Background

TEEs

Trusted Execution Environments (TEEs) such as Intel SGX aim to guarantee confidentiality and integrity to programs running within them via hardware support for memory encryption, physical isolation, and controlled interfaces between untrusted and trusted code. Compared to other software security mechanisms, they adopt a strong threat model of a potentially malicious operating system or hypervisor, and prevent both information leakage to the external environment, and interference from said environment.

Controlled Channel Attacks

Unfortunately, for the past decade, they have been riddled with a multitude side-channel attacks that have leaked secrets using quirks in CPU microarchitecture, including Malware Guard Extension, SGAxe, ÆPIC leak, Foreshadow, MicroScope replay attacks, SGX-Step, Load Value Injection, various memory side-channels and more. One particular class of attacks targeting Intel SGX is known as controlled channel attacks, which were proposed in 2015 at the IEEE Symposium on Security and Privacy. In this attack, a malicious operating system unmaps page table entries for the enclave code. When the enclave attempts to read memory on a page that has been unmapped, the CPU triggers a page fault, causing what is known as an Asynchronous Exit (AEX) where the (untrusted) operating system receives the faulting page number. The attack leverages the fact that if the page numbers being accessed by the enclave are dependent on a secret predicate, the operating system can learn the control flow of a program by observing the page fault traces. Here is one example from the paper.

If WelcomeMessageForMale

and WelcomeMessageForFemale

lie on different (usually 4KiB) pages A and B, the malicious OS can unmap both pages, and use the faulting page number to learn the value of input s.

Here is another example from the paper. Even if one page has the code for multiple functions, looking at the sequence of page faults allows one to determine control flow. In the above example, a sequence of A → B → D

implies a function call sequence of f1 → f2 → f4

, whereas the sequence A → C → D

implies f1 → f3 → f5

, based on consulting the interprocedural control flow graph (f3

calls f5

, but but not f4

).

Mitigations

In the years following the release of Controlled Channel Attacks, there have been a slew of mitigations to the attack.

One defense known as T-SGX relies on Intel Transactional Synchronization Extensions (TSX), an extension to the x86 Instruction Set Architecture that enables support for hardware transactional memory. Using this, one can run security-critical code within a transaction, which is a set of instructions that retire in an atomic fashion. That is to say, either all the instructions complete or if the transaction is interrupted for any reason by a page fault, all of the instructions are aborted. In particular, if a page fault occurs during a transaction, the operating system is not notified, thus preventing control flow leakage.

However, due to a bug in the implementation of how Intel TSX handles page faults that was used to break kernel address space layout randomization (KASLR), this feature was disabled on many Intel CPUs, rendering moot all the defenses that relied on it.

Finally in 2023, a team of academic security researchers collaborated with Intel to come out with AEX-Notify, a feature in Intel SGX which allows enclave applications to write custom exception handlers for AEXes. When code execution in the enclave resumes after an AEX, the AEX Notify handler is called, which developers can use to enforce a security policy, which could range from aborting the enclave, to flush/loading secret memory to thwart cache and page fault-based side channel attacks. The handler is given an exception vector that gives it precise information about the cause of the exception, including the faulting page number if the AEX was triggered by a page fault. While this is a remarkable step in the right direction, it cannot detect extended page table (EPT) faults, which are relevant if the enclave is being run in a virtual machine.

What about Intel TDX?

If you are new to Intel TDX or VM-based TEEs, feel free to read myblog post, or peruse Awesome TDX, a collection of explanatory articles on TDX my colleagues and I at Flashbots have collated.

Intel has learned a lot from its experience mitigating a slew of SGX side-channel attacks in developing its VM-based TEE offering called Trust Domain Extensions (TDX). In particular, trust domain (confidential VM) memory is handled by the trusted TDX Module using Secure Extended Page Tables (EPTs), which map Guest Physical Addresses (GPAs) to Host Physical Addresses (HPAs).

However, in an interesting twist, they exposed two commands that an untrusted hypervisor / virtual machine monitor (VMM) can use to unmap and remap Guest Physical Addresses to Host Physical Addresses: TDH.MEM.RANGE.BLOCK

and TDH.MEM.RANGE.UNBLOCK

, respectively. These commands can be sent by specifying the corresponding leaf function number, Guest Physical Address (GPA) range and TD identifying information and invoking the SEAMCALL

instruction, identified by the joint Google-Intel security review as opening a side-channel.

