PowSpectre: Powering Up Speculation Attacks with TSX-based Replay

Md Hafizul Islam Chowdhuryy
University of Central Florida
Orlando, Florida, USA
hafizul.islam@ucf.edu

Zhenkai Zhang
Clemson University
Clemson, South Carolina, USA
zhenkai@clemson.edu

Fan Yao
University of Central Florida
Orlando, Florida, USA
fan.yao@ucf.edu

ABSTRACT

Trusted execution environment (TEE) offers data protection against malicious system software. However, the TEE (e.g., Intel SGX) threat model exacerbates information leakage as attackers can enhance and denoise the observations from hardware-based side channels through controlled victim execution (i.e., replay). The replay mechanism is especially critical for side channels from physical traces (e.g., power consumption) that not only vary instantaneously but also necessitate successive modulation for observability. In this paper, we identify and characterize the key limitations of existing replay techniques for speculation attacks. Our study unveils that architectural support for transactional memory (i.e., Intel TSX) can be leveraged as a highly efficient replay primitive for transient execution. Based on this observation, we design TMPlayer, an efficient and high-resolution TSX-based replay framework for enclave victims. Built on top of TMPlayer, we present PowSpectre- a novel replay-based transient execution attack using software-based power side channels (via RAPL) that can exfiltrate secretive enclave data accurately in the speculative domain. We evaluate PowSpectre using case studies on several representative SGX binaries. Our evaluation shows that PowSpectre can exfiltrate unintended secrets in enclaves with very high accuracy. We perform a gadget analysis in SGX libraries and identify widely-existing code patterns that are power differentiable for PowSpectre. Our work highlights the need to synergistically understand the impact of speculation security with the introduction of new hardware functionalities.

CCS CONCEPTS

• **Security and privacy → Trusted computing**.

KEYWORDS

Power side channel, Trusted execution environment, Transaction memory, Speculative execution, Replay attack

ACM Reference Format:
Md Hafizul Islam Chowdhuryy, Zhenkai Zhang, and Fan Yao. 2024. **PowSpectre**: Powering Up Speculation Attacks with TSX-based Replay. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3634737.3661139

ASIA CCS '24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07
https://doi.org/10.1145/3634737.3661139

1 INTRODUCTION

Hardware-based side channels have raised unprecedented security concerns in computing systems. These attacks exfiltrate sensitive information in software through monitoring hardware resource usage due to secret-dependent program executions [34, 35, 61, 66, 95]. Particularly, these side channels can be manifested by modulating either: i) hardware-maintained states (i.e., *persistent states*) in numerous processor microarchitectural components (e.g., caches [61, 66, 90, 95], branch predictors [8, 33, 34], and TLBs [37]) or ii) *implicit* and ephemeral states including occupation on shared structures (e.g., interference [9, 13, 74]) as well as physical properties (e.g., power and EM [8, 57, 97]) associated with hardware activities. The development of transient execution attacks further empowers such side channel threats [10, 16, 24, 25, 51, 58, 71].

As the threat of advanced attackers in malicious environments continues to escalate, integrating hardware-based trusted execution environments (TEEs) has become imperative. Pioneered by Intel's Software Guard Extensions (SGX), TEEs take the processor as the foundation of trust, providing isolated execution for applications running within enclaves [30]. This offers confidentiality and integrity protection for enclave data, even in the case of compromised system software (i.e., malicious OS). However, numerous studies have revealed that the security guarantees of SGX are not always upheld, with the system being vulnerable to various side channels [55, 58, 68, 71, 75, 78, 80–82]. SGX exacerbates information leakage by allowing the presence of powerful, privileged adversaries with system-level control capabilities (e.g., interrupt handling). SGX attackers can utilize fine-grained execution control to build highly accurate side channels through victim execution within the enclave. This is particularly useful for stateless side channels (e.g., contention status and physical properties) due to their susceptibility to inherent noise [13, 57, 75].

Equipped with the capability of privileged system-level control by adversaries, prior studies have proposed *replay mechanisms* that allow multiple (and potentially unbounded) re-executions of certain secret-operating instructions (or instruction sequences) while stalling the forward progress of the victim [75, 79]. As side channels can be sensitive to interference, these techniques are extremely useful to denoise side channels against an enclave. So far, two classes of replay techniques have been studied: i) exploiting frequent timer interrupts to trap the enclave in the endless *exit/resume loop* (i.e., zero-stepping [79]); or ii) utilizing page faults to enforce repetitive executions from certain fault-inducing instructions (e.g., MicroScope [75]). While such mechanisms can successfully augment

classical microarchitectural attacks, the practicality of *replaying transient execution attacks* is not well explored. Our investigation reveals that existing approaches fall short of such a purpose. Specifically, we observe the existence of a *serialization effect* of ERESUME (i.e., the instruction to resume into the enclave) that zero-stepping anchored on for re-execution. As a result, it only achieves *execution rewind* and cannot transiently execute instructions after ERESUME[1]. In contrast, we find that page fault-based replay involves complex page walk operations, which can introduce non-deterministic perturbations to side channel measurements (e.g., power consumption), thereby restricting the accuracy of secret inference.

Based on the above observations, we investigate novel instruction replay capabilities exploiting hardware transactional memory, specifically using Intel TSX [40]. With the low-overhead hardware tracking of memory accesses during transactions and the provision of fast retry mechanisms through a userspace abort handler, TSX allows instructions to be repeatedly executed in the event of an abort, effectively functioning as an instruction replay primitive. Ironically, prior studies have suggested using TSX to safeguard enclaves from privileged exploitation, such as page fault-based attacks [21, 43], positioning it as a potential countermeasure for replay-based side channels [75]. Furthermore, when considering both effectiveness and implementation cost, a TSX-based approach is potentially one of the most promising defenses against cache attacks [21, 38]. However, our investigation reveals that TSX not only fails to defeat replay attacks but can also be harnessed to strengthen transient instruction replay, essentially transforming the tool into a weapon. Unlike prior attacks that utilize TSX to manipulate/monitor cache states [32, 82] and fundamentally rely on the cache being vulnerable, we exploit the intended nature of TSX as a replay primitive.

In this work, we present PowSpectre- a novel TSX-based speculative execution replay framework that exfiltrates enclave secrets using power side channels via the Intel Running Average Power Limit Interface (RAPL) [6]. To the best of our knowledge, PowSpectre is the first attack that leverages *instruction-based power leakage* to amplify power observation for speculation leakage, which can manifest even with the *most recent version* of hardened RAPL [5, 57]. Importantly, with the exploitation of *inter-instruction power differentiability*, the proposed attack manages to perform fine-grained bit stealing using highly distinguishable power observations even under Intel's most recent microcode update with *hardened RAPL* [1, 3], which can defeat state-of-the-art power-based leakage attacks [57]. To overcome the shortcomings of prior instruction replay mechanisms, we design **TMPlayer**–a TSX-based mechanism that replays instructions for transient execution with a high degree of precision. We implement two variants of TMPlayer: 1) *TMPlayer-E* that utilizes conflicts on the readset/writeset to trigger transaction aborts, and 2) *TMPlayer-I* which exploits in-transaction interrupt for efficient TSX-based replay. Notably, the new replay primitive can make *practical* noise-prone power side channels, which cannot be easily annulled via microarchitecture-level countermeasures [21, 38, 43, 65]. We investigate the efficacy of our proposed schemes on two generations of Intel processors (i.e., Skylake and Coffee Lake). The evaluation shows that TMPlayer can achieve 100% replay with extremely high resolution (i.e., up to 45K replays/second). Additionally, PowSpectre is able to infer enclave secret bits in transient execution with >95% accuracy under both TMPlayer variants. We perform two case studies of the proposed attack against the Intel SGX SSL library. Our results show that PowSpectre can exfiltrate cryptographic keys with accuracy up to 94%. Finally, we identify power-differentiable gadgets that can be leveraged by PowSpectre. Our findings reveal widely existing exploitable code gadgets for PowSpectre in representative SGX libraries (including SSL and LibC). In summary, the main contributions of this paper are:

- We investigate state-of-the-art instruction replay mechanisms and unveil their pitfalls as well as limitations for replaying speculation attacks. We then identify Intel TSX as the new replay primitive for transient execution and design TMPlayer, a high accuracy and low overhead replay attack framework for enclave victims.
- Using TMPlayer, we design PowSpectre, a novel replay-based power side channel that can accurately exfiltrate enclave secretive data accessed in the speculative domain. Notably, we show the first attack that can manifest under Intel's most recent microcode update mitigating prior RAPL-based information leakage.
- We demonstrate real-world attack capabilities of PowSpectre through i) leaking cryptographic key in RSA, and ii) stealing private plaintext in envelope-mode encryption from Intel SGX SSL library. PowSpectre achieves high bit-stealing accuracy with a modest number of samples.
- We analyze power-differentiable PowSpectre gadgets in representative SGX libraries and categorize the identified gadgets based on their bit-leaking accuracy. Our analysis shows the existence of abundant gadgets exploitable for PowSpectre to leak speculation data.
- We discuss potential mitigation strategies for PowSpectre. Our work highlights the need to rethink the security implications as hardware functionalities are utilized in processors for protection.

Responsible Disclosure. Following the practice of responsible disclosure, we have shared our findings with the product security team of Intel.

2 BACKGROUND

2.1 Hardware-based Side Channels

Microarchitectural security. Program execution leads to changes of microarchitectural state changes. Such side effects, if dependent on program secrets, introduce information leakage observable through side channels (e.g., timing). Microarchitectural attacks have been demonstrated on many hardware components such as cache [32, 61, 91, 94, 95], branch predictor [24, 25, 34], frontend buffer [69], execution port [13] and memory [22, 23, 54, 67]. To mitigate them, various microarchitecture-level protections are presented that either *isolate/limit the access* to shared resources (i.e., transmit) [2, 15, 26, 50, 60, 76] or *disrupt the observation* (i.e., receive) [8, 14, 62, 66] through such a side channel. These defenses can be either *always on*, or can be selectively enabled on demand through periodic contention monitoring [20, 44, 89, 92, 93].

[1]In other words, zero-stepping only replays ERESUME, which is sufficient in prior attacks [71, 78] as they target enclave registers loaded on chip by it.

