# How Trusted Execution Environments Fuel Research on Microarchitectural Attacks

**Michael Schwarz, Daniel Gruss**
Graz University of Technology, Austria
michael.schwarz@iaik.tugraz.at, daniel.gruss@iaik.tugraz.at

***Abstract*—Trusted Execution Environments (TEEs) enabled research in scenarios with highest-privileged attackers with precise control over system and microarchitecture. Insights gained from such attacks facilitated the discovery of non-TEE attacks like Spectre and Foreshadow from within virtual machines. Future research on microarchitectural attacks will continue to draw motivation from TEE threat models.**

**Index Terms: C.1.5.j Support for security < C.1.5 Micro-architecture implementation considerations < C.1 Processor Architectures < C Computer Systems Organization; K.6.5.e Unauthorized access (hacking, phreaking) < K.6.5 Security and Protection < K.6 Management of Computing and Information Systems < K Computing Milieux**

## Introduction

One of the key challenges of system security is to run multiple mutually distrusting programs simultaneously on shared hardware. In the modern connected world with complex systems running programs from various sources, this is more relevant than ever. In particular, content providers have a commercial interest in running software on a system in a tamper-resistant way, a driving factor behind the real-world deployment of Trusted Execution Environments (TEEs). However, TEEs have also been advertised as a key enabler for trusted cloud computing. TEEs isolate a piece of software from the rest of the system while generally still utilizing the same system resources, e.g., CPU cores and DRAM. They can be found on most Intel and ARM systems today.

In this paper, we focus on Intel SGX, a widespread commercial TEE implementation with strong security guarantees. Following from the aforementioned use cases, Intel SGX has an interesting threat model allowing very powerful attackers. Intel SGX allows physical attacks on off-chip devices, e.g., reading or modifying DRAM, in its threat model but also software-based attackers with the highest possible privileges, e.g., malicious operating systems or hypervisors. However, Intel SGX does not protect against side-channel attacks and, thus, enclaves should include side-channel protection.

Building systems of layers requires the abstraction of the complexity of lower layers through so-called architectural interfaces. As an abstraction, these interface definitions are inherently incomplete. Microarchitectural attacks break security assumptions on one layer by exploiting the behavior of lower layers that were not part of the interface definition, and are typically divided into side-channel and fault attacks. Side-channel attacks derive secrets from passively obtained meta-information. Fault attacks induce errors in computations or data on lower layers to bypass security assumptions of higher layers.

Published by the IEEE Computer Society

One important building block of all microarchitectural attacks is the interaction with the microarchitecture. This can be the execution of a single instruction, obtaining a timing source, or setting up the microarchitecture in a certain way. Traditionally, research on microarchitectural attacks assumed unprivileged attackers in native code, in a sandbox (e.g., browser), or on a remote machine; or privileged attackers in a virtual machine. In these scenarios, setting up and interacting with the microarchitecture is usually not trivial and consumes a substantial amount of work. For instance, none of these adversaries knows actual physical addresses and has to extract this information, e.g., via side channels.

As we discuss in this paper, we observed that TEE threat models fueled research on microarchitectural attacks in the past years. TEE threat models allowed researchers to investigate scenarios with highest-privileged attackers without restrictions such as sandboxes or virtualization. We show how this lead to novel insights on microarchitectural attacks that back-propagated to insights both in unprivileged scenarios as well as sandboxes and virtual machines.

We overview a selection of publications presenting attacks on SGX. We show how the SGX scenario is used as a motivation and the implications these works had on follow-up publications. We point out how SGX attack research helped the discovery of Spectre, LVI, and other unprivileged microarchitectural attacks. We argue that the discovery that Foreshadow works from within a virtual machine, which has tremendous relevance for the entire cloud industry, was only a consequence of the discovery of the regular Foreshadow attack targeting SGX enclaves.

Finally, we point out blind spots, future directions for microarchitectural attack research following from recent works on SGX attacks, and for SGX attack research.

## Background

Before we discuss a selection of microarchitectural attack research on TEEs, we provide background on microarchitectural attacks in general, as well as a short introduction to Intel SGX.

### Microarchitectural Attacks

Microarchitectural attacks exploit system behavior resulting from the microarchitectural implementation, *i.e.*, parts that are not specified by the architecture definition. Microarchitectural side-channel attacks exploit leakage of meta-information to infer secrets, while fault attacks induce errors into computations and exploit these to obtain secret information or obtain, e.g., code execution on the device.

