU energy overheads and do not include other parts of the system, e.g., disk and network card. Therefore, additional overheads induced by a patch or mitigation outside of the CPU are not covered by our results. Most of these additional overheads would most likely be due to changes in runtime, as most CVE patches and mitigations target software or CPU hardware vulnerabilities and not devices. Additionally, while our experiments are executed on an Intel CPU relying on Intel RAPL, this interface can be replaced by other available measurement interfaces, e.g., AMD RAPL [2] or even smart plugs that can cover the whole system energy consumption, making our framework applicable to a wide range of CPUs.

7 Discussion & Related Work

Our work highlights a fundamental problem in the security research community: Energy costs of security fixes and mitigations are systematically not measured and thus not understood. In the systems community, measuring energy costs has already become more common, as energy costs are highly relevant to assessing the value of newly proposed mechanisms (cf. Section 3.2). The two most closely related works are by Herzog et al. [24] and Siavvas et al. [60]. Herzog et al. [24] specifically focused on mitigations against Meltdown [44] and Spectre [37]. They made the surprising observation that for KPTI [19], some benchmarks show different overheads for energy than for performance, motivating our more comprehensive study of 1 616 Linux CVE fixes and all Linux security mitigations.

Performance and energy costs of mitigations reach far beyond the Linux kernel, and future work needs to investigate in both directions: On the level of applications and libraries, mitigations can have a substantial impact on performance and energy consumption, e.g., the Chrome site-isolation patch already had a reported overhead of 8.2 % on the CPU usage [57]. Similarly, on the hardware side, there are substantial costs attached to mitigations. Examples include the initial patches against Rowhammer that doubled the refresh rate. While the performance cost of the double refresh rate is reported as around 8 % [35, 40], it can be expected that the energy cost is higher as the performance is only affected when a DRAM access is deferred due to a refresh operation occurring exactly at the same time. However, the cost to charge and cells always applies. Another example of the hidden cost of security measures is Intel's mitigation to the Plundervolt attack on SGX [51] was to disable the corresponding DVFS interface via a microcode update, which was already only accessible with kernel privileges. As a consequence, system-specific optimization of voltage and frequency is largely impossible on updated machines and on more recent processors. Juffinger et al. [30] report efficiency gains of up to 20 % from voltage-frequency optimizations that are now made impossible due to the disabled DVFS interface. Thus, we see room for future work in many directions following up on our work.

While RAPL and similar interfaces can be used to profile a CPU's energy consumption for profiling [22, 34, 55], they are also actively used in a wide range of attacks. The first works exploring the security aspects of Intel RAPL focused on container co-location detection [16] and branch side-channel attacks [15]. Mantel et al. [49] demonstrated a side-channel attack distinguishing RSA keys. More recently, Lipp et al. [43] showed that RAPL-based energy measurements are precise enough to mount power analysis attacks [6, 38] purely from software. Surprisingly, their work showed that even the values of data operands can have a RAPL-measurable effect on the energy consumption. Subsequently, Wang et al. [63] showed that the energy-budget-induced throttling even creates remotely measurable timing differences. Liu et al. [45] demonstrated that the power side channel signal is also contained in the frequency due to the same throttling due to dynamic voltage and frequency scaling (DVFS). Kogler et al. [39] showed that specific workloads can amplify the leakage to speed up software-based power analysis attacks significantly. Qin et al. [56] and Yan et al. [64] used interfaces on mobile devices that report direct or indirect information on the power consumption (voltage, current, battery charge) and demonstrated website and application fingerprinting attacks. O'Flynn [52] exploited an onboard analog-to-digital converter to recover secrets processed in the secure world on a TrustZone-enabled device.