To protect commercial-off-the-shelf machines, system-level techniques such as constant-time programming that eliminate secret-dependent data-/control-flow are proposed [18, 22, 23, 34, 36, 64], which typically requires software rewrite by programmers.

Speculation attacks. Speculative execution [51, 58] transforms microarchitecture side channels to more severe information leakage. Specifically, exploitation of transient instruction execution can lead to *unintended data access* in the speculative domain, which could then be leaked through a microarchitectural side channel [51, 58, 87]. Existing system-level defenses for speculation-based exploitations generally attempt to prevent the malicious trigger of mis-speculation (e.g., branch poisoning) [45, 46] or stop speculation altogether (e.g., for security-sensitive branches) [47, 77]. Note that these techniques do not offer complete security as they either introduce non-trivial overhead or can be potentially bypassed with more advanced attack techniques [10].

Physical side channels. Processor hardware activities can influence physical states such as power, EM emission, and acoustic signals [42, 49, 52, 73]. Generally, this leads to side channel exploits leaking coarse-grained secrets (e.g., website fingerprinting) [41, 52, 72, 73, 96]. Recently, software interfaces exposing power measurements have shown to make power side channels exploitable remotely [56, 57, 97]. Specifically, Intel and AMD systems provide a software interface to monitor *running average power limit* (RAPL) for different domains. Prior work reveals that it is possible to infer *instruction operand* based on power observations via the RAPL interface [57]. In response, the system community has released countermeasures that block userspace access to the power measurements. Additionally, Intel has released microcode patches [5] to downgrade the power reporting to a model-based approach [3], which claims to defeat state-of-the-art RAPL side channels.

2.2 Trusted Execution Environment

TEE provide data security for user-defined regions of application called *enclaves* through hardware-enforced encryption and attestation. SGX encrypts enclave data using on-chip hardware and stores it in off-chip Enclave Page Cache (EPC), protecting it from malicious OS/hypervisor and physical attacks. SGX provides a set of instructions and SDK functions for enclave operations, including i) `EENTER`/`EEXIT` to enter/exit an enclave; ii) `ECALL`/`OCALL` to call trusted/untrusted functions from outside and inside the enclave; iii) `ERESUME` to restore the enclave after a context switch (e.g., kernel space switch to handle exceptions/interrupts). To protect architectural registers holding enclave secrets at context switches, the enclave performs an Asynchronous Enclave Exit (AEX) that securely stores its contexts, including all architectural registers and Enclave Instruction Pointer (ERIP), in an EPC region, called State Save Area (SSA). During the enclave resumption (ERESUME), this SSA is restored, and the enclave execution is resumed from ERIP.

2.3 Intel Transactional Memory Extension

Transactional memory is a high-performance alternative to explicit locks (i.e., `MUTEX`) for concurrent accesses to shared memory in parallel computing. It executes a group of memory operations as one transaction, where all or none of the operations in a transaction are completed, thereby maintaining global visibility of the operations. Intel TSX is a hardware implementation of transactional memory which tracks transactionally accessed cache lines (i.e., readset) or modified cache lines (i.e., writeset) in hardware (i.e., extension of cache coherence). If a violation of transactional property is detected, TSX provides a userspace abort handler to retry the transaction or use a fallback mechanism. If no abort is detected during the course of the transaction, all updates to the writeset are committed, marking the completion of the transaction. Interestingly, by utilizing the hardware tracking of readset/writeset in TSX, a line of defense is introduced against SGX attacks [21, 38, 43, 65]. Specifically, such defenses use TSX to i) ensure enclave data always stays in cache during execution, hiding secret-dependent data-flow [38, 43]; and ii) suppress page faults to prevent controlled-channel attacks [21, 43] and replay [43, 75].

3 THREAT MODEL

Our threat model is consistent with recent attacks that exploit vulnerabilities in Intel SGX and speculative execution techniques. This threat model aligns with pre-existing works exploiting Intel SGX [57, 75, 78, 79, 81, 83]. Despite ongoing efforts to mitigate speculative execution attacks, these exploits remain feasible, as demonstrated by recent research studies [10]. To safeguard data in SGX environments, the victim process is running within a secure enclave. We assume that TSX is available in the system. Note that, although Intel disabled TSX due to TAA, it can be enabled at user discretion through boot flags in supported platforms. Moreover, we note that hardware transactional memory (TM) is a critical architectural feature that offers high performance software parallelism. We envision that such support will be provided in certain forms in the future. It is important to understand the implication of transactional memory for microarchitecture security, a problem that severely concerns both academia and industry.

We further assume the attacker can obtain power measurements of the processor on the victim machine. While current Linux-based systems no longer offer userspace access to RAPL interfaces through `sysfs`, power consumption can still be read by a privileged attacker. Recent microcode updates from Intel can obfuscate power observations through RAPL to mitigate side channels through power [5]. This patch downgrades power reporting to a model-based approach in systems when SGX is enabled. In this study, we will investigate the practicality of power side channels where systems are equipped with the original accurate vanilla RAPL interface as well as under the hardened RAPL with the most recent microcode update.

4 UNDERSTANDING EXISTING REPLAY MECHANISMS

Instruction replay is a technique with which the attacker obtains precise control of victim execution and executes the same instruction or a sequence of instructions (i.e., target instructions) consecutively without requiring the victim's re-execution. This technique typically coordinates externally generated system events (such as interrupt or exception) with the victim execution so that target instructions are executed, but the architectural state of the program counter (PC) register is not incremented. Consequently, when the victim execution is resumed after servicing the interrupt or exception, it will execute according to the program counter, thereby

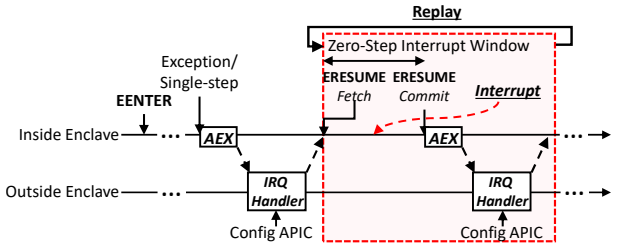


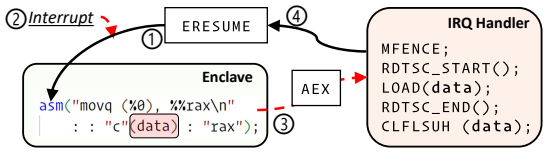
Figure 1: Timeline for timer interrupt-based replay.

re-executing the target instruction. By repeating the same procedure, attackers can achieve successive executions of selected victim instructions without committing [57, 75, 79]. In this section, we systematically investigate the two state-of-the-art replay mechanisms and demonstrate their limitations in achieving the speculative side channel through power.

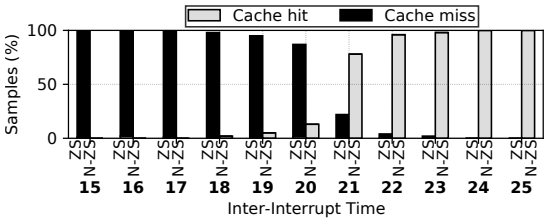
4.1 Timer Interrupt-based Replay

One possible way to realize replay is to exploit interrupts, as shown in works such as zero-stepping [79]. When the processor receives an interrupt, the CPU core typically completes the instruction at the head of the reorder buffer (ROB), squashes other in-flight instructions, and transfers control flow to the interrupt handler. In enclave execution, an AEX is performed before handling the interrupt. After interrupt processing, the enclave resumes execution by executing ERESUME. Previous works highlight two main zero-stepping use cases: 1) trapping enclave execution at the end of ERESUME to load enclave registers (secrets) in the SSA for later stealing [71, 78, 81]; 2) repetitively executing ERESUME and one or more subsequent instructions, aiming for side effects (e.g., power) leading to a collective side channel [57]. For successful replay in either case, it is crucial to ensure that for each round: ❶ ERESUME is executed, and ❷ only ERESUME is committed. Figure 1 depicts the enclave execution timeline and the interrupt arrival window required for successful replay. Notably, the timer interrupt must arrive between ERESUME being at the head of the ROB and its retirement. Due to processor pipeline uncertainties, zero-stepping faces replay failures: if the timer interrupt arrives too early, the processor may trap in an interrupt handling loop without executing subsequent instructions, including ERESUME; conversely, if the timer interrupt interval is too large, the execution window may extend beyond ERESUME's retirement, leading to execution/commit of subsequent instructions, in which case ERIP advances and zero-stepping fails.

To investigate the impact of timer-based interrupts on instruction replay, we design a microbenchmark (Figure 2a) using the zero-stepping framework [79]. The target instruction is a movq instruction loading data from memory, scheduled to execute after ERESUME. The APIC timer triggers just before the movq execution. We adjust the interrupt handler to check if the memory block of data is cached and flush it if so. Enclave execution begins with AEX, followed by the interrupt handler routine. If the enclave executes the movq, it will bring data to cache. We vary interrupt intervals and execute the experiment with 100K timer interrupt triggers, logging each execution round as a sample. Each sample records



(a) Illustration of the microbenchmark under test.



(b) Execution statistics with zero-stepping.

Figure 2: Instruction replay capability with zero-stepping. In Figure 2b, ZS and N-ZS represent samples with successful and failed zero-stepping, respectively. The inter-interrupt time is set in the unit of APIC timer interval.

whether ❶ the replay is successful (i.e., ERIP does not advance) and ❷ the load instruction (first instruction after ERESUME) is executed.

Figure 2b presents statistics on zero-stepping success and load instruction execution, as the time between interrupt arrivals varies from 15 to 25 APIC timer intervals[2]. We observe that cache misses to data consistently occur in all successful zero-stepping samples. Conversely, failed zero-stepping samples exhibit cache hits, indicating load instruction execution. These results are consistent across two different CPUs (i.e., i7-6700K and i7-9700K). Our evaluation reveals a previously unknown observation: **Instructions after ERESUME do not execute until it is retired**. We hypothesize this serialization is due to ERESUME restoring critical enclave registers, necessitating completion for correct execution. This has critical implications for replay capability. If the timer-interrupt is received during ERESUME's execution window (Figure 2b), upon the completion of ERESUME, all in-flight instructions are flushed before the handling of the interrupt. Thus, successful zero-stepping cannot advance instruction execution flow over ERESUME speculatively. Zero-stepping solely serves as an *execution rewind* and *cannot replay instructions*. We note that Platypus [57] mentions zero-stepping could be leveraged to replay instructions for power observations. However, as expected based on our experimental findings, we are not able to reproduce the replay similar to [57] using zero-stepping with timer-interrupts.