A key ingredient to many of these attacks is the measurement of the system state or behavior. Since the microarchitecture is built to optimize performance, CPUs also integrate functionality to measure the performance of applications and, hence, the system state or behavior.

Microarchitectural side-channel attacks typically rely on the existence of a microarchitectural element with the following properties:

**$\mathcal{P}1$** It is shared between the attacker and victim application.

**$\mathcal{P}2$** Its state changes based on the processed data.

**$\mathcal{P}3$** The state can be inferred from (a combination of) side channels.

Fault attacks additionally require:

**$\mathcal{P}4$** The element can be faulted such that it operates with corrupted data.

Microarchitectural attacks only affect applications that use the microarchitectural element. Thus, if the attacker and victim application do not use the same microarchitectural element ($\mathcal{P}1$), the attacker application cannot infer anything about the victim application via the state of the microarchitectural element.

Property $\mathcal{P}1$ generally defines the scope of a microarchitectural attack, as illustrated in Figure 1 for a modern Intel x86 CPU with SMT (simultaneous multithreading, also known as hyperthreading). Some microarchitectural elements are private to the CPU core (e.g., registers), shared among hyperthreads (e.g., L1 caches), shared among all CPU cores (e.g., some last-level caches), or even shared among all CPUs (e.g., main memory) if there are multiple CPUs on a mainboard. Thus, the domain in which the element is shared defines the domain in which the attacker can mount an attack.

Some microarchitectural elements optimize the performance of an application independently of the

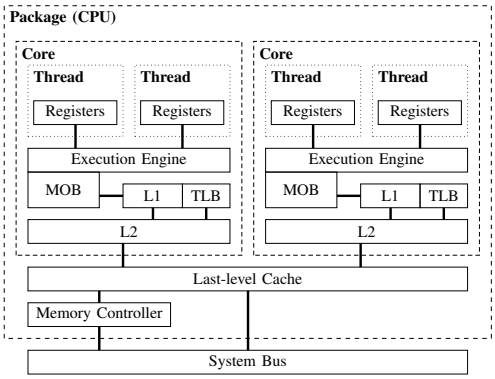


**Figure 1.** The multiple scopes of shared resources: private per thread, shared among threads, shared across cores, and global resources visible to the entire system.

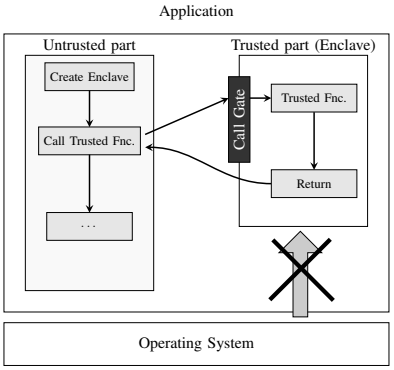


**Figure 2.** In the SGX model, applications are split into a trusted (enclave) and an untrusted (host) part. The hardware prevents any access to the trusted part. The only communication between enclave and host uses predefined `ecall`s and `ocall`s.

processed data, e.g., the hardware AES implementation AES-NI. However, many microarchitectural elements implement performance optimizations that depend on the processed data ($\boldsymbol{\mathcal{P}2}$). For example, a data cache reduces the access latency for recently used data, *i.e.*, data the application used before. For these elements, the current internal state is influenced by previously processed metadata and data, and observing the internal states leaks information.

As microarchitectural elements usually do not provide an interface to query their state, an attacker can only observe the internal state via side channels ($\boldsymbol{\mathcal{P}3}$). Many microarchitectural side-channel attacks use timing as an information source. That is, based on the execution time of an operation, an attacker can infer information on the internal state of a microarchitectural element.

Sometimes the timing cannot directly be measured, but an indirection is required to obtain side-channel information. For example, an attacker can infer information about the internal state of a control-flow predicting mechanism, such as the branch predictor, by observing the memory access time of subsequent instructions.

For fault attacks, the crucial requirement is that the targeted (shared) microarchitectural element must be susceptible to faults ($\boldsymbol{\mathcal{P}4}$). These faults may be persistent or temporary state changes ($\boldsymbol{\mathcal{P}2}$) but have to be triggerable by an attacker. The attacker also requires $\boldsymbol{\mathcal{P}1}$ to induce faults into a microarchitectural element used by the victim application and, hence, influence the victim's operation. While $\boldsymbol{\mathcal{P}3}$ is not strictly necessary, many fault attacks use side-channel information to precisely induce faults subsequently.

