Umbra: An efficient framework for trusted execution on modern TrustZone-enabled microcontrollers

Stefano Mercogliano
Dept. of Electrical Engineering and Information Technologies
University of Naples Federico II
Naples, Italy
stefano.mercogliano@unina.it

Alessandro Cilardo
Dept. of Electrical Engineering and Information Technologies
University of Naples Federico II
Naples, Italy
acilardo@unina.it

***Abstract*—The rise of microcontrollers in critical systems demands robust security measures beyond traditional methods like Memory Protection Units. ARM's TrustZone-M offers enhanced protection for secure applications, yet its potential for deploying Trusted Execution Environments often remains untapped, leaving room for innovation in managing security on resource-constrained devices. This paper presents Umbra, a Rust-based framework that isolates mutually distrustful applications and integrates with untrusted embedded OSes. Leveraging modern security hardware, Umbra features an efficient secure caching mechanism that encrypts all code exposed to attackers, decrypting and validating only necessary blocks during execution, achieving practical Trusted Execution Environments on modern microcontrollers.**

***Index Terms*—ARM TrustZone-M, Trusted Execution Environment, Rust for Secure Development, Lightweight Security Mechanisms**

I. Introduction

The widespread adoption of microcontrollers in fields like robotics, automotive, and smart devices has heightened the need for robust embedded security. Modern microcontrollers boast advanced features, yet traditional protections like MPUs are insufficient for today's interconnected environments. ARM's TrustZone-M offers a dedicated layer of security for creating Trusted Execution Environments (TEEs), which are crucial for isolating distrustful applications and safeguarding against vulnerabilities. Current TrustZone-M solutions focus on secure communication [3], system checkpointing [1], or adding isolation layers [4]. While essential, isolation alone is insufficient to maintain integrity and security, as TEEs remain vulnerable to implementation flaws or hardware attacks. The former can be mitigated with safe languages like Rust, which prevent pointer-related bugs via strict ownership rules. Addressing hardware attacks requires encryption and authentication, but such methods often slow software-based TEEs on constrained devices. Advanced microcontrollers with built-in security accelerators offer a practical path forward, enabling robust, encrypted execution environments with minimal performance impact. **Our goal** is to develop a TrustZone-based solution that enables practical TEEs on constrained microcontrollers. We present **Umbra**, a Rust-based framework that sandboxes mutually distrustful applications and integrates seamlessly with untrusted OSes. To achieve practical trusted execution, Umbra implements an efficient secure caching mechanism that leverages modern microcontroller security features. It ensures that attackers are always exposed to encrypted code, selectively decrypting and validating only the required code blocks for execution.

II. Design

Umbra operates in the TrustZone secure world as a minimal interface to the eOS, or host kernel, reusing the host's process abstraction, syscall, and scheduling. It focuses on essential security functions to keep the TCB small. Umbra uses the *enclave* process model, extending the host's process framework. Enclave binaries are encrypted in flash memory for confidentiality during linking and to protect against unauthorized access. When deployed, the flash section containing the enclave is redefined as secure but remains encrypted to prevent exposure in persistent memory. Enclave code is split into *Enclave Flash Blocks (EFBs)*, which are decrypted in a dedicated SRAM region, the *Enclave Flash Block Cache (EFBC)*, for execution. The Secure MPU and SAU protect enclave memory from external threats. When an enclave binary fits entirely within the in-RAM *EFBC*, validation and decryption are one-time costs. However, secure applications can be larger, and enclaves must share RAM with other processes and host kernel structures. If an enclave tries to execute code outside its *EFBC*, a MemFault occurs, triggering a costly re-validation process from flash memory to load a new *EFBC*. Since modern TrustZone-enabled microcontrollers typically include multi-channel DMA alongside hash and encryption accelerators, Umbra takes advantage of this by decoupling enclave execution from code validation, entrusting execution to the processor and SRAM, while validation involves the flash and the accelerators. This separation creates two nearly independent computational paths, optimizing overall performance. Umbra implements an optimized caching scheme tailored to secure microcontrollers with dual-memory models, typically featuring separate SRAM regions. It reserves a secure space, the *Enclave Swap Space (ESS)*, to cache validated and decrypted *EFBCs* based on code locality, ensuring fast execution in SRAM. Management of the ESS relies on the *Enclave Swapping Table (EST)*, a metadata table that tracks block usage and access patterns. Initially, space is evenly distributed among enclaves, with a preference for loading contiguous *EFBCs* (e.g., neighboring blocks) to improve locality. When space constraints arise, Umbra selects and replaces the least-used blocks based on access counters, ensuring efficient use of limited secure memory. *ESS* operates