8 Conclusion

In this paper, we presented the first systematic analysis of the energy costs of CVE fixes and mitigations with the Linux kernel as a case study. We evaluated Linux kernel CVE fixes starting from Linux 4.0 over an 8-year time frame, covering 1 616 CVEs that we can automatically map to patch sets present in the source-code versioning repository. We automatically compiled Linux pre- and post-patch and benchmarked them using the Stress-NG and Phoronix benchmark suits. Overall, our work confirms benchmark-affecting code changes for 108 Linux CVE fixes, for which we collected energy and performance data. We also benchmarked all flag-controlled mitigations. While energy and performance cost is largely correlated, there are notable exceptions where energy and performance costs diverge significantly. It is important to note that this is **not** a cost-benefit analysis of CVE patches and whether they should be applied. Instead, our work underscores the need for future security research to evaluate both performance and energy cost.

Acknowledgments

This research is supported in part by the European Research Council (ERC project FSSec 101076409), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 539710462 ("DOSS"), 502228341 ("Memento") and 465958100 ("NEON"), the Bundesministerium für Bildung and Forschung (BMBF, Federal Ministry of Education and Research) in Germany (project SUSTAINET-inNOvAte 16KIS2262), and the Austrian Science Fund (FWF project NeRAM 10.55776/I6054). Additional funding was provided by generous gifts from Red Hat, and Intel. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

[1] Akinlolu Adekotujo, Adedoyin Odumabo, Ademola Adedokun, and Olukayode Aiyeniko. 2020. A Comparative Study of Operating Systems: Case of Windows, UNIX, Linux, Mac, Android and iOS. *International Journal of Computer Applications* 176, 39 (2020), 16–23.

[2] Advanced Micro Devices Inc. 2019. *AMD uProf User Guide* (3.2 ed.). Advanced Micro Devices Inc.

[3] Nadav Amit, Fred Jacobs, and Michael Wei. 2019. Jumpswitches: restoring the performance of indirect branches in the era of spectre. In *USENIX ATC*.

[4] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2021. CURE: A Security Architecture with CUstomizable and Resilient Enclaves. In *USENIX Security*. 1073–1090.

[5] Lucy Bowen and Chris Lupo. 2020. The Performance Cost of Software-based Security Mitigations. In *International Conference on Performance Engineering*. 210–217. https://doi.org/10.1145/3358960.3379139

[6] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation Power Analysis with a Leakage Model. In *CHES*.

[7] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2019. A Systematic Evaluation of Transient Execution Attacks and Defenses. In *USENIX Security*.

[8] Eugenio Capra, Chiara Francalanci, and Sandra A Slaughter. 2012. Measuring application software energy efficiency. *IT Professional* 14, 2 (2012), 54–61.

[9] Alexis Challande. 2022. Towards 1-day Vulnerability Detection using Semantic Patch Signatures.

[10] Shaiful Alam Chowdhury and Abram Hindle. 2016. Greenoracle: Estimating software energy consumption with energy measurement corpora. In *Mining Software Repositories (MSR)*. IEEE.

[11] Jonathan Corbet. 2014. BPF: the universal in-kernel virtual machine. https://lwn.net/Articles/599755/

[12] Jonathan Corbet. 2019. Many uses for Core scheduling. https://lwn.net/Articles/799454/

[13] Victor Duta, Cristiano Giuffrida, Herbert Bos, and Erik van der Kouwe. 2021. PIBE: Practical kernel control-flow hardening with profile-guided indirect branch elimination. In *Architectural Support for Programming Languages and Operating Systems*. 743–757. https://doi.org/10.1145/3445814.3446740

[14] Christian Eichler, Jonas Röckl, Benedikt Jung, Ralph Schlenk, Tilo Müller, and Timo Hönig. 2024. Profiling with trust: system monitoring from trusted execution environments. *Design Automation for Embedded Systems* (2024). https://doi.org/10.1007/s10617-024-09283-1

[15] Matteo Fusi. 2017. Information-Leakage Analysis Based on Hardware Performance Counters.

[16] Xing Gao, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. 2017. ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds. In *DSN*.

[17] Alexander Gilgur, Brian Coutinho, Iyswarya Narayanan, and Parth Malani. 2021. Transitive Power Modeling for Improving Resource Efficiency in a Hyperscale Datacenter. In *Companion Proceedings of the Web Conference*.