4.2 Exception-based Replay using Page Faults

Prior works have demonstrated that a privileged attacker can deliberately set page faults to induce replay [75, 79]. For example, in MicroScope [75], the attacker clears the present bit of the victim data page, causing a page fault when a targeted instruction (I_e) in the enclave accesses it. Instructions following I_e may execute transiently in a manner similar to Meltdown [58]. Once I_e is at the

[2]An APIC timer interval is the value by which the timer counter must change before an APIC interrupt is generated.

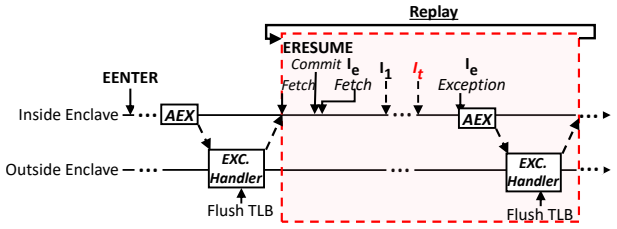


Figure 3: Timeline for page fault-based replay.

head of ROB, the page fault is detected, and all instructions in the pipeline are discarded (including I_e). An AEX is then performed to save the enclave context, and the processor execution transitions to exception handling. The attacker can tamper with the page table to clear the present bit again and flush the TLB. Upon restoration with ERESUME, the same instruction will induce a page fault exception again, leading to repeated execution of instructions following I_e.

Figure 3 outlines the timeline for replay using page faults in the enclave. Notably, the exception occurs precisely at the execution of the enclave instruction (I_e). Unlike the timer interrupt-based mechanism, page fault-based replay allows processors to advance over ERESUME as each round of ERESUME is permitted to commit. Consequently, subsequent instructions, including I_t, can advance. While this mechanism enables speculative execution and replay of a sequence of instructions, it has several limitations: ❶ Page faults trigger TLB misses and page table walks, requiring multiple memory loads, limiting replay resolution (replay frequency), and introducing non-trivial and non-deterministic noise to an elastic timing channel (e.g., power); ❷ Page faults frequently trigger the page fault handler, potentially prompting partial page table walks and leaving a system-level footprint that can be relatively easily captured. Previous studies have shown the effectiveness of (i) non-preemptable code segments (suppressing synchronous exceptions like page faults) [43, 65] or (ii) enclave-maintained timers to detect each AEX [21], both mitigating page fault-based instruction replay.

5 EXPLOITING TSX FOR INSTRUCTION REPLAY

In this section, we explore the challenges of reusing transactional memory implementations in modern processors (i.e., Intel TSX) as instruction replay techniques. Prior instruction replay techniques are either incapable of replaying (in the case of APIC timer interrupt) or exhibit non-trivial overheads (Section 4). While TSX allows for the re-execution of transactions in case of an abort, there are several challenges that must be addressed to repurpose it as an instruction replay technique. These challenges include: i) *how can the attacker trigger TSX abort deterministically?*; and ii) *how to achieve **zero** failure by exceeding the replay window?* In this section, we will address the challenges and demonstrate how this feature can be maliciously manipulated to construct highly efficient replay attack methods for the purpose of extracting sensitive information.

Abort rules in Intel TSX. Intel TSX provides synchronization across parallel threads via transactions instead of explicit locks. A transaction represents a group of memory loads and stores, succeeding only if all accesses remain valid. To ensure this, Intel defines

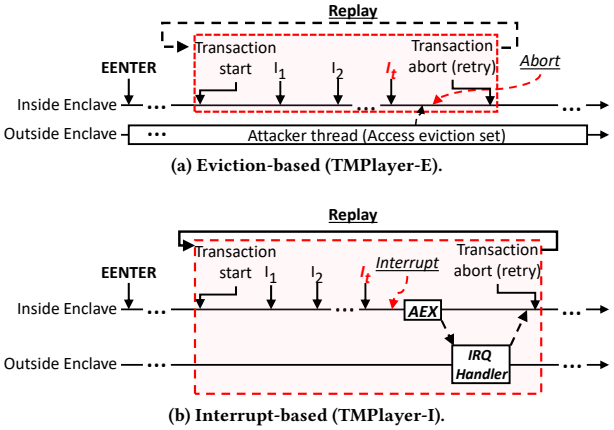


Figure 4: TMPlayer-based instruction replay.

several *abort signals* that may indicate a violation of transaction requirements: ① explicit abort (XEND instruction), ② syscall or nested transaction, ③ memory access violation/conflicts (e.g., read-after-write or write-after-write), ④ exception (i.e., page fault), ⑤ OS interrupt, and ⑥ cache eviction of readset/writeset. If any of these signals are detected during the execution of a transaction, the transaction is aborted, and an execution flow is transferred to a user-space abort handler, which typically retries the transaction.

In the event of a transaction abort, the transactional section is re-executed, effectively making the transaction hardware itself the replay agent. Signals ①, ②, and ③ are not attacker-generated, while ④, ⑤, and ⑥ can be triggered by attacker. Attackers can induce page faults for ④ (Section 4.2), configure an APIC timer for ⑤ (Section 4.1), or perform cache eviction of readset/writeset for ⑥. As inducing page faults can generate non-trivial overhead, we focus on using interrupts and cache eviction as the primary primitives for TSX-based replay. Consequently, we present two variants of our replay techniques: (a) **TMPlayer-E**, which uses cache eviction as the abort signal, and (b) **TMPlayer-I**, which uses an APIC interrupt event as the abort signal. In the following discussions, we demonstrate how the attacker can address all of the aforementioned challenges in both TMPlayer variants. Note that the availability of such transactions in the enclave can be widespread in future systems. Many SGX defenses highlight the use of TSX as a straightforward way of preventing cache and page fault-based side channels in SGX [21, 38, 43, 65]. In the presence of these defenses, the entire enclave code is wrapped inside multiple transactions, creating the wide availability of transactions in the enclave.

5.1 TMPlayer-E- Cache Eviction-based Replay Agent

Intel TSX ensures data validity during transactions via a hardware tracking mechanism, marking cache lines as readset or writeset. This involves expanding the L1 cache and last-level cache, with bits indicating transaction access to each cache line. Specifically, the TX-Read and Readset bits are set when a transaction sends a "get

message", and the `TX-Dirty` and `Writeset` bits are set when a transaction sends a "get exclusive" or "upgrade" message[39]. Tracked cache line eviction triggers transaction abort. We exploit this behavior for an effective instruction replay technique. We call this technique TMPlayer-E (Figure 4a). TMPlayer-E has three stages. ① The attacker first identifies an *abort trigger* by determining a cache line accessed during a transaction. The attacker then generates an eviction set, which is a group of cache lines that evicts other cache lines from a certain cache set when accessed in entirety. In particular, we generate the eviction set where each address in the eviction set belongs to the same L2 and LLC set following the eviction set generation method for non-inclusive cache [90]. This ensures that the attacker can perform *cache directory-based* eviction of the abort trigger without occupying the same physical core as the victim [90], and hence spatial or temporal core-sharing is not required. ② The attacker continuously accesses this eviction set from a separate physical core to evict the trigger cache line. ③ Finally, the victim enclave execution is triggered, eventually accessing the trigger cache line, and bringing it into the transactional tracked set. Thus, when the attacker evicts it due to accessing the eviction set, an abort signal will be sent to the transaction within the victim enclave, triggering subsequent re-execution of the transaction. By doing so, the attacker can deterministically trigger the abort signal (i.e., as soon as the trigger cache line is evicted) and abort the transaction without failure.

It is important to note that to avoid explicit synchronization between the attacker's eviction thread and the victim enclave execution, the chosen trigger cache line should be first accessed within the transaction after executing the target instructions (i.e., I_t). This ensures that the trigger cache line will only be a part of the transaction *after* the target instructions (I_t) have been executed. The overhead due to cache eviction is highly repetitive and stable, and can be easily filtered out from any monitored signal using a predetermined offset. Unlike previous cache side-channel attacks that use secret-dependent cache access as a secret leakage medium, TMPlayer-E exploits cache eviction as an abort signal.

5.2 TMPlayer-I - APIC Timer Interrupt-based Replay Agent

TMPlayer-I utilizes precisely configured APIC timer-based interrupts to abort ongoing transactions. Intel TSX aborts a transaction upon receiving an interrupt, forming the foundation of this technique. The key property of interrupts is that they are handled on instruction boundaries, meaning that the interrupt arrives after the retirement of the instruction that is currently at the head of the reorder buffer [80]. As a transaction comprises a set of instructions rather than a single one if the interrupt arrives after executing the target instruction (I_t) but before the transaction's completion, the transaction aborts and re-executes (Figure 4b). The key difference between TMPlayer-I and other interrupt-based replay techniques is that TMPlayer-I uses an interrupt to abort a transaction. The timing intervals between the APIC interrupts are crucial, as they must be long enough to reach the targeted instruction (I_t) but shorter than the overall execution time of the transaction. If this timing can be maintained, the transaction is guaranteed to abort without failure.

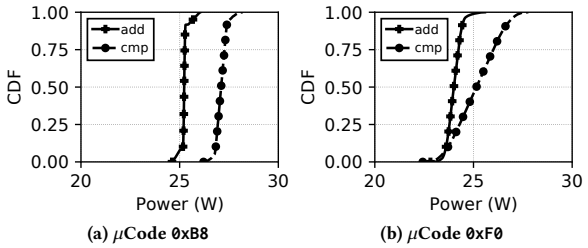


Figure 5: Cumulative distribution function (CDF) of power consumption for two different execution paths.