978-3-9826741-0-0/DATE25/© 2025 EDAA

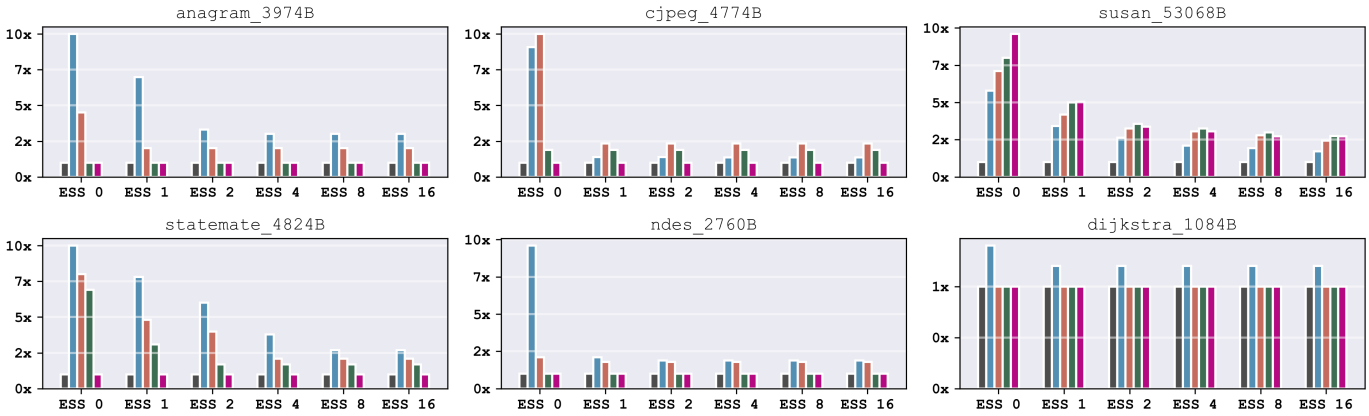

Fig. 1. Enclave execution time overhead for applications of the TACLeBench [2] benchmark with the size of the ESS. Bar colors represents increasing EFBC values in kBs (1,2,4,8), while the first dark bar is the baseline (1×).

like a cache, leveraging SRAM's faster access speeds compared to flash memory. By decoupling block validation from enclave execution, Umbra can speculatively fetch, validate, and load *EFBCs* into the *ESS* without disrupting enclave operations. Block replacement occurs transparently during execution, leveraging secure memory regions protected by the Secure MPU and SAU. However, when an enclave encounters a MemFault for a missing *EFBC*, the system performs an explicit *ESS* update, temporarily halting execution to fetch and validate the required block. This mechanism balances secure execution with practical performance in resource-constrained environments.

III. Evaluation

This section presents a performance analysis of Umbra's secure caching mechanism. Experiments were performed on an ST32L562 microcontroller, equipped with TrustZone-M, as well as AES and SHA2 hardware accelerators. The evaluations utilized test applications from the TACLeBench [2] benchmark suite, specifically tailored for embedded systems. Figure 1 shows the runtime overhead for enclaves running different applications. Without the *ESS*, all applications experience excessive overhead, up to $10\times$, unless the *EFBC* in use is large enough to accommodate the entire application. This overhead arises because a full validation is required each time a new block of code is accessed and it cannot be overlapped with the enclave execution. Runtime performance consistently improves with a larger *ESS* until the entire application binary fits within it, albeit at the cost of increased memory overhead. The size of the *EFBC* also significantly affects performance. Depending on the application's code locality, a larger *EFBC* can be either more or less effective. Applications with good code locality, such as `ndes` and `cjpeg`, show the best results overall both in terms of memory usage and execution time. The former is minimal in space, making it feasible to fully populate the *ESS*, resulting in a reasonable overhead of 68%. Similarly, the latter, can run with just 46% overhead using only one *ESS* entry and a small *EFBC*. Applications like `anagram` and `statemate`, which exhibit a uniform code access pattern, see progressively better performance with increased *EFBC* size, requiring higher memory usage to bring the overhead below 100%, marking them as the worst performers. The cases of `dijkstra` and `susan` are notable exceptions. `dijkstra` is the slowest application but also the smallest, making its access pattern distribution irrelevant. Conversely, `susan` is by far the largest binary tested, with a strided text access pattern that performs poorly with increased *EFBC* size. Although `susan` performance remains suboptimal and memory-demanding, it achieves better results with less memory compared to applications with uniformly distributed access patterns. To sum it up, Umbra manages to guarantee secure code execution with overheads that range from reasonable to negligible, depending on the available memory of the device, making robust TEEs practical on modern microcontrollers.

IV. Conclusions

In this paper, we introduced Umbra, a TEE framework designed for modern microcontrollers utilizing ARM TrustZone-M features. Umbra enables the creation of secure environments, or enclaves, and provides confidentiality and integrity guarantees at a reasonable cost by overlapping costly validation operations whenever possible.

References

[1] P. Chiavassa, F. Gandino, R. Ferrero, and J. T. Mühlberg, "Secure Intermittent Computing with ARM TrustZone on the Cortex-M," in *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2024, pp. 160–168.

[2] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *16th International Workshop on Worst-Case Execution Time Analysis*, 2016.

[3] A. Khurshid, S. D. Yalew, M. Aslam, and S. Raza, "ShieLD: Shielding Cross-zone Communication within Limited-resourced IoT Devices running Vulnerable Software Stack," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1031–1047, 2022.

[4] S. Pinto, H. Araujo, D. Oliveira, J. Martins, and A. Tavares, "Virtualization on TrustZone-enabled microcontrollers? Voilà!" in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 293–304.