[18] Corey Gough, Ian Steiner, Winston Saunders, Corey Gough, Ian Steiner, and Winston Saunders. 2015. CPU Power Management. *Energy Efficient Servers: Blueprints for Data Center Optimization* (2015).

[19] Daniel Gruss, Dave Hansen, and Brendan Gregg. 2018. Kernel Isolation: From an Academic Idea to an Efficient Patch for Every Computer. *USENIX ;login* (2018).

[20] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. 2017. KASLR is Dead: Long Live KASLR. In *ESSoS*.

[21] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. 2015. An energy efficiency feature survey of the intel haswell processor. In *International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*.

[22] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. 2012. Measuring Energy Consumption for Short Code Paths Using RAPL. *ACM SIGMETRICS Performance Evaluation Review* 40 (2012), 13–17.

[23] Ashif S Harji, Peter A Buhr, and Tim Brecht. 2013. Our troubles with Linux kernel upgrades and why you should care. *ACM SIGOPS Operating Systems Review* 47, 2 (2013), 66–72.

[24] Benedict Herzog, Stefan Reif, Julian Preis, Wolfgang Schröder-Preikschat, and Timo Hönig. 2021. The Price of Meltdown and Spectre: Energy Overhead of Mitigations at Operating System Level. In *EuroSys*.

[25] Joel Hruska. 2019. Intel Performance Hit 5x Harder Than AMD After Spectre, Meltdown Patches. https://www.extremetech.com/computing/291649-intel-performance-amd-spectre-meltdown-mds-patches

[26] Zhichao Hua, Dong Du, Yubin Xia, Haibo Chen, and Binyu Zang. 2018. EPTI: efficient defence against meltdown attack for unpatched VMs. In *USENIX ATC*.

[27] Intel. 2018. Speculative Execution Side Channel Mitigations. Revision 3.0.

[28] Intel. 2019. Intel® Transactional Synchronization Extensions (Intel® TSX) Asynchronous Abort / CVE-2019-11135 / INTEL-SA-00270. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/intel-tsx-asynchronous-abort.html

[29] Intel. 2022. Running Average Power Limit Energy Reporting. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html

[30] Jonas Juffinger, Stepan Kalinin, Daniel Gruss, and Frank Mueller. 2024. SUIT: Secure Undervolting with Instruction Traps. In *ASPLOS*.

[31] The Linux Kernel. 2023. The kernel's command-line parameters. https://www.kernel.org/doc/html/v6.2/admin-guide/kernel-parameters.html

[32] The Linux Kernel. 2024. Linux CVE-Announce Mailing List. https://lore.kernel.org/linux-cve-announce/

[33] kernel.org. 2023. The kernel's command-line parameters. https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html

[34] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K. Nurminen, and Zhonghong Ou. 2018. RAPL in Action: Experiences in Using RAPL for Power Measurements. *ToMPECS* 3 (2018), 1–26.

[35] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In *ISCA*.

[36] Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, and Gernot Heiser. 2014. Comprehensive Formal Verification of an OS Microkernel. *ACM Transactions on Computer Systems* 32, 1 (2014), 2:1–2:70. https://doi.org/10.1145/2560537

[37] 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 *S&P*.

[38] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *CRYPTO*.

[39] Andreas Kogler, Jonas Juffinger, Lukas Giner, Lukas Gerlach, Martin Schwarzl, Michael Schwarz, Daniel Gruss, and Stefan Mangard. 2023. Collide+Power: Leaking Inaccessible Data with Software-based Power Side Channels. In *USENIX Security*.

[40] Changmin Lee, Wonjae Shin, Dae Jeong Kim, Yongjun Yu, Sung-Joon Kim, Taekyeong Ko, Deokho Seo, Jongmin Park, Kwanghee Lee, Seongho Choi, Namhyung Kim, Vishak G, Arun George, Vishwas V, Donghun Lee, Kangwoo Choi, Changbin Song, Dohan Kim, Insu Choi, Ilgyu Jung, Yong Ho Song, and Jinman Han. 2020. NVDIMM-C: A Byte-Addressable Non-Volatile Memory Module for Compatibility with Standard DDR Memory Interfaces. In *HPCA*.

