POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

An automatic framework to detect transient execution vulnerabilities in modern CPUs through an architecture-agnostic analysis

Laurea Magistrale in Computer Science and Engineering – Ingegneria Informatica

Author: Luca Bancale

Advisor: Prof. Luca Cassano

Co-advisor: Dr. Elia Lazzeri

Academic year: 2025–2026

1. Introduction

Modern high-performance processors sustain throughput through aggressive microarchitectural optimisations including out-of-order execution, deep cache hierarchies, branch prediction, and speculative execution. These mechanisms allow the processor to execute instructions along uncommitted paths, creating a fundamental divergence between the ISA defined architectural state presented to software and the physical state of the processor.

The first real-world exploits of transient execution attacks (TEA), namely Spectre [4] and Meltdown [5] disclosed in January 2018, demonstrated that this divergence constitutes a measurable attack surface. By exploiting the microarchitectural state left behind after an architectural rollback, an attacker can exfiltrate sensitive data bypassing isolation boundaries like process-level separation, kernel memory isolation, computing enclaves such as Intel SGX or even virtual machine boundaries.

Since TEAs arise from intended hardware behaviour rather than software bugs, proper fixes require silicon redesigns and a complete replacement of the installed processor base, a process measured in years. In the interim, vendors deployed software countermeasures as a temporary stopgap. These proved inadequate on two fronts. First, they imposed heavy performance penalties, up to 72% for subsystems with frequent operating system interaction [3], translating directly into increased power consumption at datacentre scale. Second, they were fundamentally reactive: Retbleed bypassed the widely deployed Retpoline defence four years after its introduction [7], and the iterative discovery of new variants has made every mitigation a temporary fix rather than a permanent one.

Since vendor-reported patch status is an unreliable proxy for actual hardware security [2], independent evaluation of the underlying silicon is needed. Yet the existing approaches for such evaluation each have critical shortcomings. Architecture-aware verification relies on Register-Transfer Level specifications, commonly unavailable for commercial off-the-shelf processors. Post-silicon proof-of-concept replay is constrained to the space of known attacks. The iterative disclosure record confirms this is a permanent limitation, not an engineering shortfall. Hardware Performance Counter monitoring detects attack activity at runtime but is evadable by design [6]. No existing approach simul-

taneously offers accessibility to independent analysts, empirical grounding in fabricated hardware, and a continuous exploitability measure rather than a binary verdict.
This thesis addresses this gap through two contributions described in the following sections. The first is TEA-checker [1]. The second is a novel continuous security metric that translates validated timing differentials into structured per-attack-family assessments of processor resilience, replacing the binary *vulnerable/patched* verdict of vendor advisories with a quantitative exploitability measure.

2. The Transient Window

The mechanism underlying all transient execution attacks is the transient window opened by speculative execution. Speculative execution lets the processor process instructions before hazards are resolved. Instructions issued along this unconfirmed path, called transient instructions, consume execution resources and update microarchitectural state but carry no guarantee of ever retiring. The interval between the dispatch of the first transient instruction and the detection of a hazard is the transient execution window. When the hazard is resolved the architectural state is rolled back, but microarchitectural state including cache lines, predictor entries, and buffer occupancy remains altered. A timing side channel can then detect this residual state, providing the observable timing differential that both transient execution attacks and the proposed framework exploit.

2.1. Timing Side Channels

Many microarchitectural features create a characteristic timing differential between its active and inactive states. Cache hierarchies produce the largest, with a cache completing in approximately 4 cycles while a main-memory access requires approximately 200 cycles, yielding a 50× gap. The Flush+Reload technique [8] exploits this differential to detect cache state with an empirical error rate below 5%, and it serves as the covert channel through which transient execution effects become externally observable.

2.2. Classes of Transient Window

Two classes of mechanism open the speculative window. Spectre-type attacks poison a predictor structure such as the Pattern History Table, Branch Target Buffer, or Return Stack Buffer, causing the processor to speculatively access attacker-controlled memory. Meltdown-type attacks instead exploit a delayed permission check that temporarily permits speculative access to memory at a higher privilege level. In both cases, secret data is encoded into the cache state before the architectural rollback occurs and is subsequently recovered through the timing channel.

