Why I Stopped Caring About the TCB

Adrien Ghosn
Azure Research
Microsoft
Cambridge, United Kingdom

Marios Kogias
Department of Computing
Imperial College London
London, United Kingdom

1. Introduction

Trusted Execution Environments (TEEs) have become the standard for confidential and attested computing across devices, from embedded systems to large datacenter servers. Initially popularized by Intel SGX [5], introducing secure enclaves to protect parts of an application, TEEs have evolved towards full confidential virtual machines (CVMs). Vendors now provide support for CVMs, with Intel TDX [4], AMD SEV-SNP [7], and ARM CCA [2].

A key concern in confidential computing is the size of the Trusted Computing Base (TCB): the hardware, code, and tools that must be trusted to ensure confidentiality and integrity. The TCB typically includes both the code running inside the TEE and the tools used for compilation and image preparation. Since TEEs by design isolate workloads from a potentially malicious operating system or hypervisor, threats are limited to what runs inside the TEE. The promise of a *small* delimited TCB is that it allows developers to scrutinize code and eliminate vulnerabilities, ensure functional correctness, while also making external audits more feasible, even for those not involved in development.

This paper argues that prioritizing TCB size in TEEs is a wild goose chase, conflating policies with mechanisms. The primary security goal in a TEE deployment is to prevent unauthorized data leakage by enforcing strict information flow guarantees. TCB control is merely a mechanism by which we hope to approximate functional correctness and reduce leakage risks, but has proven to be neither practical nor sufficient, especially when considering CVMs that include full commodity operating systems or complex applications relying on extensive sets of libraries to process data.

Instead of TCB size, we should focus on interfaces between TEEs and untrusted code. These interfaces exist both within a single machine (e.g., a CVM communicating with an untrusted hypervisor) and across machines (e.g., TEEs communicating over a network). By analyzing information flow at these boundaries, we can move beyond the fixation on minimizing TCB size and build more practical, deployable confidential infrastructure. To ensure strong information flow guarantees and proper encapsulation of sensitive data, we propose enforcing strict, attested, and well-defined interfaces between TEEs and untrusted code. Our key intuition is that hardware-enforced modularity and compartmentalization, combined with careful interface design, can uphold security regardless of module size.

2. Position

Intel SGX enclaves promoted a small TCB, though whether this was driven by security principles or hardware constraints remains unclear. By isolating sensitive code within an address space, enclaves protect against a potentially malicious OS or hypervisor. Designed for critical operations like a database's data access unit [6], they ensure only trusted code handles sensitive data. The small size of the trusted code enables a full understanding and precise reasoning about its behavior. This reasoning can either happen through formal verification or code auditing. Moreover, in SGX v1, the Enclave Page Cache (EPC) was limited to 94 MB [5], causing larger enclaves to suffer from costly paging overhead. So, minimizing the TCB was crucial not only for security but also for performance.

Formal verification, even for small applications, has however proven extremely challenging – if not unrealistic – posing a barrier to wider TEE adoption. A review of recent research shows that no work fully verifies all code running inside a TEE, including both infrastructure and application logic. Instead, verification is typically limited to specific properties rather than full functional correctness, leaving room for unexpected behavior and potential data leakage.

Similarly, auditing the TCB is equally challenging. Any TEE performing meaningful operations on sensitive data inevitably depends on external libraries, leading to complex and growing dependencies. These dependencies, in turn, rely on others, rapidly expanding the effective TCB. Even if the core codebase remains small in terms of executable pages, fully assessing its security requires tracking an extensive and ever-evolving web of dependencies, making thorough audits impractical.

In real-world scenarios, the focus on a small TCB has largely faded, as confidential VMs have become the preferred approach. Unlike enclaves, which require applications to be restructured to fit within a restricted execution environment, confidential VMs offer a more developer-friendly alternative by supporting unmodified workloads. They encapsulate an entire operating system, typically Linux, along with multiple processes, significantly expanding the TCB. While this trade-off simplifies development and deployment, it also increases the attack surface, making security harder to reason about and enforce. Simply ignoring the TCB size without introducing new mechanisms to implement the functionality that formal verification or code auditing were expected to serve, opens the door to attacks and data leakage.

We make the observation that any attack towards a TEE that tries to compromise integrity or confidentiality stems from an interaction between a TEE and untrusted code, since the TEE architecture by design protects the rest of the TEE. There are several such interfaces that have been or could potentially be exploited, including but not limited to: the hypercall [8] and syscall [3] interfaces, paravirtual devices for IO, interrupt and signal handlers, or any adhoc application-level communication scheme over non-confidential shared memory. Unfortunately, these interfaces have been shown to be prown to attacks irrespective of the TCB size.

As an example, we take a look at the recent Ahoi attacks to consolidate our point that low TCB does not necessarily prevent attacks towards TEEs. The Ahoi Attacks [1] leverage malicious notifications in the form of interrupts, exceptions, and signals to compromise TEEs across different architectures. These attacks were performed against CVMs running a full-fledged operating system on SEV-SNP but also against library operating systems and runtimes optimized for low-TCB in Intel SGX. Rather than getting rid of vulnerable interfaces, the need to support existing software, such as the NGINX servers, requires even low-TCB solution to re-implement and aggressively simplify functionality that exists in operating systems, such as interrupt handlers, without always considering how these interfaces could be attacked.

