Trusted execution environments have emerged as a promising alternative to the use of Zero-Knowledge Proofs (ZKPs) and secure Multi-Party Computation (MPC) for secure, attested, remote computation. Intel TDX seems to offer a simpler and more supportive development environment for verifiable computing. In this article, I will take you through a walkthrough of a selection of security concerns with the technology, beginning with a orientation as to how it fits into a widely studied problem in academic computer security research.

Sandboxing

Sandboxing is a decades-long computer security problem that underpins all of computer systems security research today.

1. How do we run code in a confidential and integrous manner i.e. without revealing control and and data flow, and without being tampered with?
2. How do we run code without it compromising the reliability, security, and correctness of the rest of a system (runtime, operating system, VM, bare-metal server)?

First we had the process, an environment in which to run code with a virtual address space (barring side channels, other processes can't see you, and you can't see other processes). This was great, because now application A could no longer directly corrupt application B or bring down the system by itself (the OS can configure resource limits per process now). However, this level of sandbox is insufficient for secure remote

computation, a scenario which allows for a malicious operating system or hardware.

[

Memory Layout of Linux Processes

1297×841 136 KB

](https://collective.flashbots.net/uploads/default/original/2X/a/a54adeef3e22e966e0ba0faabe2df2bd8351c0e8.jpeg)

Source: Mastering Linux Development

Source: OS Notes by Xinyang Yu

Next, we had the enclave, a process with exclusive resources running on a physically isolated portion of a CPU. Enclaves come equipped with encrypted memory for confidentiality and restricted external access for integrity, and attestation, a verifiable proof of integrity. However, this requires partitioning one's application into an untrusted (not needing the sandbox) and trusted components (needing the sandbox), and defining an interface between the two. Why is this bad? Trusted code is run an extremely restricted environment with no system calls (the OS controls their implementation of course). Clearly, we need something better. Why can't we simply throw an entire unmodified application into an "enclave" environment?

[

Intel SGX Core Operating Principles

721×451 12.9 KB

](https://collective.flashbots.net/uploads/default/original/2X/2/24920acc0357aa861f3e3805693c358d969d692d.png)

Source: SecureStreams: A Reactive Middleware Framework for Secure Data Stream Processing

Here is where Intel's Trust Domain eXtensions (TDX for short) comes in: the virtual machine-equivalent of an enclave.

Intel TDX is a trusted execution environment , allowing one to run virtual machines which need confidentiality and integrity (but not availability!) guarantees. Confidential VMs, known as Trust Domains (TDs), are isolated in software and hardware, with transitions managed by the TDX Module, which is the trusted component of the otherwise untrusted hypervisor (just like enclaves assume a potentially malicious OS).

[

What is Intel TDX?

1648×927 96.8 KB

](https://collective.flashbots.net/uploads/default/original/2X/2/277e23c2cd399ce06972391dcd6766fbf8028336.png)

Source: Intel

Source: Intel TDX Security Research

Confidentiality in Intel TDX

Data

TD Memory is encrypted using Intel Multi-Key Total Memory Encryption – each trust domain can use multiple keys to encrypt pages.

Secret memory is managed by secure Extended Page Tables (EPTs) which use hardware support that is included with Intel VT-X virtualization extensions to handle Guest Virtual Address to Host Physical Address translations.

Shared memory is handled directly by the untrusted hypervisor using a separate EPT.

Code

Code in each Trust Domain can only be executed when the processor is in a special CPU mode known as Secure Arbitration Mode (SEAM), that is handled by the TDX Module. TDX is capable of detecting interference to the CPU such as modifying existing instructions or injecting new instructions in private memory. The CPU state is maintained securely during transitions (handled by the TDX Module) and each TD's execution context is stored in its own metadata storage area which is encrypted using a TD-specific key.

Integrity in Intel TDX

TDs can also respond to local and remote attestation requests, which are handled by an SGX quoting enclave.

Memory accesses also involve two possible integrity verification measures including Logical Integrity

(private memory is tracked using the TD Owner bit) and Cryptographic Integrity

(which also computes and validates a MAC of data to detect unauthorized change).

However, rollback attacks (where one state of memory is recorded, and is used to replace a future state) do not seem to mitigated against. Similarly, rowhammer cannot be prevented outright, since it is circuit-level phenomenon beyond the scope TDX.