3. TEA-Checker Framework

TEA-checker is an architecture-agnostic, software-only testing framework that infers the presence and properties of transient execution effects purely from timing observations on unmodified silicon, without hardware debug access or proprietary specifications.
This evaluation is performed through two complementary test categories. *Feature tests* probe a single microarchitectural structure in isolation, establishing presence and parameters. *System interaction tests* orchestrate multiple features to measure whether a speculative window opens and whether a secret-dependent access crosses a security boundary.

3.1. Framework Architecture

The framework is organised around an *orchestrator* that manages test execution, dependency resolution, and result serialisation. Since microarchitectural measurements require instructions unavailable at user privilege, a *probe module* in the form of a Linux loadable kernel module that exposes TLB and kernel memory manipulation primitives. Tests execute through privilege-aware *runners* that determine whether measurement code runs in user space at Ring 3 with probe-module access, directly in kernel space at Ring 0, or in bare-metal simulation for cross-ISA targets such as RISC-V. The runner abstraction isolates the measurement logic from the execution context, so the same measurement code can run at any privilege level without modification.
Each test is implemented as a module that encapsulates two components. The *microbenchmark* implements the tight timing loop and is the component executed by the runner. The *manager* handles the execution lifecycle, first-pass statistical evaluation, and iterative conver-

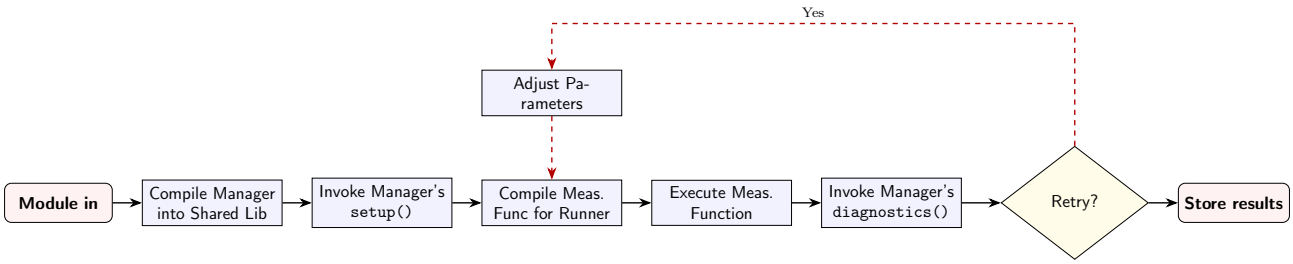

Figure 1: Module execution lifecycle

gence protocols such as binary search for ROB capacity estimation. This separation ensures the same benchmarking code can be executed as a user-space shared library, an executable, a kernel module, or a bare-metal binary.
The module lifecycle is visualised in Figure 1. The manager is compiled into a shared library, then `setup()` performs pre-processing. The measurement function is compiled for the target runner and executed on a pinned core. The manager's `diagnostics()` performs a first-pass evaluation. If the result has not converged, it returns `RETRY`, parameters are adjusted, and the measurement function is recompiled. Once accepted, results are serialised to a binary file.

3.2. Vulnerability Scoring Model

The scoring model translates validated timing differentials into structured security assessments at three levels of abstraction.

Feature severity. For each feature test, the validated timing differential is normalised into a continuous severity score:

$$S_f = \frac{|t_{\text{without}} - t_{\text{with}}|}{|t_{\text{without}}| + |t_{\text{with}}|} \in [0, 1] \qquad (1)$$

where t_{without} and t_{with} are the measured latencies in the absence and presence of the mechanism, respectively. A score of zero indicates the feature is absent or indistinguishable from noise. A score near one indicates a clearly detectable component that contributes to the speculative attack surface. For capacity-based features such as ROB, LFB, and TLB the score is derived from the detected capacity relative to the expected hardware range, because a larger buffer allows for a longer transient window.

Attack-family score. Per-feature scores are aggregated into an exploitability score for each of the supported attack families:

$$S_a = \text{avg}_{f \in \text{req}(A)}(S_f) \cdot S_{\text{cache}} \cdot \prod_{m \in \text{mult}(A)} m \qquad (2)$$

where $\text{req}(A)$ is the set of feature components the attack family requires, and $\text{mult}(A)$ is a set of context multipliers. For example, the presence of SMT doubles the score of MDS-family attacks because shared physical buffers provide a cross-hyperthread observation channel.