Summarizing, for microarchitectural side-channel attacks, an attacker relies on a microarchitectural element typically fulfilling $\boldsymbol{\mathcal{P}1}$, $\boldsymbol{\mathcal{P}2}$, $\boldsymbol{\mathcal{P}3}$, and for fault attacks also $\boldsymbol{\mathcal{P}4}$. Then, the observed leakage or induced faults can be used to attack a victim application or the system.

### Intel SGX

Intel Software Guard Extension (SGX) is an x86 instruction-set extension introduced in 2015 with the Intel Skylake microarchitecture for isolating trusted code from regular untrusted applications. Applications are split into a trusted enclave part and an untrusted application part (cf. Figure 2). The CPU fully isolates the trusted enclave, and neither the application nor the operating system can access the enclave's memory. Furthermore, to protect against bus-probing attacks on the DRAM bus and cold-boot attacks, the memory range used by SGX is encrypted via transparent memory encryption.

The application and enclave can only communicate through a well-defined interface. Using the `eenter` function, applications can call functions provided by the enclave. The hardware prevents any other attempt to access the enclave or the enclave's memory. In the attacker model of Intel SGX, only the hardware is trusted. All software, including the operating system, is assumed to be

potentially compromised and, therefore, untrusted. This is particularly important for the digital rights management use case, but it is also helpful for implementing confidential cloud computing where customers do not even have to trust the cloud provider.

Although SGX enclaves run native code, there are several restrictions for enclaves to reduce the attack surface. Enclave code cannot use any I/O operations, including syscalls. Thus, any communication with the operating system is only possible via the untrusted application as a proxy. Moreover, a range of instructions is not supported as it could either ease exploitation or would introduce implicit trust on the operating system or hypervisor, such as `rdtsc`. Hence, Intel SGX also has an impeding effect on microarchitectural attacks.

## The Intel SGX Threat Model

With TEEs, the threat model is different than for most other systems. Intel SGX has a very small trusted computing base (TCB), which consists only of the CPU. Enclave memory is inaccessible to other enclaves, user programs, and even the OS. It is placed in a physically contiguous encrypted and integrity-protected block in DRAM, the *EPC* (enclave page cache). Hence, any hardware or software component other than the CPU and the enclave can be malicious. Thus, a highest-privileged attacker is commonly assumed for attacks.

Intel explicitly states that SGX provides no protection against side-channel attacks but that it is the enclave developer's responsibility to address side-channel attack concerns. However, this introduces an asymmetry as the attacker now has stronger capabilities (e.g., native host kernel privileges) than the defender, running inside the SGX enclave with regular user privileges. Hence, applications that may be side-channel-secure in a non-SGX scenario may be vulnerable in the SGX scenario.

## Attacks on Intel SGX

As SGX enclaves share the hardware with the rest of the system, it is not surprising that microarchitectural attacks can be mounted on enclaves ($\mathcal{P}1$). Although the CPU is considered the TCB, Intel only consideres architecturally visible changes in scope for protection. Hence, the state of microarchitectural elements, such as caches, is not considered as an attack vector for enclaves. With the threat model in mind, we provide an overview of privileged and unprivileged microarchitectural attacks on Intel SGX. We categorize the attacks into three distinct categories: side-channel attacks, transient-execution attacks, and fault attacks. Table 1 provides an overview over all attacks discussed in this section.

### Side-Channel Attacks

While side-channel attacks are the primary attack vector to attack SGX enclaves, they do not directly leak sensitive data. With a side-channel attack, an attacker observes meta-information, e.g., memory-access patterns, and derives information from that meta information. The meta-information is observed through a microarchitectural element shared between the trusted and untrusted part of the enclave. We classify side-channel attacks on SGX based on which shared microarchitectural element is exploited.

**Page-Table Attacks** Page-table attacks target the mechanism which translates virtual to physical addresses, including the page-miss handler, page-table walker, and TLB. While the memory used by an enclave (EPC) is encrypted and can thus not be observed, the corresponding page tables to map EPC pages are set up by the (untrusted) operating system, as this cannot be done solely in hardware. The hardware only keeps track of the meta data to ensure that one EPC page is not mapped by multiple enclaves.

A consequence of this design is that a malicious operating system can observe interactions of an enclave with the virtual to physical address translation. Xu et al. [20] exploited that the operating system can simply unmap pages used by the enclave and observe a page fault when the enclave tries to access this page. Thus, by keeping only the active code page mapped and all other code pages of interest unmapped, an attacker can infer the control flow on a page-size granularity.