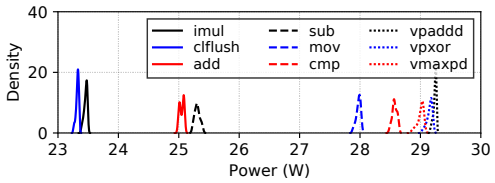
Fortunately, the execution time of a transaction is usually deterministic if all required cache lines are in the cache, which is typically the case after a few initial iterations of transactions. Therefore, the attacker first determines an optimal interrupt timing interval by executing the same instructions in the transaction from their own enclave and then configures the APIC interrupt to arrive after that interval for the duration of the attack. The overhead caused by the interrupt handling can be made negligible by configuring the periodic APIC interrupt only once [79].

6 POWER SIDE CHANNEL IN SPECULATIVE EXECUTION

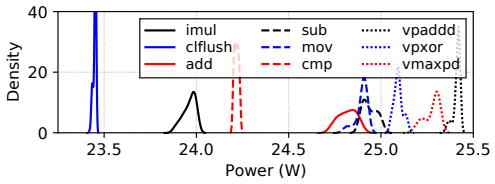
In this section, we present PowSpectre, a speculative information leakage attack that uses power side channels in SGX.

Power differentiability for instruction execution. Prior studies have shown that it is possible to distinguish *instructions* and *operands* using power side channels through RAPL. Intel has since released a microcode update [1, 3] that addresses these issues by downgrading RAPL measurements to model-based reporting in systems with SGX enabled. While it is assumed that this mitigation is sufficient to prevent previously demonstrated leakage through RAPL measurements [57], our investigation revealed that such a mechanism is not secure enough to prevent instruction differentiability. We collected 10,000 RAPL measurements (each measurement taken for the execution of 50,000 instructions inside an enclave) for running either `add` instruction or `cmp` instruction with the same operands. If everything else remains the same, these two instructions should have similar power signatures in a properly mitigated system. We ran this experiment on an Intel 9700K system (additional experiment setup can be found in Section 7). The results in Figure 5 show that even with model-based power reporting, an adversary can still observe different power signatures (shown as the CDF curves) for different instructions. This finding demonstrates that the hardened RAPL interface still allows attackers to infer the internal activities of the processor by distinguishing executions of different types of instructions or instruction sequences.

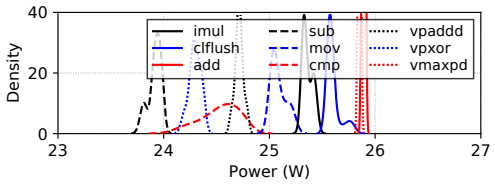
Power differentiability in the speculative domain. Power side channels are inherently coarse-grained, sensitive to system-level noise from various hardware activities. Speculative execution worsens these issues with additional branch predictor poisoning and mis-speculation overhead. To understand their impact, we conduct an experiment by repetitively executing an instruction with



(a) ***Baseline***: **no mis-speculation or enclave execution**



(b) ***In-speculate***: **with mis-speculation**



(c) ***In-enclave***: **with mis-speculation inside enclave**

Figure 6: Power signature of representative instructions in different execution environments (μCode $0xF0$).

three configurations: i) *non-speculative non-enclave* (referred to as **baseline**), ii) *speculative non-enclave* with userspace branch target poisoning (i.e., **in-speculate**), and iii) *speculative in-enclave* with kernel space branch poisoning after every execution of the target branch (i.e., **in-enclave**). Figure 6 illustrates power profiles (PKG domain) for representative instructions. We observe distinct and non-overlapping power signatures are observed for these instructions in the baseline settings (Figure 6a). Crucially, despite branch mistraining overheads, the majority of instructions exhibit distinctive power profiles under in-speculate (Figure 6b) and in-enclave settings (Figure 6c). Power differentiability across instructions persists when appropriate instructions are selected. We quantify distinguishability by calculating the accuracy of instruction detection using correlation analysis between power observations from a sliding window of 100 samples against a profiled dataset (10K samples for each instruction). Table 1 shows an average accuracy of around ~90% for instruction determination in the in-enclave environment.

6.1 Overview of PowSpectre Framework

In this section, we present the PowSpectre framework. The key insight behind PowSpectre is that the enclave's secret value in the speculative domain can be made to determine transient instruction executions that are power distinguishable. Unlike classical side channel attacks, one major challenge for the realization of PowSpectre is the need for a precise and lightweight instruction replay mechanism. This is because *a single execution of a short instruction sequence* is unnoticeable on power measurement. Additionally, since the code gadget with power differentiable paths is

Config.	imul	clflush	add	sub	mov	cmp	vpaddd	vpxor	vorpd	*Avg.*
baseline	98%	99%	96%	95%	97%	99%	95%	89%	92%	95.6%
in-spec	93%	97%	86%	85%	87%	89%	95%	96%	96%	91.6%
in-encl	94%	93%	79%	85%	89%	80%	91%	95%	91%	89.1%

Table 1: Instruction detection accuracy for each sample.

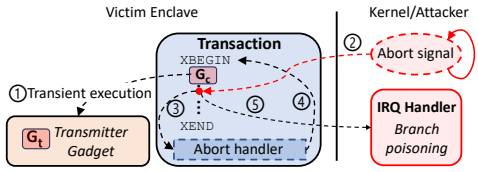


Figure 7: Overview of PowSpectre framework.

unlikely to reside within the TSX transaction code block, PowSpectre has to manipulate speculation to execute the leaky gadget in the transient path.

The high-level attack procedure is illustrated in Figure 7. As depicted, a power differentiable gadget is situated outside the scope of the TSX transaction. However, through the use of branch poisoning, the attacker redirects the control flow and executes this gadget in the transient execution path (①). After the gadget is executed, the transaction resumes, and if an abort signal is received (②), the transaction is terminated, and the abort handler is executed (③). The transaction is then re-executed from the abort handler (④). The branch poisoning is performed by the userspace IRQ handler routine, which is triggered either by an abort signal in TMPlayer-I (⑤) or via another APIC interrupt in TMPlayer-E. This poisoning is performed from the same physical core as victim execution. This process is repeated to collect the necessary amount of RAPL samples. Now we discuss the key attack steps in detail.

Step 1: Identify exploitable gadget and obtain RAPL traces for profiling. In this step, the attacker selects two gadgets: i) a control gadget (G_c) that is located within the victim's regular program execution path inside the transaction; and ii) a transmitter gadget (G_t) that contains two transient instruction execution sequences *conditioned on a speculative secret* (denoted as e_l and e_r, corresponding to ❶ and ❷). Eligible G_c gadget should include an indirect jump that, when poisoned, can transfer the speculative control flow to G_t. G_t can be any code gadget in the enclave memory with power differentiability for two different paths. After identifying the (G_c, G_t) pair in the victim process, the attacker can generate power profiles by executing the two instruction paths in the enclave. With each RAPL measurement, we compute a power sample (s) based on the energy consumption within the past interval. We collect a *trace* (T) as a vector of n consecutive power samples when certain path is repetitively replayed. T^l and T^r represents a trace for e_l and e_r, respectively. In this step, the attacker records multiple traces corresponding to each execution path and records them to generate individual power profiles (i.e., $P^l = \{T_0^l, T_1^l, ..., T_i^l\}$ for e_l and $P^r = \{T_0^r, T_1^r, ..., T_i^r\}$ for e_r).

Step 2: Set up instruction replay and branch poisoning. In this step, the attacker sets up replay primitives for G_c and G_t. Specifically, a) *for TMPlayer-E*, the attacker selects a trigger cache line that

is first accessed *within the transaction* after the indirect jump and prepares an eviction set. This ensures that the trigger cache line will be accessed after G_t has been executed in the transient path. Interestingly, we found that TSX treats any data block accessed during the transaction as part of the transaction, regardless of whether it is under a transient path. As a result, it is possible to use speculatively accessed memory blocks as the trigger cache line. However, PowSpectre does not utilize such cache lines as speculation triggers. This is because if misspeculation is not successful (due to imperfect branch poisoning), evicting of that outside cache line will *not* signal a transaction abort. Hence, the memory transaction will be committed, leading to replay failure; b) *for TMPlayer-I*, the attacker executes G_c and G_t from its own enclave while recording the execution time to determine the minimum execution latency of the transaction. The attacker then chooses a timer interval *smaller* than this latency value, ensuring that the APIC interrupt arrives within the transaction. The attacker then configures the APIC interrupt in periodic mode and selects the proper interval for the interrupt. In addition to configuring the replay primitives, the attacker also performs indirect branch poisoning to transfer control flow from G_c to the trigger gadget. In TMPlayer-I, the branch poisoning is done within the interrupt handler routine. In TMPlayer-E, an additional timer interrupt is configured (similar to TMPlayer-I).

Step 3: Trigger victim enclave execution and obtain power traces. Once all the previously mentioned attack primitives are completed, the attacker triggers the victim enclave execution and records RAPL measurements. In TMPlayer-E, the attacker at the same time executes a process that continuously accesses the eviction set. This process is continued until a sufficiently long power trace corresponding to the victim's secret-dependent path execution is obtained (T_x). It is important to note that this is the only online phase of the attack.

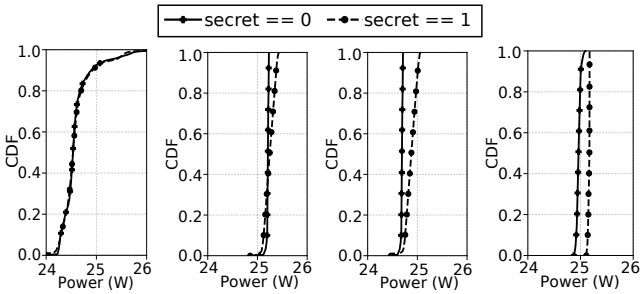
Step 4: Correlation analysis with traces from Step 1. With T_x and a power profile (P) for an execution path in G_t, the attacker derives correlation-based measure (M) between T_x and P by computing $AVG(\mathcal{R}(T_x, P_i))$, where $\mathcal{R}$ is the Pearson correlation coefficient. Essentially, the attacker runs correlation analysis between T_x and each power trace in P, and generates the average correlation coefficient. Eventually, the attacker obtains two correlations M, one with P^l and one with P^r. The M between T_x and the power profile of T_x's corresponding execution path (i.e., victim's execution) is expected to be higher due to power profile similarity. In this way, the attacker can identify which speculative path (e_l or e_r) was executed and hence successfully infer the speculative secret.