Privilege-level aggregation. System interaction tests record the highest security boundary at which a speculative load produces a detectable signal. The ordering appears in Equation 3:

$$\texttt{excluded} < \texttt{not\_detected} < \texttt{same\_space} < \texttt{user} < \texttt{kernel} \qquad (3)$$

A result of `kernel` indicates that speculative loads can leak kernel-memory data to an unprivileged observer, representing the highest severity breach. At the lower end, `not_detected` means no boundary crossing was observed, while `same_space` indicates leakage within the same address space.

3.3. Validation Methodology

Validating a timing-only framework on COTS hardware presents a fundamental challenge. The processor's internal state is inaccessible, so no oracle exists to verify a measured S_f is attributable to the claimed mechanism rather than to noise, prefetching, or power-state transitions. The validation strategy is *suppression-and-collapse*. Each test is compiled in two configurations: a standard variant that leaves the claimed mechanism active and a suppressed variant that disables it. If S_f collapses to near-zero under suppression, the measurement is validated. If it

persists, it is discarded. The check is applied at three levels: at the feature level where δ collapses, at the interaction level where no window appears when prerequisites are suppressed, and at the attack-family level where the aggregate score drops to near-zero.

3.4. Test Suite

The framework implements fourteen feature tests targeting the microarchitectural structures described in Section 2. Each test isolates a single hardware mechanism and measures whether it produces a detectable timing differential. Table 1 lists each test alongside the mechanism it probes and the software suppression used to validate the measurement via the differential collapse. On top of these, nine system interaction tests orchestrate the detected features to model attack scenarios. Table 2 lists each interaction test, the transient execution scenario it represents, and its corresponding validation suppression. Feature tests are prerequisites for interaction tests: a dependency directed acyclic graph formalises this relationship and the orchestrator resolves it before scheduling execution.

Table 1: Feature test suite.

Feature	Measures	Mitigation
Cache	L1/DRAM gap	Flush every load
PHT	Branch predictor	Unpredictable branches
BTB	Indirect jump cache	Random indir. calls
LAP	Stride predictor	Random strides
RSB	Ret. stack depth	Fill RSB before call
ROB	ROB capacity	Serialise each inst
TLB	Trans. cache reach	Flush TLB bef. access
LFB	Fill-buffer cap.	Serialise par. loads
OoO	Out-of-order exec	Barrier between ops
Pipeline	Superscalar width	Serialise between ops
STL	Store-to-load fwd	Serialise store/load
SMT	Simult. multithr.	Diff. physical core
SGX	Intel SGX support	CPUID flag check
SIMD	SIMD unit presence	Compile-time macro

Table 2: System interaction test suite.

Interactions	Scenario	Mitigation
Spec	Mispredict window	Correct training
OoO	Delayed-except window	Barrier bef. spec. load
BTI	BTB poison	Random indir. calls
STL	STL leak	Serialise store/load
RSB	RSB underflow	Fill RSB before ret.
Stale	Stale micro-op exec	Barrier after code mod
TLB	Stale TLB trans.	TLB flush after PTE mod
LFB	Cross-thread LFB	Serialise parallel fills
LAP	LAP stride bypass	Random strides

4. Experimental Evaluation

The framework was evaluated on three COTS x86-64 processors, an AMD Ryzen 9 7900X (Zen 4, 2022), an Intel Core Ultra 7 (Meteor Lake, 2023), and an Intel Core i5-8250U (Kaby Lake R, 2017). The first two incorporate the hardware-level mitigations introduced in response to the 2018 disclosures. The i5-8250U predates those disclosures entirely and serves as the unmitigated baseline. The orchestrator coordinated all execution from user space via the probe module's ioctl interface, with each test pinned to a fixed physical core on an otherwise idle system.

4.1. Architectural Feature Detection

Three questions were investigated: (i) can the framework detect speculative execution primitives on commercial processors, or do vendor mitigation flags conceal the hardware that remains present? (ii) can it differentiate between independently designed microarchitectures, or are the measured differences merely artefacts? (iii) can it detect primitives whose exploitability depends on subtle timing effects, which is the hardest case for a timing-based approach?

Before interpreting any per-platform score, the reproducibility of the measurements must be established. The BTB test depends on how the code layout interacts with the hardware's set-associative mapping and the OS memory layout, the RSB test alternates between near-zero readings and near-maximum readings 98.2% without any change to the underlying hardware. This wide run-to-run variation confirms that no single execution can be treated as the processor's definitive state. TEA-checker therefore reports both the central tendency and the dispersion as first-class outputs, since a processor whose score oscillates between extremes presents a fundamentally different evaluation target than one that produces a stable value.