Gyselinck et al. [6] demonstrated that a similar attack can be mounted on 32-bit SGX enclaves using x86 segmentation. By changing the segment limits, an attacker can cause the enclave to fault when trying to execute code that is outside of the

**Table 1. Microarchitectural attacks on Intel SGX can be categorized into side-channel attacks (SC), transient-execution attacks (TE), and fault attacks (FA).**

| Attack | Microarchitectural Element | | | | | | Type | Attacker | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | LFB | Cache | DRAM | Paging | Branch Predictor | ALU | | Unprivileged | Privileged | Cross-Enclave |
| Xu et al. [20] | | | | ✓ | | | SC | | ✓ | |
| Schwarz et al. [13] | | ✓ | | | | | SC | ✓ | | ✓ |
| Götzfried et al. [5] | | ✓ | | | | | SC | | ✓ | |
| Moghimi et al. [11] | | ✓ | | | | | SC | | ✓ | |
| Wang et al. [19] | | ✓ | ✓ | ✓ | | | SC | | ✓ | ✓ |
| Van Bulck et al. [17] | | ✓ | | ✓ | | | SC | | ✓ | |
| Lee et al. [9] | | | | | ✓ | | SC | | ✓ | |
| Moghimi et al. [10] | | ✓ | | | | | SC | ✓ | ✓ | |
| Dall et al. [3] | | ✓ | | | | | SC | | ✓ | |
| Gyselinck et al. [6] | | | | ✓ | | | SC | | ✓ | |
| Evtyushkin et al. [4] | | | | | ✓ | | SC | ✓ | ✓ | |
| Van Bulck et al. [15] | | ✓ | | ✓ | | | TE | ✓ | ✓ | ✓ |
| Chen et al. [2] | | ✓ | | | ✓ | | TE | | ✓ | |
| Koruyeh et al. [8] | | ✓ | | | ✓ | | TE | ✓ | ✓ | |
| Schwarz et al. [14] | ✓ | | | ✓ | | | TE | ✓ | ✓ | |
| Van Schaik et al. [18] | ✓ | | | ✓ | | | TE | ✓ | ✓ | |
| Van Bulck et al. [16] | ✓ | ✓ | | ✓ | | | TE | ✓ | ✓ | |
| Murdock et al. [12] | | | | | | ✓ | FA | | ✓ | |
| Kenjar et al. [7] | | | | | | ✓ | FA | | ✓ | |

segment limit, resulting in a spatial granularity of one page, *i.e.*, 4 kB.

Van Bulck et al. [17] and Wang et al. [19] showed that by observing the access and dirty bit in a page-table entry, an attacker can get the same information as with page faults, however, in a more stealthy way. Moreover, Van Bulck et al. [17] showed that in addition to this architecturally available information, an attacker can also mount a cache attack, such as Flush+Reload or Flush+Flush on a page-table entry, to infer the page-access pattern of an enclave, resulting in a high-resolution microarchitectural attack.

**DRAM Attacks** The DRAM is another microarchitectural element that can be abused for microarchitectural side-channel attacks. DRAM modules contain row buffers that act as caches for the rows in the DRAM. Every read requires the data to be copied from the destination row to this buffer. As with CPU caches, accessing data that is already in the row buffer results in faster access times.

Wang et al. [19] demonstrated that the DRAM side channel can also be leveraged to attack SGX enclaves from concurrently running SGX enclaves. All enclaves share the same physical range of main memory, and thus potentially also the same DRAM bank and row buffer. Hence, a malicious enclave can use the DRAM side channel to spy on memory access patterns of a concurrently running enclave. The spatial granularity of this attack depends on the actual DRAM configuration and ranges from 512 B to 8 kB. As the main memory is typically shared among all CPUs (even in separate sockets), DRAM-based attacks can be mounted across CPUs, e.g., if there are multiple CPUs on a mainboard.

**Cache Attacks** Cache attacks exploit the fundamental property of caches that data residing in the cache can be accessed faster than data not residing in the cache. There are different methods of how an attacker can abuse this property to infer whether a specific memory location resides in the cache.

Cache attacks can be divided into three main categories. For the first type of cache attacks (Evict+Time), the attacker modifies the cache state and monitors the runtime of the victim. In the second type of cache attacks (Prime+Probe), the attacker brings the cache into a known state and monitors whether the victim execution influenced this known state without directly observing memory accesses of the victim. In the third type of cache attacks (Flush+Reload), the attacker measures cache-state changes directly on memory, which is shared with the victim, e.g., shared libraries.

