Exploring side-channels in Intel® Trust Domain Extensions

Upasana Mandal[1], *Shubhi Shukla*[2], *Nimish Mishra*[1],
Sarani Bhattacharya[1], *Paritosh Saxena*[3], *Debdeep Mukhopadhyay*[1]
[1]*Department of Computer Science and Engineering, IIT Kharagpur, India*
[2]*Centre for Computational and Data Sciences, IIT Kharagpur, India*
[3]*Intel Corporation, USA*
{mandal.up98, shubhishukla, nimish.mishra}@kgpian.iitkgp.ac.in,
{sarani.bhattacharya,debdeep}@cse.iitkgp.ac.in,
paritosh.saxena@intel.com

Abstract

Intel Trust Domain Extensions (TDX) has emerged as a crucial technology aimed at strengthening the isolation and security guarantees of virtual machines, especially as the demand for secure computation is growing largely. Despite the protections offered by TDX, in this work, we dig deep into the security claims and uncover an intricate observation in TDX. These findings undermine TDXs core security guarantees by breaching the isolation between the Virtual Machine Manager (VMM) and Trust Domains (TDs). In this work for the first time, we show through a series of experiments that these performance counters can also be exploited by the VMM to differentiate between activities of an idle and active TD. The root cause of this leakage is core contention. This occurs when the VMM itself, or a process executed by the VMM, runs on the same core as the TD. Due to resource contention on the core, the effects of the TDs computations become observable in the performance monitors collected by the VMM. This finding underscore the critical need for enhanced protections to bridge these gaps within these advanced virtualized environments.

1 Introduction

Intel® introduced Software Guard Extensions (SGX) [9], a set of processor instructions that create secure enclaves isolated memory regions for executing sensitive code and data securely, even in compromised environments. SGX quickly became popular in applications like secure cloud computing and privacy-preserving machine learning due to its strong security guarantees. However, despite its robust design, like any new security technology, multiple vulnerabilities have been reported on SGX that exploit hardware and software weaknesses. SGX has been targeted by various attack categories, including cache, branch prediction, DVFS-based, address translation-based attacks, and software vulnerabilities [3–8, 12–17]. Intel has addressed these vulnerabilities in timely fashion, and performed TCB recoveries to demonstrate attestable application of mitigations.

To enable usage models in cloud computing, Intel introduced *Intel® Trust Domain Extensions (TDX)* [10]. TDX is an advanced security technology designed to isolate virtual machines, called TDs, from the underlying system software, including the hypervisor or Virtual Machine Manager (VMM). TDX enhances confidentiality and integrity for both memory and CPU state, offers address-translation integrity, and supports secure interrupt and exception delivery. It also includes features like remote attestation and live migration of TDs for protecting workloads in virtualized environments. Unlike SGX, TDX does not just ensure the secure execution of a part of the process but secures the entire environment for the process execution from the host system or the VMM. Although Intel® has significantly enhanced TDX to be more robust against a wide range of attack vectors, making it much more secure than SGX, no system can be entirely impervious to all threats.

In this work, we uncover a vulnerability in TDX's (Performance Monitoring Counters) PMC virtualization that not only compromises the isolation between the VMM and TD but also breaches the isolation between different TDs, thereby undermining the core guarantees of TDX. To expand on this, a primary objective of the TDX module is to ensure complete isolation of any processing within a TD from both the VMM and other TDs running concurrently on the same system. This isolation extends to concealing or obfuscating the Hardware Performance Counters (HPCs) of the TDs from the VMM, even when the VMM operates with root privileges [11] to monitor these events. However, in a particular scenario where the VMM and a TD are co-located on the same core, resource contention arises, exposing the TD's computation patterns on PMCs collected by the VMM for its own processes making PMC virtualization ineffective.

We demonstrate in this work that when a TD is launched with the `perf` command added to its prefix, we can clearly distinguish between an idle TD and an active one with some ongoing processing and this was seen through most common avaliable HPCs through the `perf` tool. In addition we also demonstrate successful process fingerprinting and class-leakage attack by exploiting this vulnerability.