All scores were validated through suppression-and-collapse (Sec. 3.3). The Cache test drops from 97.2% to 3.8%. The PHT test drops from 38.1% to 0.1%. The ROB test drops from 100% to 0% when out-of-order execution is constrained. The stale code interaction test drops from 46.4% to 0.0%. All four follow the same pattern: a clear timing differential collapses to near-zero when the claimed mechanism is sup-

pressed. With this foundation, the severity scores rest on verified measurements. Table 3 reports the continuous-output tests and the aggregate detection score.

Table 3: Feature severity scores (S_f) across the three platforms.

Feature	Platform		
	AMD 7900X	i5-8250U	Ultra 7
Cache	95.9%	93.1%	97.8%
PHT	30.5%	44.2%	39.4%
LFB	48.0%	24.0%	56.0%
TLB	16.3%	73.5%	59.9%
BTB	11.1%	21.2%	12.5%
STL	35.8%	87.8%	88.7%
RSB	24.8%	10.5%	31.6%
ROB†	100%	100%	100%
SIMD†	100%	100%	100%
Pipeline†	100%	100%	100%
SMT†	100%	0%	100%
SGX†	0%	100%	0%
OoO†	100%	100%	100%
LAP	0%	0%	0%
Aggregate (14 tests)	**54.5%**	**61.0%**	**63.3%**

(i) The aggregate Feature Detection scores confirm that all three platforms expose a broad and measurable speculative surface. The Core Ultra 7 scores highest despite being the most heavily mitigated system, which illustrates a central point of the methodology. The score measures the breadth of detectable features, and a richer feature set produces a higher aggregate regardless of whether individual primitives are mitigated.
(ii) The TLB and STL tests reveal a consistent vendor split, with both structures scoring substantially higher on Intel than on AMD across both Intel generations. Both tests collapse correctly under mitigation across all three platforms, so these differences reflect genuine design divergence rather than a measurement artefact. This is information that no vendor disclosure provides.
(iii) Branch predictor structures such as the BTB, PHT, and RSB fall into this category because the timing difference between a predictor hit and a predictor miss is smaller than the difference between a cache hit and a cache miss. All three tests produce moderate severity scores, demonstrating that the framework registers these primitives despite the inherently low observable differential. Doing so without hardware documentation and purely through user-space timing validates the architecture-agnostic approach directly.
The framework detects and differentiates speculative primitives across three commercially available processors from two vendors, using only user-space timing measurements and no privileged hardware information. Feature presence alone, however, does not determine whether a primitive can be weaponised across privilege boundaries. The scores in Table 3 measure what is present on the silicon; whether those primitives can be made to leak data across security boundaries is the subject of the following section.

4.2. Security Boundary Analysis

Three questions were investigated: (i) can the framework detect cross-boundary speculative leakage on these platforms, or does feature presence alone determine exploitability? (ii) can it identify which specific primitives are responsible for each boundary crossing and how far they reach, moving beyond a simple binary verdict? (iii) does the framework's unified score reveal structure that the disaggregated results alone cannot express, exposing information about mitigation effectiveness that the component scores do not show? Nine system interaction tests were executed under three privilege-level variants: process at Ring 3 in the same address space, user at Ring 3 cross-process, and kernel where a Ring 3 attacker targets Ring 0 via the probe module. Table 4 reports the highest privilege boundary crossed per test, and all interaction tests collapsed to near-zero under mitigation, validating the reported boundaries (Sec. 3.3).

Table 4: System interaction test scores and privilege-boundary crossings.

Test	AMD 7900X	i5-8250U	Ultra 7
Spec Window	0% safe	32.5% same	45.2% same
OoO Window	0% safe	32.5% safe	20.6% safe
BTI Window	0% safe	6.0% kernel	1.2% same
STL Window	0% safe	47.7% kernel	10.2% user
RSB Window	0% safe	100% kernel	28.4% safe
Stale Code	0% safe	32.5% kernel	45.2% safe
TLB Window	0% safe	4.1% safe	0.6% safe
LFB Window	0% safe	82.7% safe	7.1% safe
LAP Window	0% safe	0% safe	0% safe
Overview (9 tests)	**0.0% safe**	**37.6% kernel**	**17.6% user**