As SGX enclaves do not directly share any memory with the remaining untrusted system, Flush+Reload is not directly applicable to enclaves, only to page tables. In contrast, Prime+Probe attacks do not require any form of shared

memory or access to the victim. Thus, they have also been used to attack TEEs such as Intel SGX. Götzfried et al. [5] and Moghimi et al. [11] showed how a malicious operating system can leverage Prime+Probe to leak secrets from SGX. As the operating system is responsible for mapping the physical pages of the enclave, and the cache set is determined by the physical address, a malicious operating system can easily monitor a cache set for changes caused by the enclave execution. Schwarz et al. [13] also showed that even an unprivileged attacker can mount a Prime+Probe attack on an enclave. Furthermore, Schwarz et al. [13] showed that a Prime+Probe attack can be mounted from inside an SGX enclave on the host. Schwarz et al. [13] and Wang et al. [19] also demonstrated a cross-enclave Prime+Probe attack.

In all these cases where Prime+Probe is mounted on an enclave, an attacker can infer memory-access patterns with a cache-set granularity of the last-level cache, *i.e.*, typically 2 kB. As the last-level cache on Intel CPUs is shared between all CPU cores, Prime+Probe can be mounted across cores.

Moghimi et al. [10] showed MemJam, an intra-cache-line attack on enclaves that exploits performance degradation caused by false dependencies between memory reads and writes. MemJam [10] has a spatial granularity of 4 bytes within a cache line.

To prevent an attacker from leaking secrets via cache attacks, an enclave has to ensure that there are no secret dependent memory accesses, not even within a cache line.

**Predictors** Modern CPUs use several prediction mechanisms to avoid pipeline stalls. As predictions are based on previously observed data, an attacker can often infer information from observing the predictions.

One prediction mechanism used for microarchitectural attacks is the memory-aliasing prediction, also known as memory disambiguation. This prediction mechanism predicts whether a memory load has to go to the memory or whether it can consume a previous store. We discovered that the forwarding of stores misses permission checks, allowing to stealthily de-randomize the address space of SGX enclaves.

Branch predictors have not only been used for Spectre but also as side-channel attacks. Evtyushkin et al. [4] and Lee et al. [9] exploited the branch predictor to infer whether a branch is taken or not taken inside an enclave. As a consequence, any secret-dependent branch inside an enclave leaks the corresponding secret.

### Transient-execution Attacks

Transient-execution attacks are a class of microarchitectural attacks exploiting out-of-order and speculative execution of modern CPUs to leak data. In contrast to side-channel attacks, transient-execution attacks leak actual data and not meta information. Transient-execution attacks rely on computations, which were never intended in an application's control flow. Such computations by transient instructions can be a result of mispredictions in the control or data flow, or out-of-order execution after an exception. While these transient instructions are never committed to the architectural state, they may show side effects in the microarchitectural state. These side effects can then be made visible in the architectural domain using traditional side-channel attacks.

**Meltdown-type Attacks** Meltdown is a class of transient-execution attacks exploiting transient instructions caused by out-of-order execution after an exception. On affected CPUs, memory loads triggering an exception, such as a page fault, still return data, which can be used in the transient execution to encode it in a microarchitectural state. After the exception is handled (or suppressed), an attacker can again use a side channel to transfer the microarchitectural state into the architectural state.

The first Meltdown attack exploited the lazy enforcement of the user-accessible permission in the page table, *i.e.*, the bit defining whether a virtual address belongs to the kernel or user space. While this original Meltdown attack leaks kernel data, and through kernel mappings arbitrary physical memory, it cannot leak values from SGX enclaves. However, Van Bulck et al. [15] showed Foreshadow, a Meltdown-type attack that exploits the lazy enforcement of the present bit in the page table, *i.e.*, the bit defining whether a virtual address is valid. With this attack, an attacker can

leak arbitrary values from SGX enclaves, resulting in a complete breach of confidentiality.

Subsequent works have shown that also other types of exceptions can be exploited for Meltdown type attacks [14]. Two of these attacks, ZombieLoad [14] and RIDL [18], leak data from various internal CPU buffers that are currently used by the current CPU core. This also includes values that are used within SGX enclaves. While these attacks are not as targeted as Foreshadow [15], they can even be mounted on machines with hardware fixes for Foreshadow. Due to the strong attacker model of SGX, an attacker can reliably leak specific data from an enclave by using enclave-execution frameworks that we discuss later on.

*Load Value Injection.*
Load Value Injection (LVI) [16] is a transient-execution attack that turns Meltdown around. Instead of exploiting exceptions to leak data from microarchitectural buffers, LVI induces an exception in the victim application to transiently inject data. The out-of-order execution within the victim then continues with the attacker-controlled injected data. LVI can, e.g., be used to hijack the control flow in the victim to encode secrets inside a microarchitectural element, which can later on be recovered by the attacker.