We suggest that code size is irrelevant as long as it avoids interacting with vulnerable interfaces that could compromise confidentiality or integrity. Instead of fully understanding what their code does, developers of confidential applications must enforce strict guarantees on what it cannot do – shifting the focus from minimizing the TCB to controlling interactions and thus information flow. For example, an ML inference engine may use vector computation libraries, but as long as these do not (or cannot) interact with the rest of the system, they pose no security risk.

The key missing feature is the ability to enforce information flow by construction rather than relying on functional correctness. TEEs lack strict encapsulation, leaving communication with untrusted code uncontrolled and placing the burden on developers to manage it through careful reasoning about *all* of the code they deploy. This paper argues for decoupling the TCB from all code running inside a TEE. Rather than minimizing the TCB and expecting developers to manage potentially exploitable interfaces, we advocate for eliminating these interfaces whenever possible or substitute them with attested trusted modules.

3. Proposal

We propose structuring codebases inside TEEs as communicating modules, each adhering to the principle of least privilege. Module size does not impact security as long as boundaries are enforced and information flow is explicit and attested. We explore different ways to implement this design, some supported by existing TEE solutions and others paving the way for new TEE architectures. We identify the essential common elements for these approaches, providing a list of minimal requirements.

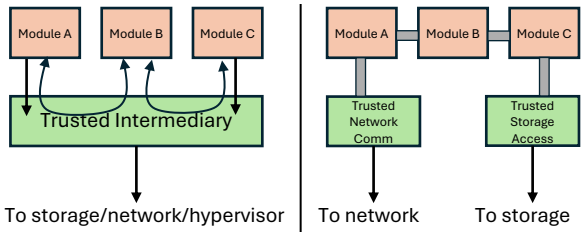

Figure 1. Alternative designs agnostic to TCB size

Developers should organize code into modules and explicitly define what each module cannot do. For example, modules should not have IO device access (e.g., network, storage), nor should they handle interrupts, exceptions, or issue arbitrary system or hypercalls. A module's capabilities should be independent of the mechanisms that control them. Critical functions should be offloaded to trusted components, allowing deprivileged modules to have arbitrarily large sizes without compromising security.

We identify two main designs that allow for such architectures, as depicted in Figure 1. The one follows a more traditional hierarchical pattern and is supported by existing TEEs, such as AMD's SEV-SNP. It depends on a trusted privileged component inside the TCB that mediates every communication between modules with the outside world and among themselves to enforce this principle of least privilege. The correctness of this approach still depends on the functional correctness of a single small privileged piece of code that needs to account for all possibly desired interactions with the untrusted world.

The alternative design follows a decentralized approach and does not depend on a single trusted component. Instead, it splits the functionality that interfaces with untrusted code to different components that are trusted, e.g. the network and storage modules. Then, it enforces and attests certain allowed communication channels (grey pipes) between the untrusted modules with the trusted components based on what the untrusted modules should or should not do. For example, a trusted component might implement encrypted network communication. Untrusted modules should only communicate with that trusted component if they want to access the network.

An extreme instantiation of the second design can take the form of an entire server using a trusted network interface. If the only way a server can communicate with the outside world, and thus leak sensitive information, is through the network, by leveraging a trusted NIC that can implement firewall and encryption mechanisms can play the role of becoming a trusted interface.

For both designs there are two main required hardware features. First, TEEs need to support some form of hardware-enforced isolation. This can come in the form of page tables, segmentation, or capabilities, as long as distinct modules can be isolated. Second, there should be at least two distinct levels of hierarchy, such that the privilege level can either mediate the communication between modules or strictly enforce certain information flows among allowed modules. Finally, attestation should make module boundaries and their communication channels explicit, allowing to reason about authorized information flow in the entire deployment.

References

[1] The ahoi attacks. https://ahoi-attacks.github.io/.

[2] Arm confidential compute architecture. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture.

[3] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, André Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Peter R. Pietzuch, and Christof Fetzer. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of the 12th Symposium on Operating System Design and Implementation (OSDI)*, pages 689–703, 2016.

[4] Intel. Architecture specification: Intel trust domain extensions (intel tdx) module. https://software.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-module-1eas.pdf, 2023.

[5] Intel. Intel software guard extensions (intel sgx). https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html, 2023.

[6] Christian Priebe, Kapil Vaswani, and Manuel Costa. EnclaveDB: A Secure Database Using SGX. In *IEEE Symposium on Security and Privacy*, pages 264–278, 2018.

[7] AMD Sev-Snp. Strengthening vm isolation with integrity protection and more. *White Paper, January*, 53:1450–1465, 2020.

[8] Ziqiao Zhou, Anjali, Weiteng Chen, Sishuai Gong, Chris Hawblitzel, and Weidong Cui. VeriSMo: A Verified Security Module for Confidential VMs. In *Proceedings of the 18th Symposium on Operating System Design and Implementation (OSDI)*, pages 599–614, 2024.