Toward Process-Level TEEs with OS Compatibility and Minimal TCB

Guojun Wu* Keisuke Iida* Satoru Takekoshi* Takahiro Shinagawa
The University of Tokyo
{wu,iida,takekoshi}@os.is.s.u-tokyo.ac.jp *shina@is.s.u-tokyo.ac.jp*

Abstract

Background Traditionally, cloud users had to trust the infrastructure of cloud service providers to use their services. However, demand is growing to safeguard workload data even when provider systems or personnel cannot be fully trusted. Trusted execution environments (TEEs) have gained significant attention as a practical foundation for secure cloud computing. By using CPUs as a root of trust, TEEs enforce confidentiality and integrity for sensitive applications against privileged software, enabling secure execution even under a compromised OS or hypervisor. This protection relies on hardware-based memory encryption, secure key management, and remote attestation, which allows remote parties to verify that their code and data are protected. These features have made TEEs appealing for secure cloud computing, including privacy-preserving analytics, machine learning inference, and confidential containers.

Challenges However, existing TEEs face significant challenges. While enclave-based TEEs such as Intel SGX offer strong isolation with a small trusted computing base (TCB), they create major compatibility issues for existing systems. Because enclaves impose strict constraints on the code that can run inside them, such as disallowing system calls and limiting library support, applications cannot execute on Intel SGX without modification, which complicates their development and verification. To address these limitations, recent CPUs support confidential virtual machines (CVMs), which allow unmodified OSes and applications to run inside a TEE. Although this improves compatibility, it brings the entire guest OS into the TEE, enlarging the TCB and increasing security risks. Indeed, several attacks have exploited vulnerabilities in guest OSes inside CVMs [7, 8].

Related Work Prior studies have explored two main directions to address the trade-off between compatibility and TCB size. One line of work focuses on running unmodified applications inside SGX enclaves. For example, Haven [2], SCONE [1], Panoply [10], and Occlum [9] extend compatibility by supporting standard OS abstractions, but they still require dedicated toolchains, code annotations, or incur a large TCB. Another line of work aims to reduce the TCB of guest OSes in CVMs while retaining compatibility. Gramine-TDX [5] employs a minimal library OS to shrink the attack surface, but still includes tens of thousands of lines of privileged code. Striking a balance between compatibility and TCB minimization thus remains a central challenge for practical trusted execution.

Proposal We propose a new abstraction of TEEs called a *confidential process*, a process-level TEE that runs a single user-level process inside a CVM. By placing only the process in the TEE and reusing the system call interface to interact with an untrusted host OS, our approach preserves compatibility without including the OS kernel in the TCB. System calls are intercepted by a *host proxy process* outside the TEE, which enforces sandboxing before forwarding them to the host OS. To defend against Iago attacks [3], a small in-TEE component called the *syscall mediator* verifies results and copies returned data into the process's address space. This enables safe delegation of services such as file and network access while keeping the kernel outside the TCB. Unlike enclaves, confidential processes run unmodified binaries without enclave runtimes or developer refactoring. Compared to CVMs, they eliminate full guest OS stacks and significantly reduce trusted code. Our design resembles Overshadow [4], but builds on a modern CVM-based architecture with a much smaller TCB. Overall, confidential processes combine the compatibility of CVMs with the minimal TCB of enclaves, offering a practical foundation for process-level trusted execution.

Implementation We are implementing confidential processes on AMD SEV-SNP using KVM with a custom virtual machine monitor (VMM). At startup, the VMM prepares a process image containing the guest program and a lightweight in-TEE runtime that includes a syscall mediator running in privileged mode. The VMM then creates a CVM instance via the KVM interface, loads the process image, and initial-

*These authors are students.

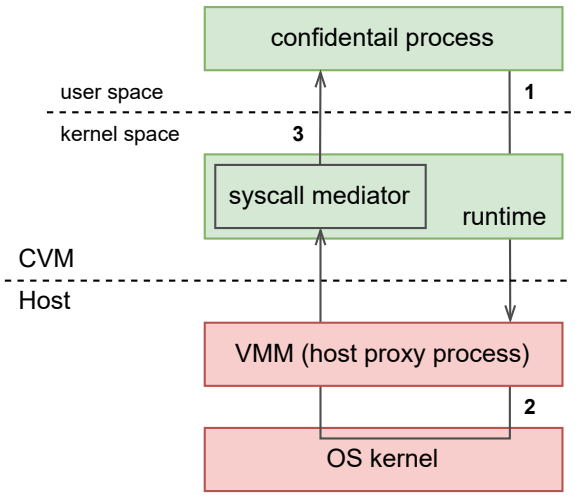

Figure 1: System call handling flow for confidential processes: (1) the runtime writes the system call arguments to shared memory pages, (2) the VMM forwards the system call to the host kernel and return the result, and (3) the syscall mediator validates the response and copies data into process memory.

izes the virtual CPU. After this setup, the runtime sets up the environment and transfers control to the program entry point. When the process issues system calls, the runtime intercepts the invocations and copies arguments from private pages (encrypted memory regions) to shared pages (unencrypted memory regions accessible to the host), then triggers a VM exit. The VMM, acting as a host proxy process, forwards the request to the host kernel, return the result, and restarts the CVM. The in-TEE syscall mediator validates the response and copies data into process private pages.