As an attacker can arbitrarily change the page tables for enclaves, inducing a fault for any instruction loading data from memory is straightforward. Hence, potentially all memory-load operations inside SGX can be exploited to transiently work with attacker-controlled data.

**Spectre-type Attacks** Spectre is a class of transient-execution attacks exploiting control- and data-flow mispredictions of CPUs. By triggering such a misprediction, Spectre attacks transiently execute code to access data that is architecturally accessible but never reached. Due to this execution path, Spectre attacks can potentially access sensitive data of the application. Subsequently, Spectre attacks encode the accessed data in the microarchitectural state, e.g., in the cache. This is done using a so-called Spectre gadget, which has to be found in the transiently executed code path. Similarly to return-oriented programming, such gadgets are likely already in the victim application and only have to be found by the attacker. A naïve Spectre gadget is as simple as a secret-dependent array access, causing a memory location corresponding to the accessed data to be cached. An attacker can then rely on traditional side-channel attacks to transfer the microarchitectural state to the architectural state.

Spectre attacks have not only been shown on regular applications, the kernel, and hypervisors, but also on SGX enclaves. Koruyeh et al. [8] presented Spectre-RSB, which exploits the prediction of function returns. They showed that this is not only applicable to regular application but also SGX enclaves. Chen et al. [2] analyzed widespread SDKs for enclave development and found multiple exploitable Spectre gadgets for different Spectre variants.

Spectre attacks on SGX enclaves show that software mitigations against Spectre, such as memory fences and retpoline, have to be applied to enclaves as well. Otherwise, an attacker can exploit Spectre attacks to break the confidentiality of enclaves. While this is feasible for some Spectre variants, some mitigations rely on the operating system and can thus not easily be used inside an enclave. Although it is possible to prevent these variants in microcode (the CPU's firmware), the performance impacts on SGX would be non-negligible.

### Fault Attacks

Fault attacks try to manipulate computations to divert control or data flow or directly manipulate code or data. Rowhammer attacks are the most prominent example of software-based fault attacks. They exploit that a burst of accesses to a DRAM cell can lead to bit flips in a physically adjacent cell. Hence, the attacker may gain precise control over the location of the bit flip. However, the EPC region of SGX is securely encrypted and integrity-protected. Consequently, Rowhammer attacks on SGX memory can only lead to a failure of the integrity check.

Two recent attacks called Plundervolt [12] and V0LTpwn [7] overcome this limit by not inducing faults in DRAM, but directly in the CPU's core, where the data is already decrypted and not integrity protected anymore. It is based on reducing the supply voltage of the CPU for a very short time frame to a range where the

CPU largely still operates correctly, but bit flips occur for certain operations with a low probability. Only when the attacker triggers many of these operations in a victim computation inside an SGX enclave, a bit flip occurs in the computed result, e.g., enabling differential fault analysis.

## Impact of TEE Threat Models on Microarchitectural Attack Research

Trusted-execution environments had a significant impact on microarchitectural attack research. There have been many new attacks but also frameworks to simplify attack research. There are mainly two reasons why TEE threat models had such an impact. First, as privileged attackers are in the SGX threat model, it became possible to analyze previously ignored elements and interactions. Second, SGX is highly controllable for privileged attackers, often simplifying the first attack prototypes. Hence, we see SGX as a stepping stone for non-SGX attacks impacting everyone and not just SGX users.

### Frameworks

Several attack frameworks leverage, within the SGX threat model, arbitrary hardware features to control the environment when running enclaves. These frameworks focus on precise execution control for the enclave, similar to single stepping in a debugger. Well known examples are SGX-Step[1], PTEditor[2], and MicroScope[3]. SGX-Step allows an attacker to either advance one instruction at a time (single stepping), or even transiently execute one instruction arbitrarily often (zero stepping [15]). MicroScope achieves a similar goal using page faults.

While these frameworks were all developed to research microarchitectural attacks on SGX, they have also been used to analyze transient-execution attacks in unrealistic non-SGX scenarios. More precisely, as all frameworks required operating-system privileges, they cannot be used for actual attacks on user-space applications or the kernel. While this limits their real-world use case to attacks on SGX and hypervisors, they simplify the task of prototyping attacks and testing whether systems are affected by certain attacks.[4]

### Privileged Attackers

Table 1 shows that many attacks on enclaves assume or even require a privileged attacker, e.g., to manipulate scheduling or page tables. A highest-privileged attacker can minimize unrelated influences, such as other running applications or interrupts, but also repeating specific victim code as often as required. As a result, attackers can get fine-grained attack traces from enclave victims.