7 EVALUATION

We evaluate the effectiveness of the proposed TMPlayer and PowSpectre on Intel Core i7 9700K system running Ubuntu 20.04 and Linux kernel 5.8.0-59 with SGX SDK version 2.11. Note that collecting RAPL measurements for PowSpectre through `rdmsr` interface requires a privileged attacker. In addition, SGX model enables *privileged side channel amplification* techniques for further noise reduction [63, 86]. In particular, we utilize branch poisoning from kernel IRQ handler at a specific point of victim program execution, which requires SGX-based privileged attacker model. All experiments are carried out under two system configurations: i) without RAPL mitigation (i.e., microcode-20190918/0$xB8$) and ii) with mitigation (i.e., μCode version microcode-20220510/0$xF0$), which incorporates Intel's platform update to defend against attacks exploiting RAPL energy reporting (i.e., IPU 2021.2) [3, 4, 6]. Finally, power measurements are taken from RAPL `PKG` (package) domain.

```
1  .START:
2      cmp     $0, secret;
3      je      .ZERO; // If secret == 0, goto .ZERO
4  .ONE:   ❶
5      add     %R12, %R13; // else if secret == 1
6      ret;
7  .ZERO:  ❷
8      cmp     %R12, %R13;
9      ret;
```

Listing 1: A PowSpectre transmitter gadget leaking `secret`.

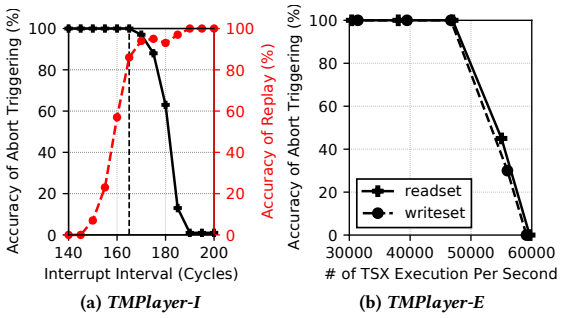


(b) *APIC Interrupt* **(c) *Page Fault*** **(d) *TMPlayer-E*** **(e) *TMPlayer-I***

Figure 8: Cumulative distribution function (CDF) of power consumption of two different execution paths corresponding to the value of secret. Non-overlapping CDF represents high accuracy of the control-flow detection.

7.1 Evaluation of TMPlayer

Instruction replay accuracy. We perform experiments to characterize the accuracy and effectiveness of TMPlayer compared to other replay mechanisms. This evaluation is performed in a non-speculative path (i.e., without the need to trigger speculation with G_c). We leverage a microbenchmark (pseudo-code shown in Listing 1) as G_t. For each instruction replay mechanism, we collect one million RAPL measurements by executing the G_t with both values of `secret` (i.e., both 0 and 1). The results (Figure 8) show the distribution of power consumption corresponding to each value of `secret`. We observe that with APIC interrupt-based instruction replay, the microbenchmark exhibits almost no power differentiability. This aligns with the findings that speculative instructions are not executed after `ERESUME` in zero-stepping. In contrast, the power profile under page fault-based instruction replay demonstrates differentiability to some extent. Specifically, this replay mechanism can detect speculatively executed instructions with 63% accuracy. We believe that page table walk introduces irregular noise patterns in the power consumption, which reduces the differentiability. Notably, both variants of TMPlayer show very high replay accuracy (i.e., 97% and 95% with TMPlayer-E and TMPlayer-I, respectively) with the help of precise and low-overhead abort trigger.



(a) *TMPlayer-I* (b) *TMPlayer-E*

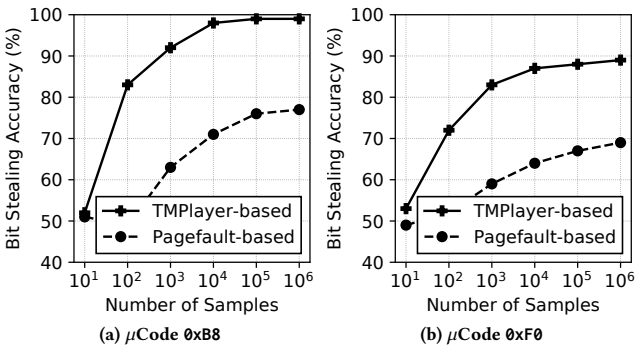
Figure 9: Accuracy of TSX abort triggering using TMPlayer.

```
1  while (true) {
2    execution_counter++; // Count transaction
3  if ((status = _xbegin ()) == _XBEGIN_STARTED) {
4      // Note: use either writeset or readset
5      // writeblk->writeset OR readblk-> readset
6      (*writeblk)++;              OR tmp &= *readblk;
7      for (size_t i = 0; i < count; i++) { }
8      _xend ();
9    } else {
10     if (status !!= 0) abort_counter++; // aborts
11   }
12 }
```

Listing 2: Code sequence to evaluate temporal resolution of TMPlayer-E.

(a) Source Code	(b) Instruction Sequence
`secret = 0 or 1;`	`// if (array[idx])`
`if (array[idx])`	`cmp [rax - 4], $0`
`// idx overshoots to secret in ↪ speculative domain`	`je .L2`
❶ `x = x + 1;`	❶ `add [rax - 8], 1`
	`...`
`else`	`.L2:`
❷ `x = x × 1;`	❷ `imul [rax - 8], 1`

Listing 3: Pseudo-code for a potential transmitter gadget.



(a) μCode `0xB8` (b) μCode `0xF0`

Figure 10: Bit stealing accuracy with various samples (results showing for TMPlayer-E).

Temporal resolution of TMPlayer. Next, we evaluate the maximum replay frequency using TMPlayer:

TMPlayer-I: To investigate the temporal effects of the timer interval in TMPlayer-I, we conducted the same experiment as before by configuring the APIC interrupt to arrive at different intervals. As discussed earlier, the TMPlayer-I interrupt needs to occur before the transaction commits and after the execution of the target gadget (G_t). From Figure 9a, we observe that in our selected gadget, PowSpectre achieves a 100% abort success rate with a relatively high (88%) instruction replay when using a fixed interval of 162 cycles for the APIC timer. This indicates that during execution, the transaction is always aborted, ensuring the complete success of replay. In addition, in the vast majority of executions, the target instruction is executed, potentially leading to distinguishable power observations through RAPL. Note that the selected timer interval heavily depends on the specific transaction or gadget being analyzed. Therefore, a one-time profiling of the transaction's runtime is necessary to determine a suitable interval.

TMPlayer-E: To determine the detection rate of abort signals in TMPlayer-E, we continuously execute the TSX code block in Listing 2 inside the enclave, utilizing either `writeblk` (or `readblk`) in Line 6. Another thread executes continuously on a different physical core, flushing `writeblk` (or `readblk`) accordingly. We record the number of times the TSX block is executed and the number of times the transaction is aborted. It is important to note that the replay frequency can be adjusted by altering the value of `count`. Figure 9b demonstrates that both readset and writeset eviction-based aborts can achieve 100% accuracy in triggering TSX aborts, with a transaction rate exceeding 46,000 transactions per second. This indicates the highly accurate instruction replay capability of TMPlayer-E, even at a very high frequency of execution. Additionally, this is very close to the maximum possible abort frequency of 49,000, which is achieved using the explicit `XABORT` instruction.

7.2 Evaluation of PowSpectre

Accuracy of bit leakage. In order to evaluate the accuracy of PowSpectre in terms of bit leakage, we perform a covert channel experiment (Listing 3). We utilize a simple control gadget (G_c) containing one indirect jump instruction and a transmitter gadget (G_t) used to leak specific bits of a given register. The process begins with the spy executing G_c and G_t in its enclave and collecting profile traces P_l and P_r. Subsequently, the spy triggers the execution of the trojan and records RAPL samples. For each bit, we collect 10,000 samples (T_x) using both variants of TMPlayer, as well as the page fault-based instruction replay. We then perform Pearson's correlation analysis [11] on sliding windows of 100 samples with the profile data (P_l and P_r). We observe that PowSpectre with TMPlayer achieves very high accuracy in classifying the enclave's transiently executed instructions, with a bit stealing accuracy of 96% in TMPlayer-E and 95% in TMPlayer-I when leaking 100 bits. In contrast, the page fault-based technique demonstrates significantly lower detection accuracy, achieving only 66%.

Sensitivity to sample size. To understand how sample size affects PowSpectre's accuracy in bit stealing, we collect 1M RAPL samples for each bit using both TMPlayer variants and page fault-based instruction replay. Results (Figure 10a) show TMPlayer can maintain high accuracy (<90%) with as few as 1,000 samples, contrasting with the page fault-based method, which only achieves

```
1  void func() {
2      __asm volatile("call %
3      "r" (*target) : ); // Indirect jump to [target]
4  }
5  *target = &benign_func; // Set target to benign
6  if (_xbegin() == _XBEGIN_STARTED) {
7      func(); // Indirect call to benign function
8      _xend();
9  } else retry(); // Retry on abort
```

Listing 4: Example TMPlayer gadget.

Attack	TMPlayer-Type	0xB8	0xF0
Case study 1	*TMPlayer-E*	93.2%	81.4%
	TMPlayer-I	92.7%	82.4%
Case study 2	*TMPlayer-E*	92.7%	83.6%
	TMPlayer-I	94.3%	82.4%

Table 2: Attack accuracy under different microcodes (i.e., 0xB8 = insecure RAPL, 0xF0 = secure model-based RAPL).

the highest accuracy of 62% with 1M samples per bit. This highlights TMPlayer's efficiency and lightweight nature. Furthermore, repeating the experiment under the latest Intel microcode update (Figure 10b) reveals that while the new microcode may impact attack accuracy, PowSpectre with TMPlayer remains highly effective, achieving 82% accuracy with 1,000 samples and 87% accuracy with 10,000 samples.