2 Background

2.1 Intel® Trust Domain Extensions (TDX)

In the initial stages of virtualization, the focus was on optimizing resource utilization and reducing operational costs by consolidating multiple VMs on a single physical server, with the hypervisor, managing VM execution and resource allocation. However, the hypervisor's privileged position introduced substantial security vulnerabilities. As cloud computing and multi-tenant environments grew in popularity, the need for stronger isolation between VMs became apparent.

To mitigate these challenges, Intel® initiated the development of a novel architectural extension aimed at safeguarding sensitive operations within virtualized environments, even in the presence of a compromised VMM named as ***Intel® Trust Domain Extensions*** **(TDX)** .

The Intel® TDX module is a CPU-attested software module that which enhances Virtual Machine Extensions (VMX) and Multi-Key Total Memory Encryption (MKTME) by introducing a novel type of virtual machine guest known as a Trust Domain (TD). It is responsible for managing and protecting TDs. The TDX module leverages Secure Arbitration Mode (SEAM) to create a secure environment where it can run without being exposed to potentially malicious software outside the SEAM context.

The SEAM, is an extension to the existing Virtual Machine Extensions (VMX) architecture. SEAM introduced a paradigm shift by enabling the establishment of a secure execution environment within the CPU, isolated from the rest of the system, including the VMM. SEAM defined two distinct operational modes:

SEAM VMX Root Operation: A secure mode in which CPU-attested software modules can operate, fully isolated from the VMM and other software.

SEAM VMX Non-Root Operation: A mode allowing secure VMs, referred to as TDs, to execute with their memory and CPU state safeguarded from the VMM.

Building upon SEAM's foundation, Intel® developed Intel® Trust Domain Extensions (TDX) to extend SEAM's security capabilities to a broader area. Leveraging SEAM's architecture, TDX ensures that TDs execute securely, with their memory contents and CPU state shielded from all other software, including a potentially compromised VMM.

At the core of TDX architecture is the SEAM Range Register (SEAMRR), which defines a protected memory region, accessible solely by SEAM VMX root operations. This secure environment hosts two pivotal CPU-attested modules: the Intel® TDX module and the Intel® Persistent SEAMLDR (P-SEAMLDR) module. This architecture ensures that the memory and CPU state of TDs remain protected from unauthorized access, thereby establishing a robust security foundation for the execution of sensitive workloads in virtualized environments.

The role of the Intel® TDX module can be summarized as:

❶ The TDX module enforces strict access control policies, ensuring that only the TD has access to its memory and CPU state. The hypervisor and other VMs cannot access the TDs resources unless explicitly allowed by the TD.

❷ The TDX module ensures that the memory allocated to a TD cannot be tampered with. It tracks the memory pages assigned to the TD and ensures that no unauthorized modifications can occur.

❸ The TDX module supports attestation policies that allow external parties to verify the integrity and security of the TD. By measuring the TDs firmware and providing cryptographic proofs, the TDX module enables the creation of a trusted execution environment.

❹ The TDX module enforces policies that ensure the TD is securely initialized and that its resources are securely wiped during shutdown. This prevents any residual data from being accessed after the TD is no longer in operation.

2.2 Trust Domain (TD)

Trust Domains (TDs) are the secure VMs that are managed by the TDX module. These TDs operate in SEAM VMX non-root mode, meaning they are isolated from the regular VMX root environment (where the VMM operates). This isolation ensures that the memory and CPU state of the TDs are protected from the VMM and other external software. The process of launching of a TD from the VMM can be described as below:
The VMM initiates the creation and launch of a TD by preparing the TDVF image, which contains essential boot code, security configurations, and initialization components. The VMM employs the Intel® TDX module to perform an initial cryptographic measurement of the TDVF image, storing the resulting hash in the TD Measurement Register (MRTD) to ensure firmware integrity. Subsequently, the VMM initializes and encrypts the memory allocated to the TD, ensuring its isolation from other software, including the VMM itself. Virtual CPUs (vCPUs) are also set up, with one designated as the Bootstrap Processor responsible for initial system setup.