In non-TEE threat models, attackers typically do not run at the highest privileges in the system, *i.e.*, higher than that of the victim. Hence, with the strong threat model of TEEs, it became possible to analyze microarchitectural effects, which can only be observed by a highest-privileged attacker. This includes, e.g., the manipulation of interrupts, page tables, model-specific registers, or voltage.

TEE threat models made it possible to publish insights on these microarchitectural elements and interactions. Without a strong attacker model as provided by SGX, it is hard to justify why such an analysis is relevant for security research. However, the insights gained from privileged microarchitectural attacks are not only applicable to attacks on SGX. In fact, some of them turned out to be a stepping stone for other attacks with a much broader impact.

### Stepping Stone

The SGX threat model motivated research on branch predictors [4, 9]. In these attacks, a privileged attacker observes the influence of branches inside an enclave on branch-predictor states. Mounting these attacks required a good understanding of the microarchitectural internals of branch predictors. Hence, these attacks first required reverse engineering of branch predictors, leading to valuable insights into how branch predictors work. These insights have subsequently been used in Spectre attacks in a non-TEE use case. The impact of Spectre outside of TEEs can be seen in the tremendous performance impacts caused by the deployed mitigations. Depending on the exact Spectre variant, performance overheads between $5\,\%$ and $74.8\,\%$ per mitigated variant are measured in real-world software.

[1] https://github.com/jovanbulck/sgx-step
[2] https://github.com/misc0110/PTEditor
[3] https://github.com/dskarlatos/MicroScope
[4] https://github.com/IAIK/transientfail

Similarly, the Foreshadow attack [15] targeted SGX enclaves. For this attack, an attacker has to map an enclave page and clear the present bit in the according page-table entry. A faulting access to this mapping leads to data leakage in the transient domain, similar to Meltdown. At first, this attack only made sense in the context of TEE threat models, where an attacker has full control over a native host page-table entry. A subsequent analysis of the root cause of Foreshadow showed that Foreshadow also works from within virtual machines, attacking co-located virtual machines and the hypervisor. While not apparent at first glance, an attacker inside a malicious virtual machine also has control of page-table entries. However, this extremely powerful attack from within virtual machines would have likely not been discovered without the original Foreshadow attack on SGX. Foreshadow forced cloud providers to either disable hyperthreading, or provide both hyperthreads, *i.e.*, the entire CPU core, to the same virtual machine. This basically halved the number of possible small virtual machines, which, before Foreshadow, used only one hyperthread. Additionally, the required patches lead to another performance impact of a few percent, according to Phoronix.[5]

LVI [16] was also first analyzed on SGX, as SGX allows to easily induce faults into the victim application by modifying page-table entries. Especially one variant, LVI-NULL, is reliably exploitable when targeting SGX but considered infeasible in a non-SGX scenario. However, from the main insight that hardware patches for Meltdown simply replace the leaked values with 0, Canella et al. [1] developed a method to break the randomization of the kernel image from unprivileged code, and even from the browser. Preventing the attack requires more changes to the memory system of operating systems, which also incurs a small performance overhead.

Another widely used building block that emerged from research on microarchitectural attacks on SGX is the method for getting high-resolution timestamps in the absence of a high-precision timer. As SGX does not provide access to the high-resolution timer, the first microarchitectural attack from within SGX [13] presented a highly optimized timing primitive based on multithreading. This timing primitive was later used not only for attacks from within SGX but also in other scenarios where no high-resolution timer is available, e.g., on AMD CPUs and in browsers.

Spectre, Foreshadow, and LVI are entirely new attacks from the class of transient-execution attacks. Both attacks heavily relied on research initially focussing on microarchitectural attacks on TEEs. Moreover, certain primitives developed specifically for SGX have later on been used as building blocks for attacks not targeting enclaves. Hence, we can expect more attacks not targeting enclaves, originating from the insights gained by research on microarchitectural attacks on TEEs.

## Consequences for TEEs and Enclave Development

The past and ongoing stream of research shows that developing secure enclaves is substantially more difficult than developing a secure application. The main reason is the more powerful attacker that is assumed in the SGX threat model owing to the ambitious goals of SGX. In the current situation, enclave developers ideally are also experts on microarchitectural attacks and familiar with a frequently extended state of the art. While, e.g., Intel published guidelines for developing side-channel resilient software[6], we believe that future research should develop techniques to simplify the development of secure enclaves and remove this burden from developers. This is not only relevant for the microarchitectural attacks we described but also for various other problems and vulnerabilities around SGX, e.g., attacks exploiting information leakage through architectural interfaces, as well as CPU bugs that expose parts of the SGX memory to untrusted software.