TDX Attacks

Ahoi Attacks – Heckler

The Ahoi attacks are a pair of vulnerabilities (WeSee and Heckler) related to interrupt-handling in AMD SEV-SNP (AMD's VM-based TEE offering) and Intel TDX. The hypervisor must hook on interrupts in the interrupt controller to be able to forward them to the right TD. This allows each guest OS to run the interrupt service routine for the corresponding interrupt. However, this allows the hypervisor to inject interrupts willy-nilly.

In Heckler, a malicious TDX hypervisor inserts interrupts at carefully chosen times to cause changes to data and control flow in guest applications such as OpenSSH and sudo, unbeknownst to the application or guest operating system. In one example, if the hypervisor sent int 0x80

which is the interrupt used to run a system call in x86

processors, it could change the value in the rax

register. For the sshd

binary, when this was done on a user connection, changing the value of rax

allowed the authentication process to be bypassed, allowing a malicious hypervisor to connect to a TD, breaking the sandbox around the trusted software.

Malicious interrupts were a cornerstone of the SGX attack landscape, and it looks like they are here to stay with VM-based TEEs.

TDX Bomb

While there are no availability guarantees provided by TDX, interruptions to TD execution is something that could disrupt the correct functioning of a distributed protocol. SGX-Bomb is a rowhammer-based fault attack that flips physical bits in DRAM that back SGX memory. This affects the integrity of the enclave causing it to abort and necessitating a restart.

Now, TDX uses SGX for attestation, which could be one vector. Furthermore, overwriting TD private memory causes an abort if the TD Owner bit is set to 0 on the next read. If the only goal is to cause the TD to be destroyed on a frequent enough basis, one could identify frequently called functions or frequently accessed data structures to continuously flip to prevent the TD from being able to complete execution. In fact, the TDX Module could be an independently interesting region of memory to cause flips in for more severe security violations.

Side Channels

[

Side Channels. Side Channels Everywhere.

1688×933 95.3 KB

](https://collective.flashbots.net/uploads/default/original/2X/e/e86deb05a5e6d85cb8a0e51c4da978bb0642c448.jpeg)

Side-channel attacks are a class of vulnerabilities that involve the unintended disclosure of secret data via side-effects or metadata of computation, including the execution time, power consumption, processor frequency, and temperature changes, that might vary as a function of said data. Intel SGX in particular, has been notorious in the security community for dozens of attacks in since its release in 2015. A lot of these have been mitigated in the design of TDX, but since I'm writing this article in 2024, the journey is nowhere near done.

Here is quick example known as simple power analysis. In the below two figures, we're attacking an algorithm called left-to-right square-and-multiply which is a routine used in the RSA algorithm to perform modular exponentiation. Here, the secret is the private exponent E

. For every bit in E

, we always square the intermediate result R_0

, but if the bit is 1 we do an additionally multiply. Thus, iterations with e_i = 1

will consume more power, which can be measured using an oscilloscope.

In contrast, software-based microarchitectural side-channels, such as the ones I will present below, do not require specialized equipment and use measurements such as timing that can be obtained purely in software.

Source: Efficient Randomized Regular Modular Exponentiation using Combined Montgomery and Barrett Multiplications

Source: [Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman

](https://www.mdpi.com/2410-387X/4/2/15)

Non-TDX Specific

The vulnerabilities below broadly apply to a wide range of CPUs. They rely on CPU microarchitecture that any code executed relies on. There are excellent resources on understanding these issues already and I will not be diving deep in any of these for this article.

1. Spectre
2. Meltdown
3. Cache Side-Channel Attacks
4. Microarchitectural Data Sampling
5. Differential Power Analysis
6. Differential Fault Analysis
7. Power/Frequency/Thermal Side-Channel Attacks
8. and more!

TDX Specific

Hypervisor Guest Page Blocking

Intel TDX allows the hypervisor/VMM to block and unblock Guest Physical Address ranges using TDH.MEM.RANGE.BLOCK / UNBLOCK

. This allows the VMM to receive EPT faults when code within the Trust Domain reads from pages that are mapped to the blocked range. This feature presumably exists to help with live VM migrations (reallocation of physical resources to one or more VMs) and for the hypervisor to reclaim memory when it is under memory pressure from other TDs.