The TDVF is launched in 32-bit protected mode, with the BSP transitioning the system to 64-bit long mode, enabling modern OS and application support. The memory paging system is configured, and the stack is set up to support more complex firmware execution. The TDVF establishes the Unified Extensible Firmware Interface (UEFI) environment, bypassing the Pre-EFI Initialization (PEI) phase due to pre-configured memory settings. The BSP then initializes the Driver Execution Environment (DXE) Core, responsible for loading and initializing necessary drivers, and prepares the final memory map and ACPI tables.

Finally, the TDVF loads the operating system (OS) loader, transitioning control to the OS and completing the secure launch of the TD. The TD operates in a fully isolated and secure environment, with the VMM providing necessary resources while remaining isolated from the TD's internal operations.

TDs are instantiated by the host VMM through a sequence of SEAMCALL instructions, integral to the Intel® TDX module architecture. The TDX module enforces strict memory isolation between the TDs by leveraging Secure Extended Page Tables (SEPT), ensuring that each TD's memory space is cryptographically protected and inaccessible to other TDs. Each TD operates with its own set of virtual CPUs, and the TDX module guarantees that the CPU state, including control registers, Model-Specific Registers (MSRs), and other critical execution contexts, remains completely isolated across TDs. This robust isolation is critical in ensuring that even in the presence of a compromised TD, the execution environment of the TD_y remains secure and free from interference, maintaining the integrity of sensitive operations within clearly defined and isolated boundaries.

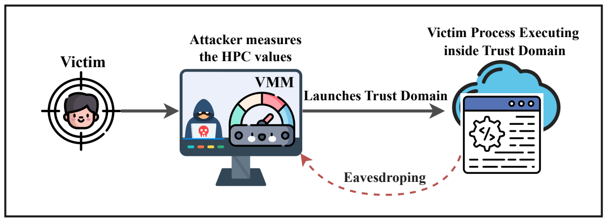

Victim
Attacker measures the HPC values
VMM
Launches Trust Domain
Victim Process Executing inside Trust Domain
Eavesdroping

Figure 1: Threat Model: Isolation Breach between TD and VMM

3 Threat Model

We now discuss adversarial assumptions that we follow throughout the rest of the paper. We assume an adversary with access to an Intel Xeon Sapphire Rapids system with TDX enabled. All the data within a TD remains encrypted. When a user performs tasks inside the TD, the data gets decrypted and upon exiting the TD the data is re-encrypted to safeguard the sensitive information [10]. The Trusted Computing Base (TCB) of Intel TDX, is the set of hardware/software components within the trust boundary of a single TD. Concretely, each TD trusts only the TDX Module, Attestation software, hardware, and Authenticated Code Modules. The adversary is VMM or host SW. We assume that the victim TD is configured with both `ATTRIBUTES.PERFMON`[1] and `ATTRIBUTES.DEBUG`[2] disabled. The adversary is allowed to execute arbitrary code *outside* the TCB of victim TD and collect its PMC traces. The objective of this adversary is to probe the PMC virtualization boundary between a victim TD and the VMM in order to explore its isolation properties. Refer Fig. 1 for a graphical representation.

4 Isolation Breach between TD and VMM

The Intel® TDX module is integral to managing and upholding security properties within a TD. It enforces rigorous access control policies, ensuring that only the TD can access its associated memory and CPU state while maintaining an isolation boundary between the TD and the VMM. However, since by design VMM remains in control of TDs execution and TDs resource management, this isolation boundary is not tight, and in this work, we aim to explore the security implications that arise from this.

We start by utilizing HPCs to facilitate a side-channel leakage from within the TD. HPCs, which track various low-level events such as cache misses, branch instructions, and CPU cycles, are critical for optimizing system performance. However, when exposed or misused, these counters can serve as powerful side-channel sources [1, 2]. By carefully analyzing the patterns in these metrics, an attacker can extract sensitive information, such as identifying the type of computation being performed, the execution flow of cryptographic algorithms, or even specific data-dependent operations.

In a TD where confidentiality and integrity are critical, the capability to monitor hardware performance metrics from the VMM represents a significant security vulnerability. Although the VMM is designed to be isolated from the TD, and TDs HPCs are not accessible to VMM, VMM can nonetheless measure its own HPCs while executing the TD, facilitating the creation of a detailed profile of the processes within the TD.