Ostensibly, this was done to allow hypervisors to reclaim memory temporarily from one VM to allocate it to another VM when under memory pressure. However, I argue this essentially takes us back to Intel SGX's situation in 2015, with just one difference. In Intel SGX, the malicious operating system is in the same address space as the victim enclave being attacked, whereas in TDX, Guest Virtual Address (GVA) to Guest Physical Address (GPA) mappings are managed by the trusted Guest OS. Therefore, an attacker needs to learn GVA → GPA mappings for pages which are loaded in a secret-dependent manner.

How To Leak A Secret

Consider a program with the following pseudo-code, where b is a secret input.

if secret bit b is 1 access pages 1-4 else access pages 5-8

[

|1729.2768211920527x809.3968186483015

1600×749 177 KB

](https://collective.flashbots.net/uploads/default/original/2X/8/82b4f3d7faa0795a3504eacb9d29e43f4c2eebd4.png)

1. Offline Phase

a. Run the program with bit = 1.

b. Block all pages in the program from the hypervisor.

c. Observe the faulting guest physical addresses.

d. Repeat steps a → c, with bit b = 0.

[

|1639.5232558139533x788.8072132246225

1600×770 151 KB

](https://collective.flashbots.net/uploads/default/original/2X/4/40e62c36c4f615b12588145ba0ab520901ce3e8b.png)

1. Online Phase

a. Run the program with a randomly chosen value for b

.

b. Block all pages in the program from the hypervisor.

c. Observe the faulting guest physical addresses.

d. Use the page fault trace from the offline phase to determine the value of b.

If one can assume every page of the program is loaded sequentially by the operating system into memory when application within the Trust Domain is launched, one can construct a table mapping virtual page numbers to physical page numbers. Then, unmap all pages, and extract the page fault trace to determine the value of b. This is similar to techniques used in Heckler (Benedict Schlüter et al., USENIX Security 2024) and Cache Template Attacks (Daniel Gruss et al., USENIX Security 2015) to determine runtime data and control flow based on pre-configured templates or profiles of activity.

Countermeasures

1. Removing or restricting the scope of TDH.MEM.RANGE.BLOCK

and TDH.MEM.RANGE.UNBLOCK

instructions

Intel could entirely remove these instructions, or restrict them to only work on Guest Physical Addresses for shared (public) memory and not private (secret) memory. Since availability is a non-goal for Intel TDX, one point on the feasible design space includes allowing a TD to hold on to arbitrary amounts of memory, even if that slows down other TDs.

1. Memory Capped Trust Domain configurations

If the TDX module configures a static maximum memory limit per TD, we could prevent more TDs from being created that would exceed the existing physical memory capacity. This would obviate the need for the hypervisor to migrate memory. A common special case would be a single TD configuration. If we could attest to how many TDs were allowed by the TDX Module for use cases where only 1 security domain was needed, these two instructions would no longer be necessary and could be disabled.

1. Application-level mitigations

Trust Domain applications or VM images could include a registration or heartbeat protocol with an external entity to measure execution progress milestones. Slow progress or repeated registration attempts could be detected by the notification receiving party as an indication that the enclave is under attack, and cause other parties to apply penalties to the TEE infrastructure providers. For example, if we are running a validator node for a proof-of-stake blockchain such as Ethereum in TDX, they could be slashed for attempting to attack or leak information from the EVM execution. Finally, software could use data structures based on Oblivious RAM to obfuscate memory access patterns.

Unfortunately, TDX-compatible hardware will only load code signed by Intel, which means the implementation of the microhypervisor i.e. the (trusted) TDX Module, is fixed by Intel and cannot be modified or extended.

The Case for TDExit-Notify

In place of point mitigations that deal with the page blocking leaf functions, I believe it makes sense to expose TDExit-Notify, an exception handler for TDs that allows applications to flexibly apply security policies based on domain specific constraints. This is especially given page blocking side-channels are not the only type of interrupt-based attack.

While developers using TDX just for integrity (verifiable computation) may not need to protect against these, however, those relying on it for confidential execution of security-critical code might need to prefetch/flush memory regions or abort the TD's execution based on heuristics to thwart leakage. Applications could configure a threshold on the benign number of acceptable page faults above which they assume the TEE is under attack. While exploiting side-channel channels is a specialized art and science, given increasing adoption of TEEs by web3 firms with significant capital at stake, today's adversaries have more than enough incentive.

Further Reading

1. Awesome TDX – collection of TDX-related resources collated by the Flashbots team
2. Intel TDX Documentation – includes an overview of TDX, the ISA extensions specification and application binary interface specification
3. Intel TDX Security and Side-Channels – more information about TDX security and other side-channels not covered here