Improving Security and Practicality of Modern Trusted Execution Environments

Background and research vision. Cloud computing underpins critical services in healthcare, finance, and AI. Yet, it faces a constant threat of cyberattacks and data breaches. In 2024, these cybercrimes exposed over 280 million records in the US's health sector alone. Data breaches are not only financially devastating for the organizations employing cloud services, costing them up to $4.4 billion in remediation on average in 2025, they also erode customers' trust in cloud providers.

To regain trust, cloud providers are now strengthening the security of their own infrastructure and isolating their customers' systems from it to minimize the risk of compromises in their platform from permeating to customers. In the latter case, particularly, providers are increasingly adopting specialized hardware from compute vendors (e.g., Intel, AMD). These are new processors with secure primitives built into the silicon that enable the creation of a secure "bubble" on the platform, called a Trusted Execution Environment (TEE), which protects sensitive data and programs, even if the rest of the system, including the cloud administrator, is compromised.

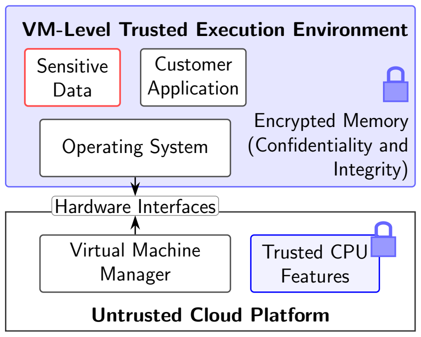

Figure 1: Modern TEE design

After the initial exploration of TEE designs, the community has settled on the idea of isolating an application along with its complete execution environment, including libraries and supporting software, within a single TEE. This design has helped migrate many applications into TEEs, resulting in reduced performance and engineering costs. Especially, companies have been deploying cutting-edge workloads such as large language model inference into TEEs with only trivial adaptation. This ease of adoption requires a significant amount of code being placed inside a TEE. This code (called the trusted computing base or the TCB of an application), on which the security guarantees for the customer's application and data depend, includes the operating system (OS). Placing feature-rich OSes in the TEE eases adoption, but it also increases the risks of vulnerabilities.

My vision is to tame these risks in the TEE OSes while further enabling them to provide stronger security and better functionality to the TEE applications. Towards this end, my research will focus on two main directions: *compartmentalizing the TEE OS* to reduce the exploitability of vulnerabilities within the OS, and *designing OS-assisted mitigations* for novel threats to TEE applications, such as side-channel attacks from a compromised host platform.

Taming TEEs' large TCB with compartmentalization. The large TCB hinders efforts to prevent and predict bugs that allow attackers to compromise the security of the TEE; widely-used OSes like Linux often consist of several million lines of code and known to have thousands of vulnerabilities. Worse, a feature-rich TCB exposing many endpoints (storages and external communication) through which information can be leaked, so any part of the TEE that is comrpomise can easily be used to leak information.

Prior research has tackled this challenge in three broad ways: (i) comprehensively testing the TEE code to detect bugs and vulnerabilities, (ii) reducing the size of TCB by removing extraneous functionality, and (iii) compartmentalizing the large codebase into smaller, isolated compartments to restrict the impact of attacks in individual compartments from spreading to the entire TEE. Unfortunately, (i) and (ii) are prone to missing vulnerabilities in complex TCBs and do not generalize to multiple OSes and applications.

Instead, my focus is on improving compartmentalization techniques such that it is incredibly challenging to exfiltrate sensitive information, even when an attacker is able to compromise the TEE. Building on my experience in software compartmentalization, I will explore in-TEE compartmentalization mechanisms to establish in-OS compartments based on their functionalities, as shown in Figure 2. While it is possible to manually rewrite the OSes with compartmentalization in mind, such an approach would be impractical in terms of engineering costs. I plan to explore automated approaches that leverage program analysis to alleviate developers burden on compartmentalizing the in-TEE OS.

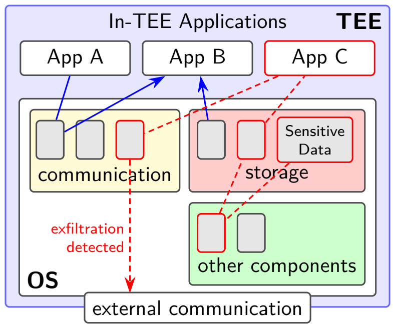

Figure 2: Compartmentalization of in-TEE OS and cross-compartment information flow tracking.