Our Objective

Our aim is to prove that it remains possible for the VMM to thereby posing a significant risk of information leakage from the TD and lead to the isolation breach between TD and VMM.

Methodology: Each TD is instantiated from the VMM, using the command:

To measure the HPCs for processes running inside the Trust Domain (TD), we utilized the `perf stat` command-line tool in Linux, which is widely used for performance analysis and

[1]Allows PMC profiling *inside* the TD. By default, TD is prohibited from explicit PMC profiling.

[2]Allows the VMM to debug the TD by disabling several protection features in TDX.

TD launch command

```
./start-qemu.sh -i TD_image.qcow2 -k
kernel_file.rpm
```

monitoring. The TD was initiated from the Virtual Machine Manager (VMM) using the command:

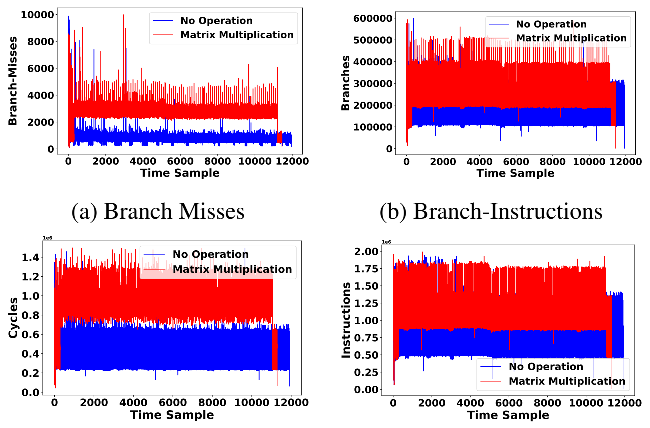
perf command

```
sudo perf stat -e cycles, instructions, L1-dcache-
load-misses,L1-dcache-loads, branches, branch-
misses -I 50 -o output.txt ./start-qemu.sh -i
TD_image.qcow2 -k kernel_file.rpm
```

This command enables the monitoring and collection of detailed performance statistics, such as CPU cycles, instructions, cache load misses, branch instructions, and branch misses, while the TD is operational. It is important to note that the above command collects HPC counters for the VMM process (the qemu process in our case) executing a TD guest and not the TD guest itself, and therefore we were able to collect this data. If the command is modified to collect just TDs HPC via :G qualifier (i.e. instructions:G,cpu_clk_unhalted.thread:G,cpu_clk_unhalted.ref_tsc:G), then the counters will not be exposed when TD is executing. These statistics were captured at sampling mode and saved to a designated output file throughout the execution of the TD. By analyzing the metrics recorded in the output file, we were able to gain insights into the processes executed within the TD.

To demonstrate the capability of distinguishing between different processes running inside a TD, we conducted an experiment using two distinct types of processes: a) **Process 1** a simple idle operation where the code instructs the system to sleep for 10 minutes and, b) **Process 2** a computationally intensive process, involving the execution of complex matrix multiplication. The experiment was carried out as follows: **TD Executing Process 1:** We initiated the TD environment using the `perf stat` command and executed Process 1 within this TD. The collected data was then saved into a designated file for subsequent analysis.
TD Executing Process 2: Similarly, we initiated the TD a second time using the `perf stat` command, but this time Process 2 was executed within the TD. As with Process 1, the performance metrics were recorded at 50-millisecond intervals and saved to a separate output file.

Results: By comparing the performance metrics gathered from the two processes, we were able to identify significant differences in the recorded HPCs. The clear distinction between the performance profiles of Process 1 and Process 2, as observed in the recorded metrics, underscores the ability to differentiate between different types of workloads running inside a TD based on HPCs. This differentiation is crucial for analyzing the behavior of processes within secure environments and for identifying potential side-channel charac-



(a) Branch Misses (b) Branch-Instructions

(c) Cycles (d) Instructions

Figure 2: A comparative visualization of different hardware performance metrics of Process 1 (No operation) and Process 2 (Matrix multiplication) executed inside a TD

teristics that may arise from such performance discrepancies. Figure 2 shows the comparison between Process 1 and Process 2, where Process 2 consistently shows higher values across all metrics compared to Process 1. Therefore, we assert that an ability to collect these measurements and use them to obtain information about the workload executing inside TD has implications on the security and confidentiality of processes executing within the TD, and while all confidential computing technologies including Intel TDX do not currently protect against this, we think this is a gap worth addressing.