7.3 Case Studies with PowSpectre

In this section, we present two case studies of leaking cryptographic secrets from the latest version of Intel SGX SSL [29] based on OpenSSL (v1.1.0o) using PowSpectre.

```
6c590 <EC_KEY_METHOD_free>:
6c594: test [rdi+0x8], 0x1
6c598: jne 6c5a0
6c59a: ret                    ❶
6c59b: nop [rax+rax*1+0x0]
6c5a0: mov edx, 0xaa          ❷
6c5a5: lea rsi, [rip+...]
6c5ac: jmp 8ae50
...
8ae59: endbr64
8ae54: mov rax, [rip+...]
```

Figure 11: Gadget in elliptic curve key management leaking info. from [RDI].

```
d6910 <asn1_enc_init>:
...
d692a: test [rdx+0x8], 0x2
d692e: je d694d
d6930: movsxd rdx, [...]      ❶
d6934: add rax, rdx
d6937: mov [rax], 0x0
d693e: mov [rax+0x8], 0x0
d6945:
d6946: mov [rax+0x10], 0x1
d694d: ret                    ❷
```

Figure 12: Gadget in abstract syntax encoding of keys leaking info. from [RDX].

7.3.1 Stealing Cryptographic Secrets in Speculation. In the first case study, we demonstrate an attack on an enclave utilizing Intel SGX SSL to create an asymmetric key pair for the Diffie-hellman key exchange using `secp384r1` Elliptic Curve groups. The algorithm employs 384-bit prime factorization to produce the private-public key, which is then used for subsequent cryptographic operations. Listing 4 shows a representative enclave code that can act as G_c. In practice, any TSX block inside the enclave with an indirect jump or call instruction can serve as G_c. To carry out the attack, we utilize a gadget (G_t) from the OpenSSL Elliptic Curve implementation to leak the LSB of the 8^{th} byte from `[RDI]` via instruction **`test`** `[rdi+0x8], 0x1` (Figure 11). The target for the attack is the call to the `i2d_PrivateKey` function (`i2d_PrivateKey(EVP_PKEY *a, unsigned char **pp)`), which converts the generated key from OpenSSL internal format to a common digital encoding format. This prepares the `RSI` register with the address of the private key. We assume the victim enclave uses the gadget in Listing 4 that executes an indirect jump to `i2d_PrivateKey` function (i.e., in place of Line 2) to perform the private key conversion. In our gadget analysis, we have not found any appropriate gadget capable of directly leaking bits from `RSI`. As a result, we use a different gadget to copy `RSI` to another callee-saved register (`78e: mov rdi,rsi`). Alternatively, register preparation can also be done using gadget chaining (i.e., chaining indirect jumps) [12, 31], which first prepares the appropriate register (e.g., `RDI`) with the memory location holding the decoded key. Once the register preparation is done, G_t is chained to execute speculatively, leaking the bit from enclave memory. Table 2 shows the results of using both TMPlayer variants. The bit leakage accuracy is computed by checking bits correctly recovered over the total bits (average of 1000 runs). We observe upto 93.2% accuracy in systems without recent RAPL mitigation. In systems with the latest mitigation, the accuracy is still upto 82.4%.

7.3.2 Exfiltrating Plaintext Input from OpenSSL. In the second case study, we demonstrate the ability to exfiltrate plaintext from OpenSSL envelope mode encryption. Specifically, we target a scenario where the victim enclave uses the envelop mode encryption (EVP) of OpenSSL to encrypt session keys using slower asymmetric keys. As the session key is used as the plaintext for encryption, it is considered a secret. The target enclave performs encryption of the given plaintext, in this case, the session key, using the `EVP_EncryptDecryptUpdate` interface of OpenSSL. This function includes a virtual function call (i.e., `ctx->cipher->do_cipher(ctx, out, in, inl)`) that takes the plaintext as the third argument, passed as a pointer through the RDX register. Similar to the previous attack (Section 7.3.1), we assume the victim enclave has a TSX block similar to Listing 4 which contains the virtual function call (`ctx->cipher->do_cipher` in place of Line 2). To leak the plaintext bits, the attacker first poisons the indirect jump inside the TSX block to execute the encryption interface. Then within that, the attacker further poisons the `do_cipher` call to execute the gadget (G_t) used for leaking the plaintext. In this attack, we use a transmitter gadget from the abstract syntax encoding of keys in OpenSSL (Figure 12). The results of this attack, as shown in Table 2, demonstrate high accuracy in bit recovery using either variant of TMPlayer. The results indicate that this type of attack is successful in exfiltrating sensitive information from enclaves using OpenSSL encryption. Similar to the previous attack, we achieve upto 94.3% and 83.6% accuracy using old and new microcode, respectively.

7.3.3 PowSpectre Gadget Analysis. To determine the number of bits per byte that can be leaked, we build a static analysis tool to identify and analyze G_t from Intel SGX SSL [29] and LIBC binaries. We conservatively assume the speculative instruction window to be 30 instructions [10] following the initial indirect jump. Note that while the actual speculation window can be larger, this threshold is set to search for efficient gadgets with fewer non-leaking instructions (i.e., instructions that are executed regardless of the

	RAX	RBX	RCX	RDX	RDI	RSI	RSP	RBP	Total
TEST	3371	0	2	179	0	24	19	14	3609
CMP	1672	8	3	168	374	1	2	398	2626
SHR+ TEST/CMP	3122	100	409	459	5	20	0	70	4185
Mask	0xD3	0xD5	0x91	0xDD	0xE1	0x91	0xD1	0x8B	0xFF

Table 3: Availability of potential PowSpectre transmitter gadget in Intel SGX SSL [29] and LIBC libraries. *Mask* showing leakable bits positions in a byte using the register.

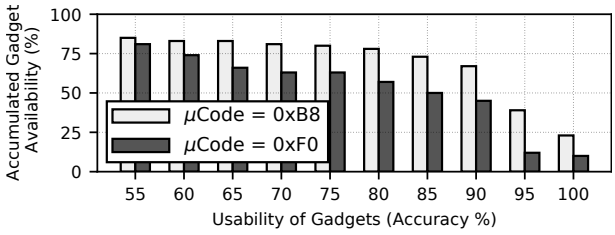


Figure 13: Tradeoff between gadget availability and usability.

secret value). Our study here focuses on gadgets with `TEST` and `CMP` instructions. Moreover, we only consider gadgets that are capable of leaking *precisely a single bit* rather than merely telling a register value equals a constant. Table 3 summarizes our findings. In particular, we find that using `TEST->Jump` or `CMP->Jump` gadget, we can leak all 8 bits of the first byte of a register (**Mask** represents the leakable bit vector). In addition, to leak bits from other bytes, we use `SHR` gadget on register holding data in speculative domain (`SHR` performs logical right-shift operation). This is chained [10, 12, 31] with G_t to enable bit stealing for a different byte through the use of the same sets of gadgets. Furthermore, we randomly choose 500 G_t gadgets and calculate their bit leakage accuracy as a heuristic analysis. Figure 13 shows that a significant number of the gadgets (73% in old microcode and 52% newer microcode) are capable of achieving at least 80% bit leakage accuracy. More notably, 39% and 12% of gadgets respectively can achieve bit leakage accuracy of over 90%. While the gadget analysis is not intended to be exhaustive, our results show *abundant exploitable gadgets* that can exfiltrate secret bits in registers at arbitrary offsets. Finally, for gadgets with high differentiability, we additionally perform experiments to determine their sensitivity to speculation depth in gadget chaining (i.e., the number of indirect control flow transfers under the transient execution path). As illustrated in Figure 14, a speculation depth of one marginally changes the instruction differentiability. Additionally, speculation depth of 2 and 3 still show significant differentiability, making them suitable for statistical analysis. Finally, speculation depths 4 and 5 have high noise, considerably reducing accuracy. We conjecture that the overhead of mistraining up to such depth is the major cause of this degradation. Regardless, this result shows PowSpectre is highly effective even in the case where certain gadget chaining is needed for the attack.

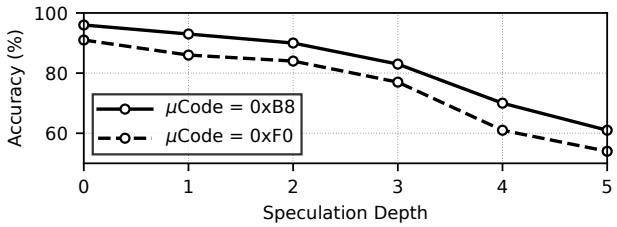


Figure 14: Sensitivity of gadgets with respect to speculation depth (0 represents non-speculative).

8 DISCUSSIONS

8.1 Mitigation

Existing system-level mitigation. Several system-level defenses are available to limit speculative control flow hijacking. For instance, IBPB (Indirect Branch Prediction Barrier)[45] and STIBP (Single Thread Indirect Branch Predictors)[47] prevent BTB poisoning across processes and within the same thread, respectively. IBRS (Indirect Branch Restricted Speculation)[46] and enhanced IBRS (eIBRS) are designed to prevent poisoning across privilege modes. However, recent research has shown that these defenses can be bypassed, and poisoning can still be effective even with these mitigations in place[10]. To mitigate RAPL-based side channel attacks [57], Intel introduced a microcode update that switches RAPL power reporting to a model-based approach [5]. Specifically, Intel introduced a RAPL filtering method that introduces small, random energy noise and reduces the update frequency. It is important to note that these measures exclusively affect RAPL measurements conducted through the `rdmsr` interface. Hardware power management within Intel's SoC continues to operate with unfiltered RAPL values, ensuring that critical hardware features (e.g., Turbo Boost) remain unaffected by this mitigation. However, as demonstrated by this work, Intel's mitigation does not prevent RAPL measurement variance introduced by different instruction executions.

Potential future mitigation for PowSpectre. One potential mitigation is to inject noise into the software measurement interface. With every update to the corresponding MSR, a random bias can be added to the actual energy samples. This may potentially result in irregular noise in the measurement, which can degrade the attack accuracy. However, this method may also result in an overestimation of running power and thermal utilization, as any process observing the interface to maintain device features (such as thermal management systems, turbo boost management) will also observe the overshoot of added noise. An alternative approach would be to execute additional random instructions along with the original sets of instructions, which will have a similar random bias effect on the actual power consumption. Since Intel is already using model-based power reporting, a promising approach specifically for PowSpectre is to exclude squashed instructions from the power reporting logic, eliminating differentiable power observation due to transient execution. Finally, this can be combined with hardware support to limit the number of retries for memory transactions with TSX.