(i) The AMD Ryzen 9 7900X crosses no boundary in any interaction test, while both Intel plat-

forms leak across multiple privilege levels. This divergence is the central result. Feature presence alone does not determine exploitability, and the framework captures that distinction.
(ii) The interaction tests produce a range of signals across the two Intel platforms, with some primitives reaching kernel space on the older platform while the same primitives are contained to user or same space on the newer one. These results show that the framework distinguishes not only whether leakage occurs but which mechanism drives it and how deeply it penetrates. The comparison also reveals a further capability: both Intel platforms exhibit similar feature breadth, yet the newer model reduces the reach of each primitive without removing the underlying mechanism. The framework therefore separates hardware capability from mitigation effectiveness, a distinction that vendor flags collapse into a single verdict.
(iii) The Feature Detection scores alone give each platform a similar rating, leaving the differences in boundary containment invisible. The aggregate exposes this hidden dimension. The two Intel platforms diverge based on how effectively their mitigations contain the same underlying speculative structures, a separation that neither the feature detection scores nor the interaction tests alone provide.
These scores measure the presence and strength of detectable microarchitectural features and boundary crossings, not real-world exploitability. A higher score indicates a richer attack surface, but turning this surface into a working exploit requires engineering effort the framework does not attempt to quantify, hence the scores are best read as a relative comparison across the three evaluated platforms rather than as an absolute measure of risk.

4.3. Attack-Family Scoring

Two questions were investigated: (i) does the framework ever contradict a vendor-confirmed vulnerability, which would be a direct falsification of its correctness? (ii) what do the seven discrepancies reveal about the evaluation gap between binary vendor flags and continuous hardware capability? Per-family scores S_a were computed from Equation 2 and cross-referenced against vendor-reported vulnerability status. Table 5 summarises the aggregated vulnerability scores combining both feature detection and speculative overview results.

Table 5: Aggregated vulnerability scores across all 23 tests per platform.

	AMD 7900X	**i5-8250U**	**Ultra 7**
Feature Det. (14 tests)	54.5%	61.0%	63.3%
Spec. Overview (9 tests)	0.0%	37.6%	17.6%
Aggregated Score	**33.1%**	**51.6%**	**45.4%**

(i) The outcome is unambiguous. Zero false negatives occur on any platform. Every attack family that a vendor flags as vulnerable receives a nonzero score from the framework, indicating that the framework correctly models the mechanisms these families target.
(ii) In each discrepancy, the vendor reports a platform as not vulnerable while the framework assigns a nonzero score. These are not false positives. The discrepancies are the evaluation gap that the framework is designed to characterise. The framework's continuous scores expose residual attack surface that binary vendor flags cannot represent. That the discrepancies cluster on AMD and Intel Core Ultra 7, the two platforms with the most extensive mitigation deployment, is consistent with this interpretation. These are precisely the platforms where a binary mitigated flag is most likely to conceal remaining hardware capability.

4.4. Cross-Platform Portability

An architecture-agnostic methodology must work across platforms without per-vendor calibration. Portability within a single ISA is a necessary condition before we can pursue cross-ISA portability. To test this, we executed the identical test suite on all three platforms, running each feature test both unmitigated and with its corresponding software mitigation active, applying the validation methodology described in Section 3.3 uniformly. Of the fourteen feature tests, thirteen collapsed correctly under mitigation on all three platforms.
The ROB feature test is the exception, and it exposes a limitation in the portability of the detection algorithm itself. On AMD Ryzen 9 7900X and Intel Core i5-8250U the mitigated reading remains at 100%, identical to the unmitigated value, while the same test collapses correctly to 0% on Intel Core Ultra 7. The fault lies in the

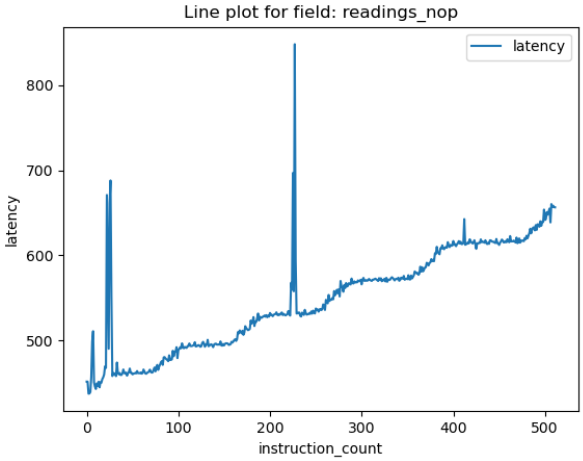

Figure 3: ROB readings with mitigations.

detection function rather than in the mitigation itself: the function does not generalise correctly across different execution environments during mitigated runs, and the sudden spikes visible in the traces (Figures 2 and 3) are attributable to operating system scheduling interference rather than to the ROB mechanism. The mitigation does suppress the timing jump that normally indicates ROB capacity, as Figure 3 shows, but the detection algorithm fails to register this suppression on two of the three platforms.