4.1 Process Fingerprinting

Initially, we employed a foundational approach, leveraging HPC metrics to differentiate between two processes running within the TD, as illustrated in Figure 2. Building on this foundational insight, we now explore more sophisticated case studies that reveal the extended risks of this vulnerability. Specifically, we aim to profile multiple processes within the TD by monitoring the HPC metrics from the VMM.

We assume the VMM to be a potentially malicious entity. With the necessary control structures and memory encryption mechanisms established, we initiate the TD from the VMM using the aforementioned `perf` command. Once launched, the TD operates within a completely isolated environment, where its memory and state are safeguarded by hardware-enforced encryption and access controls.

Inside the TD we execute the UnixBench benchmark suite designed to measure the performance of systems. UnixBench performs a variety of tests to evaluate different components of the system. The different benchmark processes that the UnixBench suite consists of are: ⓐ dhry2reg, ⓑ whetstone-double, ⓒ syscall, ⓓ pipe, ⓔ context1, ⓕ spawn, ⓖ execl, ⓗ fstime, and ⓘ shell16.

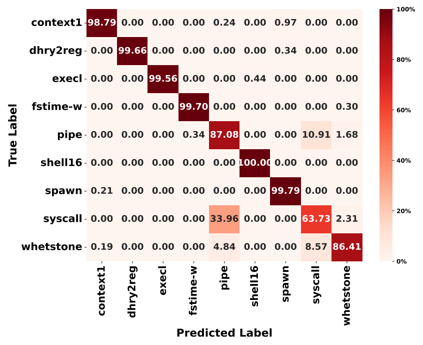


Figure 3: Classification accuracies for different Benchmarks

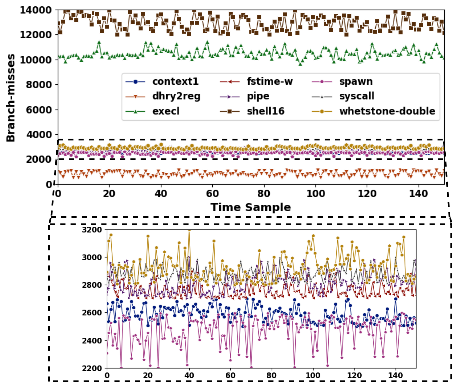


Figure 4: Branch miss values across different Benchmarks

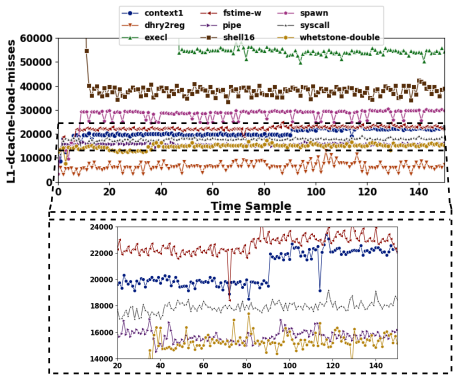


Figure 5: L1-dcache-load-miss values across different Benchmarks

We sequentially execute all nine benchmarks within the TD. For each benchmark, the TD is launched from the VMM, and the benchmark is executed. Simultaneously, the perf stat command collects hardware performance metrics in a sampling mode with a 50ns interval. Upon completion of each benchmark, the TD is exited, and the process is repeated for the remaining eight benchmarks.

We systematically gathered the HPC values for all of the nine benchmarks. By performing a comprehensive analysis of this HPC data, we successfully generated distinctive fingerprints for each benchmark. As illustrated in Figure 4, branch miss values are pivotal in distinguishing between benchmarks running within the TD. We observe that the shell16 benchmark has the highest range of branch miss values because shell16, executing 16 concurrent shell instances, likely incurs higher branch miss rates due to increased conditional branches, and frequent context switches. Similarly, Figure 5, describes how the different benchmarks can be distinguished with their respective L1-dcache-load-miss values.