Restricting software interface for power measurement. RAPL-based power side channels can be entirely mitigated by disabling

the software interface, although this is an extreme measure that can impair system functionality and performance. Many critical functions rely on power management and monitoring, and disabling the interface could render these features incompatible. An alternative approach involves restricting access to the RAPL interface, allowing only trusted parties to access it. This can be achieved by encrypting the MSR content related to power measurements, with only trusted services possessing the decryption key. When a service requests access to power information, it must authenticate itself as a trusted party, such as a trusted enclave, to obtain raw power data. Essentially, TEE can be augmented to include power measurement services, where the TEE manages decryption keys and verifies the identity and integrity of requesting applications or services. It is worth noting that implementing such a mechanism requires extending the trusted computing base of the TEE to include services that necessitate RAPL measurements.

9 RELATED WORKS

Since the advent of speculation execution attacks [51, 58], side channels have become an extremely worrisome attack vector [19, 24, 27, 71, 78, 81, 82]. Many defense works are introduced to mitigate transient secret leakage through microarchitectural components (e.g., cache [17, 70, 87, 88] while keeping the performance benefits of speculative execution. In contrast, power side channels have been previously studied as a means of extracting sensitive information in *non-speculative domains* [42, 52, 57, 97]. In this study, we demonstrate the feasibility of using a power side channel to recover fine-grained *speculative secrets*, which significantly extends the attack surface. Given the widespread threat of speculative execution attacks, this new attack adds an even harder-to-mitigate transmitting medium for these types of attacks. In addition to information leakage through *direct power measurements*, recent works have investigated the indirect impacts of dynamic power consumption in other channels [27, 59, 84, 85]. Among them, HammerScope [27] demonstrates a way to correlate the rowhammer bit-flip observations with memory power consumption, which can be used for leakage in the speculative domain. Hertzbleed [84, 85] and Frequency throttling [59] utilize dynamic frequency scaling of modern processors due to imposed power limits to determine dynamic power consumption through operating frequency. However, these works have not investigated speculation replay techniques to enable secret stealing without program re-execution.

In a separate line of work, speculation replay techniques have also been demonstrated to enable secret inference from coarse-grained side channels with a high degree of precision [57, 71, 75, 78, 79]. However, they have limited applicability in transient instruction replay. Collide+Power [53] reveals power distinguishability when a memory line with different data patterns is loaded into the cache. The attack leverages such observation and performs a power side channel using differential analysis by exploiting speculative loads. Notably, PowSpectre leverages *replay* of speculative instruction execution without advancing instructions while [53] exploits *repetitive* speculative loads in regular program constructs. To the best of our knowledge, our work is the first to demonstrate instruction-based power leakage in most recent RAPL using replay of speculative instruction execution. Among the most related works, Platypus [57] uses zero-stepping for non-speculative side channels against TEE. Concurrent to our investigation, the studies in AEX-Notify [28] indicate that if an interrupt arrives during the execution of `ERESUME` routine (i.e., zero-step), which is similar to our observation. However, they did not investigate whether enclave instructions are still executed under transient execution in zero-step. In addition, our paper revealed that zero-stepping cannot be used for transient execution replay. MicroScope [75] uses page fault for replaying timing channels, but only for enhancing non-speculation leakage. In this study, we utilize TSX gadgets present in victim applications [21, 38, 65] as an instruction replay technique (TMPlayer) which is highly effective in the speculative domain. Differently from prior work in Prime+Abort [32] and DrK [48] that use TSX in the *attacker space* to observe timing information, TMPlayer utilizes the replay capabilities of TSX to *replay* instructions for denoising purpose only.

Applicability of PowSpectre Due to the identification of the TSX Asynchronous Abort (TAA) vulnerability shown in prior works [7, 32], Intel has deprecated TSX from modern processors along with disabling TSX by default. However, the performance benefits of transactional memory are non-negligible, and we envision that Intel (and other vendors) are likely to roll out new variants of TAA-secure transactional memory implementation in the near future. Therefore, it is important to understand the holistic security of transactional memory to ensure microarchitecture security for future systems. Note that in processors with the deprecated TSX, this feature can still be enabled at user discretion through boot flags (i.e., "tsx async abort=off tsx=on"). In TMPlayer, we repurpose the abort handling of TSX as a replay primitive, which is different from the TAA exploitation and is fundamental to the design of transactional memory. PowSpectre is independent of TMPlayer and can manifest (with a lower accuracy) even when TSX is disabled, using page fault-based replay primitive instead of TMPlayer, as highlighted by our evaluation in Section 7.2.

10 CONCLUSION

In this work, we present PowSpectre- a software-based (i.e., RAPL) power side channel exfiltrating enclave secrets from the speculative domain. This is the first work that highlights the secret leakage capabilities of power side channel in transient execution. To denoise the power measurements from RAPL interface, we investigate prior instruction replay techniques and discover their shortcoming in transient execution. We further design TMPlayer- an instruction replay technique utilizing the Intel TSX which can overcome the prior limitations. Equipped with TMPlayer, our evaluation shows that PowSpectre can be used to exfiltrate enclave secretive data (e.g., crypto keys and private plaintext) in the *speculative domain* with very high accuracy. Our further analysis shows the wide availability of the power distinguishable gadgets for PowSpectre. Our work highlights the importance of understanding advanced replay-based side channels in SGX.

ACKNOWLEDGMENTS

This work is supported in part by U.S. National Science Foundation under CNS-2008339 and CNS-2147217.

REFERENCES

[1] [n. d.]. 2020.2 IPU - Intel RAPL Interface Advisory. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00389.html

[2] [n. d.]. Introduction to Cache Allocation Technology in the Intel® Xeon® Processor E5 v4 Family. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-cache-allocation-technology.html

[3] [n. d.]. microcode-20201110 release: Intel INTEL-SA-00381. https://github.com/intel/Intel-Linux-Processor-Microcode-Data-Files/releases/tag/microcode-20201110/

[4] [n. d.]. microcode-20220510 release. https://github.com/intel/Intel-Linux-Processor-Microcode-Data-Files/releases/tag/microcode-20220510/

[5] [n. d.]. PLATYPUS: With Great Power comes Great Leakage. https://platypusattack.com/

[6] [n. d.]. Running Average Power Limit Energy Reporting / CVE-2020-8694 , CVE-2020-8695 / INTEL-SA-00389 . https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html

[7] [n. d.]. TAA - TSX Asynchronous Abort. https://www.intel.com/content/www/us/en/newsroom/news/ucsf-propel-medical-device-innovations.html

[8] Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. 2007. On the power of simple branch prediction analysis. In *IEEE ISCA*.

[9] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida Garcia, and Nicola Tuveri. 2019. Port Contention for Fun and Profit. In *IEEE S&P*.

[10] Enrico Barberis, Pietro Frigo, Marius Muench, Herbert Bos, and Cristiano Giuffrida. 2022. Branch history injection: On the effectiveness of hardware mitigations against cross-privilege Spectre-v2 attacks. In *USENIX Security*.

[11] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Springer Noise reduction in speech processing*.

[12] Atri Bhattacharyya, Andrés Sánchez, Esmaeil M Koruyeh, Nael Abu-Ghazaleh, Chengyu Song, and Mathias Payer. 2020. Specrop: Speculative exploitation of ROP chains. In *USENIX RAID*.

[13] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandtner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. 2019. SMoTherSpectre: Exploiting Speculative Execution through Port Contention. In *ACM CCS*.

[14] Thomas Bourgeat, Jules Drean, Yuheng Yang, Lillian Tsai, Joel Emer, and Mengjia Yan. 2020. Casa: End-to-end quantitative security analysis of randomly mapped caches. In *IEEE MICRO*.

[15] Thomas Bourgeat, Ilia Lebedev, Andrew Wright, Sizhuo Zhang, Arvind, and Srinivas Devadas. 2019. MI6: Secure Enclaves in a Speculative Out-of-Order Processor. In *IEEE MICRO*.

[16] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. 2019. Fallout: Leaking Data on Meltdown-resistant CPUs. In *ACM CCS*.

[17] 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*.

[18] Sunjay Cauligi, Craig Disselkoen, Klaus v Gleissenthall, Dean Tullsen, Deian Stefan, Tamara Rezk, and Gilles Barthe. 2020. Constant-time foundations for the new spectre era. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[19] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2019. Sgxpectre: Stealing intel secrets from SGX enclaves via speculative execution. In *IEEE EuroS&P*.

[20] Jie Chen and Guru Venkataramani. 2014. Cc-hunter: Uncovering covert timing channels on shared processor hardware. In *IEEE MICRO*.

[21] Sanchuan Chen, Xiaokuan Zhang, Michael K. Reiter, and Yinqian Zhang. 2017. Detecting Privileged Side-Channel Attacks in Shielded Execution with Déjà Vu. In *ACM ASIACCS*.

[22] Md Hafizul Islam Chowdhuryy, Rickard Ewetz, Amro Awad, and Fan Yao. 2021. R-SAW: New Side Channels Exploiting Read Asymmetry in MLC Phase Change Memories. In *IEEE SEED*.

[23] Md Hafizul Islam Chowdhuryy, Rickard Ewetz, Amro Awad, and Fan Yao. 2023. Understanding and Characterizing Side Channels Exploiting Phase-Change Memories. *IEEE Micro* (2023).

[24] Md Hafizul Islam Chowdhuryy, Hang Liu, and Fan Yao. 2020. BranchSpec: Information Leakage Attacks Exploiting Speculative Branch Instruction Executions. In *IEEE ICCD*.

[25] Md Hafizul Islam Chowdhuryy and Fan Yao. 2021. Leaking Secrets through Modern Branch Predictor in the Speculative World. *IEEE TC* (2021).

[26] Md Hafizul Islam Chowdhuryy, Zhenkai Zhang, and Fan Yao. 2023. BeKnight: Guarding Against Information Leakage in Speculatively Updated Branch Predictors. In *IEEE ICCAD*.