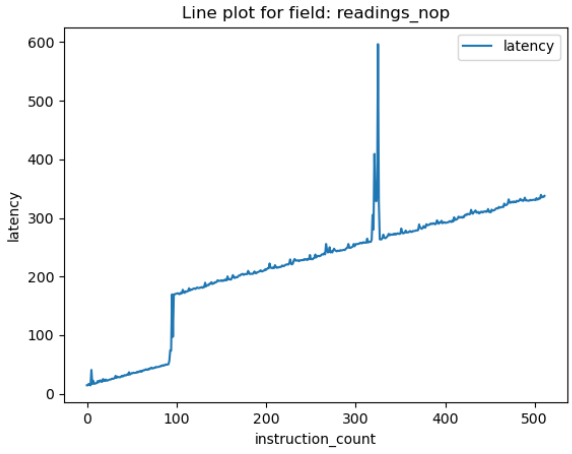

Figure 2: ROB readings under normal load.

This exception highlights the portability limit. Detection thresholds calibrated on one platform cannot be blindly transferred to another. The remaining thirteen tests validate correctly across all three platforms, demonstrating that the framework's measurement principle is portable even if the signal-processing parameters require per-platform tuning.

5. Conclusions

This thesis demonstrates that architecture-agnostic timing measurement can characterise transient execution behaviour on COTS processors without access to proprietary hardware designs. The data collected across three physical platforms points to three findings in particular.

Speculative behaviour detection. Across every case where an independent ground truth was available, TEA-checker's feature and interaction tests detected the corresponding primitives. The structurally paired timing differential, validated against its own software mitigation, correctly isolates the microarchitectural mechanisms it targets.

Exploitability quantified through window measurement. The system interaction tests measure the actual size of the transient execution window in instruction count, scaled through the platform's measured execution rate. This provides a more fine grained security evaluation than simple binary classification through vendor flags.

Attack-family independent severity. The generic boundary-breach test probes for any speculative access that crosses an isolation boundary, regardless of the specific triggering mechanism, giving the framework a way to assign severity even to undiscovered attack vectors.

This thesis makes two contributions. The first is the architecture-agnostic testing framework itself, constituted by the orchestrator, the privilege-aware execution runners, the dependency-resolved module system, and the validation methodology. The second is the continuous security model, a scoring scheme that moves from a single timing differential through a per-feature severity score to an exploitability measure and a boundary-crossing assessment without collapsing information into a binary verdict.

Two limitations qualify these results. The ROB test collapsed under mitigation only on the Core Ultra 7, exposing detection-threshold portability limits. The feature set was shaped by vulnerabilities available on the development processor and does not encompass the full TEA landscape.

Future work should broaden the module set, extend RISC-V to physical hardware, deploy the framework on ARM, and replace fixed detection thresholds with adaptive pipelines that cancel platform-specific noise.

References

[1] Luca Bancale. TEA-checker. `https://github.com/sbancuz/TEA-checker`, 2026.

[2] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. A systematic evaluation of transient execution attacks and defenses, 2019.

[3] Benedikt Herzog, Hans P. Reiser, and Rüdiger Kapitza. The price of meltdown and spectre: Energy overhead of mitigations at operating system level. In *Proceedings of the 14th European Workshop on Systems Security (EuroSec '21)*, pages 8–14, 2021.

[4] 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. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2019.

[5] 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. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[6] Minkyu Song, Taeweon Suh, and Gunjae Koo. Vizard: Passing over profiling-based detection by manipulating performance counters. *IEEE Access*, 11:48099–48112, 2023.

[7] Johannes Wikner and Kaveh Razavi. RETBLEED: Arbitrary speculative code execution with return instructions. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2787–2804. USENIX Association, 2022.

[8] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732. USENIX Association, 2014.