To validate our findings further, we collected approximately 20 HPC data files per benchmark. We constructed a CNN classifier model that can classify between all these nine benchmarks executing inside the TD based on the HPC data. CNNs are highly effective for classification tasks involving structured data due to their ability to learn hierarchical feature representations through convolutional layers. They excel at capturing local patterns and spatial dependencies, allowing the model to automatically extract meaningful features from raw data. Figure 3 presents the confusion matrix evaluating the model's classification accuracy. The diagonal entries in the matrix signify the model's precision in correctly identifying each benchmark, whereas the off-diagonal entries represent misclassifications, indicating instances where the model incorrectly labels one benchmark as another.

4.2 Class Leakage Attack

In prior work, we have demonstrated the effectiveness of utilizing `perf stat` command to classify and distinguish between multiple processes executing within a TD. We extend our analysis by targeting the fine-grained profiling of a single process within the TD. In particular, we focus on extracting detailed information about the specific image class being inferred by a machine learning model operating within the TD.

During the inference operation, a trained machine learning model uses its learned parameters to compute outputs such as class probabilities, regression values, etc. While the primary goal of inference is to make accurate predictions, this can inadvertently reveal sensitive information through information leaks. These leaks can be quantified by various performance metrics, including execution time, branch misses, cache accesses, cycles, etc. The study presented in [18] demonstrates that timing values can be exploited to infer the class predicted by a CNN model by monitoring timing variations, named class leakage attack. The attack exploits a timing side-channel vulnerability found in the PyTorch framework, particularly in the Max Pooling operation of CNNs. This vulnerability arises from the non-constant time implementation of the Max Pooling function. Specifically, the function relies on a conditional branching operation (if statement) to determine and update the maximum value within a pooling window. The frequency of this conditional branch being executed varies depending on the specific input data. This variance leads to differences in the execution time of the Max Pooling operation, which can be exploited to infer the class labels of input data.

In our scenario, the host VMM is assumed to be the adversary. We launch the TD from the compromised host VMM and inside the TD, a trained CNN model is deployed. The CNN model is trained on two widely recognized image classification benchmarking datasets: CIFAR-10 and CIFAR-100, after training it we exit the TD.

We again launch the TD, from the VMM with the `perf stat` command and we execute inference operations using the CNN model within the TD while simultaneously monitoring various performance counter metrics from the VMM. For the CIFAR-10 dataset, the CNN model performs the inferencing operation of 10 images, belonging to one of the ten distinct classes. The performance counter events are recorded for each inference operation. This process is repeated 10 times, once for each of the 10 classes, yielding a detailed performance profile for each class. We repeat the same procedure for the CIFAR 100 dataset.

By analyzing HPCs, we can effectively differentiate between the class images undergoing inference within the TD. Specifically, Figure 6 demonstrates that, for CIFAR-10, 42 out of 45 class pairs are distinguishable, while for CIFAR-100, 4,489 out of 4,950 class pairs are distinguishable depending on the branch misses values. The vertical axis of the figure quantifies the total number of distinguishable class pairs for CIFAR-10 (represented in red) and CIFAR-100 (represented in blue). This data underscores the high efficacy of observing the VMMs process HPCs in revealing class distinctions during CNN inference operations within the TD. The distinguishability of class pairs across the majority of models in CIFAR-10 and CIFAR-100 using HPC data emphasizes the potential risk of side-channel attacks exploiting these metrics to infer sensitive information, thereby posing a substantial threat to the confidentiality of the inferences performed within the TD.

5 Conclusion

In this paper, we have put forward a potential side-channel that is present in the platform when Intel TDX technology is in use. Our findings illustrate that performance counters can be exploited to discern some gross quantification of activity states of TDs that can be correlated with activity in VMM. As part of responsible security disclosure, we have discussed the finding with Intel. It is not in the scope of the TDX module to prevent leakage from VMMs process performance counters. We will continue to explore threats to compromise the confidentiality of workloads in Intel Trust Domain Extensions.