Figure 1 illustrates this system call handling flow. Our approach resembles Noah [6], but differs in that the runtime uses the CVM's shared pages to communicate with the host proxy process. We are currently extending the implementation to support a broader set of system calls.

Evaluation We conducted a preliminary evaluation by measuring file I/O throughput using the read and write system calls. Our evaluation ran on an AMD EPYC 9754 machine with 128 GB of memory. With a buffer size of 2 KB, the read and write throughputs of confidential processes were 173 MB/s and 143 MB/s, respectively. These values are 12-fold and 8-fold lower than those of normal processes. Although throughput increased with buffer size, it remained considerably lower than that of normal processes. We further measured the time spent inside the CVM during each read/write system call. As buffer size increased, a growing fraction of time was spent inside the CVM. With a buffer size of 32 KB, about 80% of the total time for a single write system call was spent inside the CVM. The remaining 20% covered CVM exit/enter and host-side system calls. These results indicate that the main bottleneck is data copying between private and shared pages, which incurs overhead from encryption and decryption. These preliminary findings highlight the need for further optimization of data movement and cryptographic operations to make confidential processes practical.

Discussion Data copying between private and shared pages is a major challenge for confidential processes, as shown in our evaluation. This problem is inherent to CVMs and therefore does not constitute a disadvantage of confidential processes compared to conventional CVM architectures. However, reducing this overhead is essential for practical adoption. We plan to make the number of shared pages in a CVM adjustable at runtime. Currently, this number is fixed when the VMM creates the CVM instance and cannot be changed once the CVM is launched. This limitation degrades performance when an I/O request exceeds the capacity of the shared pages, since the in-TEE runtime must split the request into multiple smaller I/Os. Allowing the number of shared pages to increase dynamically would avoid this situation and improve performance.

We also plan to implement content-based encryption avoidance. In the current implementation, data read via system calls is always copied into process memory, incurring encryption overhead. Since the data provided by the host is not confidential, encryption is unnecessary if the application only reads the data. We can apply copy-on-write to avoid redundant encryption: when the application attempts to write to a shared page, we copy that page to a private page and update the page table entry to point to the private page.

Memory management is another target of our optimization. Unlike other system calls, memory-related system calls are handled within the CVM. CVMs require guest page tables to reside in private pages, which means only code inside the CVM can modify them. If we delegated memory management to the VMM, we would need another interface to update guest page tables, enlarging the attack surface. It is therefore appropriate to handle memory management inside the CVM. However, placing memory management code in the in-TEE runtime enlarges the TCB. To optimize this, we plan to leverage Virtual Machine Privilege Levels (VMPLs), a new feature of SEV-SNP. We can decouple memory management into privileged and non-privileged components and run the latter at a less privileged VMPL, reducing the risk of memory access patterns leaking sensitive data to shared pages.

Summary We presented confidential processes, a process-level TEE that combines CVM compatibility with a minimal TCB. Our design places only the application inside the TEE, while an in-TEE syscall mediator securely delegates system calls to an untrusted host. Implemented on AMD SEV-SNP, our preliminary evaluation shows data copying as the main bottleneck. We are exploring optimizations such as dynamic shared page management and encryption avoidance to make confidential processes practical for secure cloud computing.

References

[1] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. M. Eyers, R. Kapitza, P. R. Pietzuch, and C. Fetzer. Scone: Secure linux containers with intel sgx. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pages 689–703, 2016.

[2] A. Baumann, M. Peinado, and G. C. Hunt. Shielding applications from an untrusted cloud with haven. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*, pages 267–283, 2014.

[3] S. Checkoway and H. Shacham. Iago attacks: why the system call api is a bad untrusted rpc interface. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, page 253–264, 2013.

[4] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, page 2–13, 2008.

[5] D. Kuvaiskii, D. Stavrakakis, K. Qin, C. Xing, P. Bhatotia, and M. Vij. Gramine-tdx: A lightweight os kernel for confidential vms. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 4598–4612, 2024.

[6] T. Saeki, Y. Nishiwaki, T. Shinagawa, and S. Honiden. A robust and flexible operating system compatibility architecture. In *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '20, page 129–142, 2020.

[7] B. Schlüter, S. Sridhara, A. Bertschi, and S. Shinde. WeSee: Using Malicious #VC Interrupts to Break AMD SEV-SNP. In *Proceedings of the IEEE Symposium on Security and Privacy (IEEE S&P '24)*, pages 4220–4238, May 2024.

[8] B. Schlüter, S. Sridhara, M. Kuhne, A. Bertschi, and S. Shinde. HECKLER: Breaking Confidential VMs with Malicious Interrupts. In *Proceedings of the 33rd USENIX Security Symposium (USENIX Security '24)*, pages 3459–3476, Aug. 2024.

[9] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 955–970, 2020.

[10] S. Shinde, D. Le Tien, S. Tople, and P. Saxena. Panoply: Low-TCB Linux Applications With SGX Enclaves. In *Proceedings of the Network and Distributed System Security Symposium (NDSS) 2017*, 2017.