I plan to track the flow of sensitive information such that the in-TEE OS is able to detect whenever secret information is about to be exported from the TEE. To this end, the work on whole-system provenance analysis, led by Thomas Pasquier and Margo Seltzer at UBC, is particularly promising. Whole-system provenance analysis techniques record all resource accesses made by applications within the system. With this data, these techniques can pinpoint the source (or provenance) of any resource about to be accessed. I plan to incorporate these techniques into my work to provide the OS with a mechanism for determining whether a particular operation could potentially leak sensitive data, allowing the it to strictly negate information leaks in a principled manner.

Practical side-channel mitigation with OS assistance Side channels are a critical yet persistent class of vulnerabilities at the software-hardware boundary of TEEs, allowing attackers (e.g., in a compromised cloud hypervisor) to leak sensitive information from sensitive computations even without direct access. For instance, even though a compromised cloud hypervisor cannot access a TEE's memory, it can control and observe the order in which a TEE application accesses memory locations during its computations, which is sufficient to reveal secrets in many applications. For example, such attacks have been shown to reveal the image being processed by the TEE or its secret encryption key.

One way to mitigate such attacks would be for the cloud provider to secure the memory management mechanisms in their hypervisors, for instance by eliminating dynamic and flexible memory management across multiple co-located tenants. However, such an approach involves substantial engineering costs for the provider and also raises concerns of resource utilization efficiency on the cloud platforms. The second, less intrusive approach, is to modify the resource usage patterns generated from the sensitive application, such that they can no longer be used to infer secrets. Unfortunately, many solutions that adopt this require a significant software rewrite, which is often not scalable in terms of engineering and performance costs.

My proposal aims to explore OS-assisted solutions toward making the second approach for side-channel mitigation more practical and performant. The in-TEE OS can control the placement of data within a TEE's memory, and also all its file accesses. This capability enables the OS to transform resource usage patterns, eliminating side channels, all without requiring any modifications to the resource-consuming applications. During my PhD, I developed a minimal, proof-of-concept OS that provides such a solution using a OS with custom memory management, which can only secure the usage patterns for resources used by simple cloud applications. I plan to integrate the into a production-scale, feature-rich OS like Linux to support a larger class of applications, which will require addressing several fundamental security, performance, and engineering costs due to scale.

First, the sheer complexity of the OS makes attempts to protect all resource accesses across both the OS and application code prohibitively expensive in terms of performance cost and demand significant engineering effort. A more sensible approach is to apply protection only to resource accesses that can potentially leak secrets. Toward this goal, I expect the cross-compartment information flow tracking system proposed previously would be helpful here. With the information, the OS would only protect the sensitive resource accesses, while ommiting the protection of other accesses.

Next, even when these decisions are made, applying protections to resource accesses still implies significant changes to how the OS manages its resources. Such changes can easily break compatibility with existing workloads, undermining the deployability advantage of modern TEEs. A way forward is to develop mechanisms that extend OS resource management in a non-intrusive manner. To achieve this, I plan to explore synergies with the work on application-aware memory management by Alexandra Fedorova. These systems provide extension points to the OS, upon which custom resource management policies for specific applications can be implemented. Thus, the OS side-channel protection would only affect the management of selected applications and resources, while leaving the rest to use standard OS management methods.

I propose shifting the focus from the size of the TCB to the means by which trusted and untrusted worlds communicate – their interfaces. These interfaces, often inherited from legacy systems, were rarely designed with strong adversarial models in mind and are easy targets for attacks. Numerous studies have demonstrated that the interfaces expose critical vulnerabilities that leak sensitive data due to their lack of security considerations and legacy design choices.

First, building on my experience in software compartmentalization, I plan to explore methodologies that compartmentalize code interacting with the interfaces, thereby containing the impact of the untrusted platform on security where it matters most. A challenge would be determining the boundary of the compartment to maximize performance while minimizing possible attack surfaces. To this end, I plan to collaborate with researchers from Systopia Lab, who are currently developing theoretical models to evaluate compartmentalization strategies, thereby informing my proposed compartmentalized interface designs.