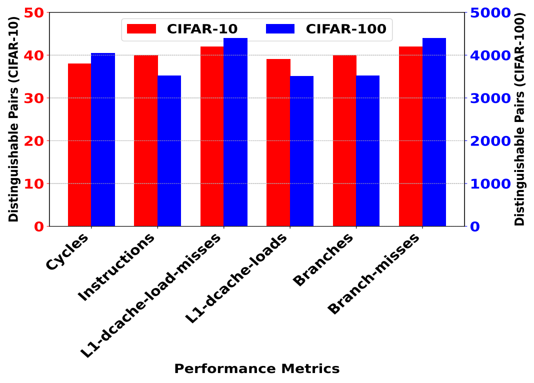


Figure 6: Distinguishable Class Pairs Identified via HPCs in the VMM during CNN Inference on CIFAR-10 (45 Total) and CIFAR-100 (4950 Total) within the Trust Domain

References

[1] Ayaz Akram, Maria Mushtaq, Muhammad Khurram Bhatti, Vianney Lapotre, and Guy Gogniat. Meet the sherlock holmes' of side channel leakage: A survey of cache SCA detection techniques. *IEEE Access*, 8:70836–70860, 2020.

[2] Manaar Alam, Sarani Bhattacharya, Debdeep Mukhopadhyay, and Sourangshu Bhattacharya. Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks. *IACR Cryptol. ePrint Arch.*, page 564, 2017.

[3] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In William Enck and Collin Mulliner, editors, *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017*. USENIX Association, 2017.

[4] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 991–1008. USENIX Association, 2018.

[5] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai. Sgxpectre: Stealing intel secrets from SGX enclaves via speculative execution. *IEEE Secur. Priv.*, 18(3):28–37, 2020.

[6] Dmitry Evtyushkin, Ryan Riley, Nael B. Abu-Ghazaleh, and Dmitry Ponomarev. Branchscope: A new side-channel attack on directional branch predictor. In Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar, editors, *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, pages 693–707. ACM, 2018.

[7] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Comput. Surv.*, 54(6):126:1–126:36, 2022.

[8] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel SGX. In Cristiano Giuffrida and Angelos Stavrou, editors, *Proceedings of the 10th European Workshop on Systems Security, EUROSEC 2017, Belgrade, Serbia, April 23, 2017*, pages 2:1–2:6. ACM, 2017.

[9] Intel Corporation. Intelő software guard extensions (intelő sgx). https://software.intel.com/en-us/sgx, 2017.

[10] Intel Corporation. Intelő trusted domain extensions (intelő tdx). https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html, 2021.

[11] Intel Corporation. Intel trust domain extension linux guest kernel security specification. https://intel.github.io/ccc-linux-guest-hardening-docs/security-spec.html#msrs, 2024.

[12] Yeongjin Jang, Jae-Hyuk Lee, Sangho Lee, and Taesoo Kim. Sgx-bomb: Locking down the processor via rowhammer attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017*, pages 5:1–5:6. ACM, 2017.

[13] Jae-Hyuk Lee, Jin Soo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent ByungHoon Kang. Hacking in darkness: Return-oriented programming against secure enclaves. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, pages 523–539. USENIX Association, 2017.

[14] Moritz Lipp, Andreas Kogler, David F. Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: software-based power side-channel attacks on x86. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 355–371. IEEE, 2021.

[15] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How SGX amplifies the power of cache attacks. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 69–90. Springer, 2017.

[16] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, Ruidong Tian, Chunlu Wang, and Gang Qu. Voltjockey: A new dynamic voltage scaling-based fault injection attack on intel SGX. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 40(6):1130–1143, 2021.

[17] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: Using SGX to conceal cache attacks. In Michalis Polychronakis and Michael Meier, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings*, volume 10327 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2017.

[18] Shubhi Shukla, Manaar Alam, Sarani Bhattacharya, Pabitra Mitra, and Debdeep Mukhopadhyay. "whispering mlaas" exploiting timing channels to compromise user privacy in deep neural networks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(2):587–613, 2023.