Having said that, one can employ this command to construct a timing oracle to observe control flow in a fine-grained manner. Consider an if

statement in a program where each branch falls on a different page. If the branch direction is determined by a secret bit b

, by blocking one of the pages and observing execution, the hypervisor can determine the value of a secret.

This is extremely similar to Controlled Channel Attacks, discovered by researchers from Microsoft Research in 2015. However, several things are different:

1. The VMM operates in a different address space.
2. The VMM cannot influence page tables like an OS can for an enclave process. Private memory in a Trust Domain by the TD Guest OS, with the second level address translation handled by the TDX module's secure EPT.

To mitigate this, one would need to distinguish between benign and malicious guest physical address blocks, which is non-trivial. If we relax the problem constraints by only allowing one TD and ensuring the TD is run on an otherwise quiet system with minimal interrupts, this attack vector may be easier to block.

TD Memory Overwrite by VMM

The hypervisor can overwrite memory within the Trust Domain. However, if TD memory is written to outside of SEAM mode, the TD Owner bit that is present in each cache line is set to 0 (newer versions are likely to use more bits to distinguish Trust Domains, based on personal communication with Intel employees). This bit is set to 1 when in a special CPU mode known as Secure Arbitration Mode (SEAM).

On the next read by the TD, this mismatch in bits is detected by the CPU, which leads to a TD abort (the VM is destroyed). In a similar manner as above, the hypervisor can strategically overwrite memory in one side of a secret-dependent branch to learn the secret predicate. In this case, our oracle is destructive, since the TD's state is destroyed and it must be restarted. A possible mitigation could involve detecting an abnormal number of restarts.

Ciphertext Side-Channel Attacks

TEEs use memory encryption to prevent users in other execution contexts from learning sensitive data, in addition to the physical isolation used to prevent access. However, AMD SEV, a VM-based TEE solution allows hypervisors access to the VM State Save Area on VM Exits. The VM save area is akin to the process control block for a UNIX process, that stores information about the execution context of the VM, including the program counter, register file, and interrupt and exception information.

While this is encrypted in memory, this is unencrypted in the CPU, it can be viewed on state exits. Furthermore, since randomness is expensive, AMD uses deterministic encryption scheme, which was broken in an attack called CipherLeaks. The below image is a simplified explanation of one case of the vulnerability, from a mitigation known as CipherFix. Essentially, when performing a swap of two pages x

and y

, the ciphertext is preserved when they are not swapped, revealing secret bit b

.

[

How deterministic encryption leaks secrets

720×315 53.3 KB

](https://collective.flashbots.net/uploads/default/original/2X/a/a423d0a60c65af303d471761beb096466d32d967.png)

Source: Slides from CipherFix

In TDX, TD memory is stored in plaintext in the CPU. When the hypervisor attempts to read from TD secret memory, it receives all zeroes, since the CPU is not executing in SEAM mode. However, since only a single bit is used to determine whether a page is secret or not (the TD Owner bit), it may be possible for a TD to obtain ciphertext for another TD.

This can be made harder for adversaries with the use of multiple keys by each TD, usage of message authentication codes for integrity, and oblivious RAM (AES XTS uses tweaking to adjust the ciphertext based on the sector number).

Final Thoughts

As the quote from the song [Side Channels are Everywhere

by A Few Mistakes Ago

](https://www.youtube.com/watch?v=qiihq9qmoL8) goes:

Side channels are everywhere; most of the time we're unaware

What I presented was a selection of TDX-specific security vulnerabilities. For side-channels, there are various mitigations that can be applied from a software and system-level to reduce the attack surface, including using constant time implementations (particularly for cryptography) to reduce timing channels, enabling data operand independent timing bits (DOIT) to turn off leaky performance optimizations, masking secrets in memory to reduce the impact of ciphertext side-channel attacks, using attestations to prevent security-critical from running in an environment with simultaneous multithreading enabled, and using oblivious random access memory (ORAM) to access large data structures (particularly those located in shared memory).

Side-channel security is a dynamic, evolving, and active field of research. It is important to be on the lookout for and apply the latest security patches, microcode updates, and other mitigations to reduce the potential for information disclosure, particularly in a world where [information is money

](https://writings.flashbots.net/frontrunning-mev-crisis).

Further Reading

1. Intel TDX Demystified: A Top-Down Approach
2. Intel-Google Security Review of TDX
3. Intel TDX White Papers and Manuals