By focusing security decisions at the interfaces where they matter, TEE designs can manage complexity in a principled manner without sacrificing functionality. This enables the introduction of additional features, such as performance optimizations. More importantly, security mechanisms that are based on complex OS features can be implemented, as presented next.

Exploration of OS as a security component. My exploration in this direction will focus on practical challenges that plague modern TEE deployments: side-channel mitigation and private data processing in the cloud.

Second, as a component of TEEs, the role of operating systems must now be rethought. Unlike normal applications, the operating system's primary task is to manage system resources, such as memory and files, by utilizing its access to low-level hardware mechanisms that provide fine control over software running on the system. Importantly, the operating system's high degree of control not only enables transparent resource management but also offers powerful tools for building in-TEE security mechanisms that are practical and performant. The work conducted during my doctoral studies demonstrated the promise of this approach, leveraging OS capabilities to implement practical TEE self-defense mechanisms, particularly against side-channel attacks. Building on that foundation, the challenge now is to explore a broader class of OS-supported techniques that address the issues in practical TEE deployments.

One way to mitigate such attacks would be for the cloud provider to secure the memory management mechanisms in their hypervisors, for instance by eliminating dynamic and flexible memory management across multiple co-located tenants. However, such an approach involves substantial engineering costs for the provider and also raises concerns of resource utilization efficiency on the cloud platforms. The second, less intrusive approach, is to modify the resource usage patterns generated from the sensitive application, such that they can no longer be used to infer secrets. Unfortunately, many solutions that adopt this require a significant rewrite of the software, which is often not scalable in terms of engineering and performance costs.

My proposal aims to explore OS-assisted solutions toward making the second approach for side-channel mitigation more practical and performant. The in-TEE OS can control the placement of data within a TEE's memory, and also all its file accesses. This capability enables the OS to transform resource usage patterns, eliminating side channels, all without requiring any modifications to the resource-consuming applications. During my PhD, I developed a minimal, proof-of-concept OS that provides such a solution using a OS with custom memory management, which can only secure the usage patterns for resources used by simple cloud applications. I plan to integrate the into a production-scale, feature-rich OS like Linux to support a larger class of applications, which will require addressing several fundamental security, performance, and engineering costs due to scale.

First, the sheer complexity of the OS makes it impractical protect all resource accesses across both the OS and application code; such an approach be prohibitively expensive in terms of performance cost, and demand significant engineering effort. In response I plan to explore two directions toward addrsesing these limitations. First, I will explore theoretical models

A more sensible approach is to apply protection only to resource accesses that can potentially leak secrets. This raises the next challenge: deciding which accesses should be protected. This is nontrivial, especially in complex workloads where multiple resources are accessed, often by interacting applications. To this end, the work on whole-system provenance analysis, led by Thomas Pasquier and Margo Seltzer at UBC, is particularly promising. Whole-system provenance analysis techniques record all resource accesses made by applications within the system. With this data, these techniques can pinpoint the source (or provenance) of any resource about to be accessed. I plan to incorporate these techniques into my work to provide the OS with a principled mechanism to identify exactly at which point a sensitive resource access is about to be made. This information renders the sensitive resource protection efficient; the OS would only modify the sensitive resource accesses about to be made to eliminate the side channels.

Next, even when these decisions are made, applying protections to resource accesses still implies significant changes to how the OS manages its resources. Such changes can easily break compatibility with existing workloads, undermining the deployability advantage of modern TEEs. A way forward is to develop mechanisms that extend OS resource management in a non-intrusive manner. To achieve this, I plan to explore synergies with the work on application-aware memory management by Alexandra Fedorova. These systems provide extension points to the OS, upon which custom resource management policies for specific applications can be implemented. Thus, the OS side-channel protection would only affect the management of selected applications and resources, while leaving the rest to use standard OS management methods.

Practical private data processing.