[41] Hugo Lefeuvre, Vlad-Andrei Bădoiu, Alexander Jung, Stefan Lucian Teodorescu, Sebastian Rauch, Felipe Huici, Costin Raiciu, and Pierre Olivier. 2022. FlexOS: Towards Flexible OS Isolation. In *Architectural Support for Programming Languages and Operating Systems*.

[42] Frank Li and Vern Paxson. 2017. A Large-Scale Empirical Study of Security Patches. In *Conference on Computer and Communications Security*. 2201–2215. https://doi.org/10.1145/3133956.3134072

[43] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. 2021. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *S&P*.

[44] 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 *USENIX Security*.

[45] Chen Liu, Abhishek Chakraborty, Nikhil Chawla, and Neer Roggel. 2022. Frequency throttling side-channel attack. In *CCS*.

[46] Michail Loukeris. 2019. Efficient computing in a safe environment. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1208–1210. https://doi.org/10.1145/3338906.3342491

[47] Nicholas Luedtke. 2023. Linux Kernel CVEs. https://www.linuxkernelcves.com

[48] Teng Ma, Shanpei Chen, Yihao Wu, Erwei Deng, Zhuo Song, Quan Chen, and Minyi Guo. 2023. Efficient Scheduler Live Update for Linux Kernel with Modularization. In *Architectural Support for Programming Languages and Operating Systems*. 194–207. https://doi.org/10.1145/3582016.3582054

[49] Heiko Mantel, Johannes Schickel, Alexandra Weber, and Friedrich Weber. 2018. How Secure is Green IT? The Case of Software-Based Energy Side Channels. In *ESORICS*.

[50] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP laboratories* 27 (2009), 28.

[51] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *S&P*.

[52] Colin O'Flynn and Alex Dewar. 2019. On-Device Power Analysis Across Hardware Security Domains. In *CHES*.

[53] Jacob Pan. 2013. RAPL (Running Average Power Limit) driver. https://lwn.net/Articles/545745/

[54] Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy efficiency across programming languages: how do energy, time, and memory relate?. In *ACM SLE*.

[55] James Phung, Young Choon Lee, and Albert Y Zomaya. 2018. Modeling System-Level Power Consumption Profiles Using RAPL. In *NCA*. IEEE.

[56] Yi Qin and Chuan Yue. 2018. Website Fingerprinting by Power Estimation Based Side-Channel Attacks on Android 7. In *TrustCom/BigDataSE*.

[57] Charles Reis, Alexander Moshchuk, and Nasko Oskov. 2019. Site Isolation: Process Separation for Web Sites within the Browser. In *USENIX Security*.

[58] Xiang (Jenny) Ren, Kirk Rodrigues, Luyuan Chen, Camilo Vega, Michael Stumm, and Ding Yuan. 2019. An analysis of performance evolution of Linux's core operations. In *Symposium on Operating Systems Principles*. 554–569. https://doi.org/10.1145/3341301.3359640

[59] Alireza Shameli-Sendi. 2021. Understanding Linux kernel vulnerabilities. *Journal of Computer Virology and Hacking Techniques* 17, 4 (2021), 265–278.

[60] Miltiadis Siavvas, Charalampos Marantos, Lazaros Papadopoulos, Dionysios Kehagias, Dimitrios Soudris, and Dimitrios Tzovaras. 2019. On the relationship between software security and energy consumption. In *China-Europe International Symposium on Software Engineering Education*.

[61] Xin Tan, Minghui Zhou, and Brian Fitzgerald. 2020. Scaling open source communities: an empirical study of the linux kernel. In *Conference on Software Engineering*.

[62] The Linux Kernel. 2024. Dealing with bugs: CVEs. https://docs.kernel.org/process/cve.html

[63] Yingchen Wang, Riccardo Paccagnella, Elizabeth He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. 2022. Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86. In *USENIX Security*.

[64] Lin Yan, Yao Guo, Xiangqun Chen, and Hong Mei. 2015. A Study on Power Side Channels on Mobile Devices. In *Symposium on Internetware*.