[27] Yaakov Cohen, Kevin Sam Tharayil, Arie Haenel, Daniel Genkin, Angelos D Keromytis, Yossi Oren, and Yuval Yarom. 2022. HammerScope: Observing DRAM Power Consumption Using Rowhammer. *ACM CCS* (2022).

[28] Scott Constable, Jo Van Bulck, Xiang Cheng, Yuan Xiao, Cedric Xing, Ilya Alexandrovich, Taesoo Kim, Frank Piessens, Mona Vij, and Mark Silberstein. 2023. AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves. In *USENIX Security 23*.

[29] Intel Corporation. [n. d.]. Intel® Software Guard Extensions SSL. https://github.com/intel/intel-sgx-ssl

[30] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* (2016).

[31] Jinhua Cui, Jason Zhijingcheng Yu, Shweta Shinde, Prateek Saxena, and Zhiping Cai. 2021. SmashEx: Smashing SGX Enclaves Using Exceptions. In *ACM CCS*.

[32] Craig Disselkoen, David Kohlbrenner, Leo Porter, and Dean Tullsen. 2017. Prime+Abort: A Timer-FreeHigh-Precision L3 Cache Attack using Intel TSX. In *USENIX Security*.

[33] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2016. Jump over ASLR: Attacking branch predictors to bypass ASLR. In *IEEE MICRO*.

[34] Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. Branchscope: A new side-channel attack on directional branch predictor. In *ACM ASPLOS*.

[35] Hongyu Fang, Sai Santosh Dayapule, Fan Yao, Miloš Doroslovački, and Guru Venkataramani. 2018. A noise-resilient detection method against advanced cache timing channel attack. In *IEEE ACSSC*.

[36] GPG. 2013. Mitigate a flush+reload cache attack on RSA secret exponents. (2013). https://github.com/gpg/libgcrypt/commit/e2202ff2b

[37] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2018. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In *USENIX Security*.

[38] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory. In *USENIX Security*.

[39] Michael Karl Gschwind, Valentina Salapura, and Chung-Lung K Shum. 2018. Accurate tracking of transactional read and write sets with speculation. US Patent 10,055,230.

[40] Michael K Gschwind, Valentina Salapura, and Chung-Lung K Shum. 2020. Read and write sets for ranges of instructions of transactions.

[41] Jawad Haj-Yahya, Lois Orosa, Jeremie S Kim, Juan Gómez Luna, A Giray Yağlıkçı, Mohammed Alser, Ivan Puddu, and Onur Mutlu. 2021. IChannels: Exploiting Current Management Mechanisms to Create Covert Channels in Modern Processors. In *IEEE ISCA*.

[42] Yi Han, Matthew Chan, Zahra Aref, Nils Ole Tippenhauer, and Saman Zonouz. 2022. Hiding in Plain Sight? On the Efficacy of Power Side Channel-Based Control Flow Monitoring. In *USENIX Security*.

[43] Zhou Hongwei, Ke Zhipeng, Zhang Yuchen, Wu Dangyang, and Yuan Jinhui. 2021. TSGX: Defeating SGX Side Channel Attack with Support of TPM. In *IEEE ACCTCS*.

[44] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. 2015. Understanding contention-based channels and using them for defense. In *IEEE HPCA*.

[45] Intel. 2018. Deep Dive: Indirect Branch Predictor Barrier. https://software.intel.com/security-software-guidance/deep-dives/deep-dive-indirect-branch-predictor-barrier

[46] Intel. 2018. Deep Dive: Indirect Branch Restricted Speculation. https://software.intel.com/security-software-guidance/insights/deep-dive-indirect-branch-restricted-speculation

[47] Intel. 2018. Deep Dive: Single Thread Indirect Branch Predictors. https://software.intel.com/security-software-guidance/deep-dives/deep-dive-single-thread-indirect-branch-predictors

[48] Yeongjin Jang, Sangho Lee, and Taesoo Kim. 2016. Breaking kernel address space layout randomization with intel tsx. In *ACM CCS*.

[49] S Karen Khatamifard, Longfei Wang, Amitabh Das, Selcuk Kose, and Ulya R Karpuzcu. 2019. Powert channels: A novel class of covert communicationexploiting power management vulnerabilities. In *IEEE HPCA*.

[50] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. 2018. DAWG: A defense against cache timing attacks in speculative execution processors. In *IEEE MICRO*.

[51] 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 *IEEE S&P*.

[52] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Springer Annual international cryptology conference*.

[53] 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*.

[54] Vamsee Reddy Kommareddy, Baogang Zhang, Fan Yao, Rickard Ewetz, and Amro Awad. 2019. Are Crossbar Memories Secure? New Security Vulnerabilities in Crossbar Memories. In *IEEE CAL*.

[55] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *USENIX Security*.

[56] Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. Amd prefetch attacks through power and time. In *USENIX Security*.

[57] 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 *IEEE S&P*.

[58] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. 2018. Meltdown: Reading kernel memory from user space. In *USENIX Security*.

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

[60] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B Lee. 2016. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *IEEE HPCA*.

[61] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *IEEE S&P*.

[62] Robert Martin, John Demme, and Simha Sethumadhavan. 2012. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *IEEE ISCA*.

[63] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. Cachezoom: How SGX amplifies the power of cache attacks. In *Springer CHES*.

[64] OpenSSL. [n. d.]. OpenSSL 1.1.0 Series Release Notes. https://www.openssl.org/news/openssl-1.1.0-notes.html

[65] Meni Orenbach, Andrew Baumann, and Mark Silberstein. 2020. Autarky: Closing controlled channels with self-paging enclaves. In *ACM EuroSys*. 1–16.

[66] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *Springer CT-RSA*.

[67] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM addressing for Cross-CPU attacks. In *USENIX Security*.

[68] Ivan Puddu, Moritz Schneider, Miro Haller, and Srdjan Čapkun. 2021. Frontal Attack: Leaking Control-Flow in SGX via the CPU Frontend. In *USENIX Security*.

[69] Xida Ren, Logan Moody, Mohammadkazem Taram, Matthew Jordan, Dean M Tullsen, and Ashish Venkat. 2021. I see dead μops: Leaking secrets via Intel/AMD micro-op caches. In *IEEE ISCA*.

[70] Gururaj Saileshwar and Moinuddin K Qureshi. 2019. CleanupSpec: An 'Undo' Approach to Safe Speculation. In *IEEE MICRO*.

[71] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *ACM CCS*.

[72] Nader Sehatbakhsh, Baki Berkay Yilmaz, Alenka Zajic, and Milos Prvulovic. 2020. A new side-channel vulnerability on modern computers by exploiting electromagnetic emanations from the power management unit. In *IEEE HPCA*.

[73] Madura A Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. 2019. Rosita: Towards automatic elimination of power-analysis leakage in ciphers. *arXiv preprint arXiv:1912.05183* (2019).

[74] Mert Side, Fan Yao, and Zhenkai Zhang. 2022. Lockeddown: Exploiting contention on host-gpu pcie bus for fun and profit. In *IEEE EuroS&P*.

[75] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W Fletcher. 2019. Microscope: Enabling microarchitectural replay attacks. In *IEEE ISCA*.

[76] Daniel Townley, Kerem Arıkan, Yu David Liu, Dmitry Ponomarev, and Oğuz Ergin. 2022. Composable Cachelets: Protecting Enclaves from Cache Side-Channel Attacks. In *USENIX Security*.

[77] Paul Turner. 2018. Retpoline: a software construct for preventing branch-target-injection. https://support.google.com/faqs/answer/7625886

[78] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *USENIX Security*.

[79] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2017. SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In *ACM SsyTEX*.

[80] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In *ACM CCS*.

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

[82] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2021. CacheOut: Leaking Data on Intel CPUs via Cache Evictions. In *IEEE S&P*.

[83] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. 2017. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In *ACM CCS*.

[84] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang 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*.

[85] Yingchen Wang, Riccardo Paccagnella, Alan Wandke, Zhao Gang, Grant Garrett-Grossman, Christopher W Fletcher, David Kohlbrenner, and Hovav Shacham. 2023. DVFS frequently leaks secrets: Hertzbleed attacks beyond SIKE, cryptography, and CPU-only data. In *IEEE S&P*.

[86] Haocheng Xiao and Sam Ainsworth. 2023. Hacky racers: Exploiting instruction-level parallelism to generate stealthy fine-grained timers. In *ACM ASPLOS*.

[87] Wenjie Xiong and Jakub Szefer. 2021. Survey of Transient Execution Attacks. *Comput. Surveys* (2021).

[88] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher Fletcher, and Josep Torrellas. 2018. InvisiSpec: Making speculative execution invisible in the cache hierarchy. In *IEEE MICRO*.

[89] Mengjia Yan, Bhargava Gopireddy, Thomas Shull, and Josep Torrellas. 2017. Secure Hierarchy-Aware Cache Replacement Policy (SHARP): Defending Against Cache-Based Side Channel Atacks. In *IEEE ISCA*.

[90] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. 2019. Attack Directories, Not Caches: Side Channel Attacks in a Non-Inclusive World. In *IEEE S&P*.

[91] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. 2018. Are coherence protocol states vulnerable to information leakage?. In *IEEE HPCA*.

[92] Fan Yao, Miloš Doroslovački, and Guru Venkataramani. 2019. Covert timing channels exploiting cache coherence hardware: Characterization and defense. *Springer IJPP* (2019).

[93] Fan Yao, Hongyu Fang, Miloš Doroslovački, and Guru Venkataramani. 2019. COTSknight: Practical defense against cache timing channel attacks using cache monitoring and partitioning technologies. In *IEEE HOST*.

[94] Fan Yao, Guru Venkataramani, and Miloš Doroslovački. 2017. Covert timing channels exploiting non-uniform memory access based architectures. In *ACM GLSVLSI*.

[95] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security*.

[96] Zihao Zhan, Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xenofon Koutsoukos. 2022. Graphics peeping unit: Exploiting em side-channel information of gpus to eavesdrop on your neighbors. In *IEEE S&P*.

[97] Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xing Gao. 2021. Red Alert for Power Leakage: Exploiting Intel RAPL-Induced Side Channels. In *ACM CCS*.