To give a more concrete example of the second challenge, companies like 23andMe provide personal health data analytics as a cloud service. The service recieves users' private data, on which it runs the health analytics program, and returns the results to the user. These deployment scenarios require addressing conflicting security interests of the users and analytics providers. Users require that the analytics services keep their data confidential and do not share with third parties. One way they could get this assurance is by inspecting and verifying the code of the analytics providers, but the analytics providers wish to keep their proprietary analytics confidential from their customers. Thus, existing deployments employ a trusted third-party, e.g., a cloud provider with TEE-enabled hardware, as a mediator. The mediator deploys a TEE, to which users entrust their data and the analytics providers entrust their analytics program secrets (Figure 3). It then runs the analytics program on behalf of the analytics providers on users' data and prevents exfiltration of the program and the data to untrusted communication or storage endpoints. Unfortunately, such endpoints are abundant in most OSes, making it difficult to prevent information leak.

Stakeholder	Sensitive Asset	Security Interest
Service users	Personal data	Ensure data confidentiality
Service providers	Proprietary data processing	Keep trade secrets confidential

Table 1: Representative private data processing stakeholders, and their assets and interests.

A common scenario for CCC deployments is *private data processing* involving multiple stakeholders, as shown in Table 1. For instance, companies like 23andMe provide personal health data analytics as a cloud service. Its users upload private data to the service's server, which runs a health analytics program on the data and returns the results to the user. These deployment scenarios require addressing conflicting security interests of the users and analytics providers. Users require that the analytics services keep their data confidential and do not share with third parties. One way they could get this assurance is by inspecting and verifying the code of the analytics providers. However, the analytics providers wish to keep their proprietary analytics confidential from their customers.

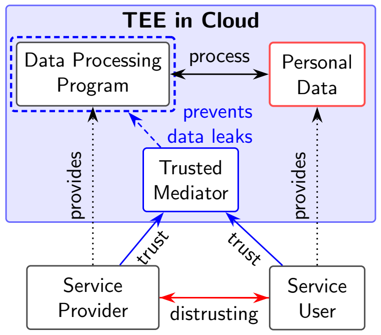

Figure 3: How CCC resolves stakeholders' conflicts.

My proposal looks to improve the usability of existing mediator designs by exploring how the trusted mediator can be incorporated into the in-TEE OS. The rationale is that, since the OS is in charge of resource management for the TEE, it is able to track how the data processing application interacts with sensitive data. With this information, the program's operations are restricted in a "smart" manner – stopping operations only when they may lead to information leaks, but allowing them otherwise. Compared to a static allowlist approach, this dynamic approach would allow the mediator to be compatible with a wider range of compliant programs.

A challenge with this direction is how to reliably detect the propagation of sensitive data across the complex codebase of the OS. To this end, the work on whole-system provenance analysis, led by Thomas Pasquier and Margo Seltzer at UBC, is particularly promising. Whole-system provenance analysis techniques record all resource accesses made by applications within the system. With this data, these techniques can pinpoint the source (or provenance) of any resource about to be accessed. I plan to incorporate these techniques into my work to provide the mediator with a mechanism for determining whether a particular operation could potentially leak sensitive data, allowing the mediator to strictly negate information leaks in a principled manner.

1 Appendix: Guide (REMOVE IN FINAL)

The proposal should be written in clear, non-technical language that allows a non-specialist to comprehend the overall content and importance of the work. Members of the Killam Postdoctoral Fellowships and Prizes Committee are from abroad and range in disciplines, and may not have expertise in your area of study.

Although the fellowship may be used to extend or expand upon doctoral work, it must be made clear that you are not intending to use the award to wrap up a thesis. While it is expected that a postdoctoral fellow will be taking the next step beyond the PhD thesis, you must differentiate clearly between the postdoctoral project and the thesis research. Feel free to include hyperlinks to provide links to additional information.

As applicants must make UBC their base, it is important – particularly for applicants whose primary research materials are elsewhere – to indicate what travel is involved, to where, and for how long. You should describe how you will deal with the remoteness of the primary materials.

The adjudication committee is very interested in “fit” with the selected UBC department or unit and the university’s research programs. You must provide information on how your research relates to that of specific campus programs and advisors. If a colloquium is envisioned, a possible title should be proposed. If you visit the classes, please suggest which ones. Since interdisciplinarity is often a valuable dimension (the Killam Trusts declares that a candidate shall not be “a one-sided person”), specific details on proposed interdepartmental connections are welcome.