One feature that makes it particularly challenging to maintain the SGX security guarantees is hyperthreading. Hyperthreading assigns instruction streams from multiple independent workloads dynamically to the same set of execution units of a core. This inherently introduces timing side channels revealing whether an ex-

[5] https://www.phoronix.com/scan.php?page=article&item=l1tf-foreshadow-xeon&num=1

[6] https://software.intel.com/security-software-guidance/insights/guidelines-mitigating-timing-side-channels-against-cryptographic-implementations

ecution unit is occupied. But also the sharing of these microarchitectural resources introduces problems as hyperthreads may transiently pick up incorrect data. As a consequence to the risks of hyperthreading, enclave developers even have the possibility to prevent enclaves from executing if hyperthreading is enabled. More static partitioning, as implemented for the store buffer, appears to be more secure. As of today, hyperthreading is essential for the cloud to the point that it cannot be disabled, disabling hyperthreading shows average performance losses between 0 % and 30 % according to Red Hat.[7] Consequently, future research should develop techniques to support secure hyperthreading.

In addition to securing existing TEEs, there are academic proposals for different TEEs, e.g., Sanctum and Keystone. While only demonstrated for RISC-V CPUs, they show that TEEs can be designed with stronger protection against some of the attacks shown in Table 1, especially page-table attacks.

## Conclusion

We discussed a selection of state-of-the-art microarchitectural attacks on SGX and analyzed how they influenced the general landscape of microarchitectural attacks. The SGX threat model often simplifies the initial analysis of attack vectors and microarchitectural effects. We showed that as a result, attacks on TEEs are often the foundation for more sophisticated attacks not targeting TEEs. We highlighted how SGX attack research, relevant to a smaller group of SGX users, was a stepping stone for non-SGX attacks impacting everyone, leading to widely deployed countermeasures. In particular, we highlighted how research on branch predictors in the SGX threat scenario helped to discover Spectre, a much broader issue of modern computer systems. Even more clearly, we argue that the discovery that the Foreshadow attack works from within a virtual machine, leaking any data across isolation boundaries on cloud computers, was a direct consequence of the research on the regular Foreshadow attack on SGX enclaves. Both Spectre and Foreshadow had a significant real-world impact in many aspects. These examples, amongst others, show that it is important to continue research with strong attacker models, like the one of SGX, as insights will propagate back and then have a real-world impact outside of TEEs, impacting a much larger number of users.

[7] https://access.redhat.com/security/vulnerabilities/L1TF-perf

## REFERENCES


1. Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl, and Daniel Gruss. KASLR: Break It, Fix It, Repeat. In *AsiaCCS*, 2020.
2. Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution. In *EuroS&P*, 2019.
3. Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks. In *CHES*, 2018.
4. Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. In *ASPLOS*, 2018.
5. Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache Attacks on Intel SGX. In *EuroSec*, 2017.
6. Jago Gyselinck, Jo Van Bulck, Frank Piessens, and Raoul Strackx. Off-limits: Abusing legacy x86 memory segmentation to spy on enclaved execution. In *ESSoS*, 2018.
7. Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. V0LTpwn: Attacking x86 Processor Integrity from Software. In *USENIX Security Symposium*, 2020.
8. Esmaeil Mohammadian Koruyeh, Khaled Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. Spectre Returns! Speculation Attacks using the Return Stack Buffer. In *WOOT*, 2018.
9. Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In *USENIX Security Symposium*, 2017.
10. Ahmad Moghimi, Thomas Eisenbarth, and Berk Sunar. MemJam: A false dependency attack against constant-time crypto implementations in SGX. In *CT-RSA*, 2018.
11. Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How SGX amplifies the power of cache attacks. In *CHES*, 2017.
12. Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. In *S&P*, 2020.
13. Michael Schwarz, Daniel Gruss, Samuel Weiser, Clémentine Maurice, and Stefan Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks . In *DIMVA*, 2017.
14. Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *CCS*, 2019.
15. 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 *USENIX Security Symposium*, 2018.
16. Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *S&P*, 2020.
17. Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without

page faults: Stealthy page table-based attacks on enclaved execution. In *USENIX Security Symposium*, 2017.
18. Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue in-flight data load. In *S&P*, 2019.
19. Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *CCS*, 2017.
20. Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *S&P*, 2015.