SafeTEE: Combining Safety and Security on ARM-based Microcontrollers

Martin Schönstedt
Technical University of Darmstadt
Darmstadt, Germany
martin.schoenstedt@sanctuary.dev

Ferdinand Brasser
Technical University of Darmstadt
Darmstadt, Germany
ferdinand.brasser@sanctuary.dev

Patrick Jauernig
Technical University of Darmstadt
Darmstadt, Germany
patrick.jauernig@sanctuary.dev

Emmanuel Stapf
Technical University of Darmstadt
Darmstadt, Germany
emmanuel.stapf@sanctuary.dev

Ahmad-Reza Sadeghi
Technical University of Darmstadt
Darmstadt, Germany
ahmad.sadeghi@trust.tu-darmstadt.de

***Abstract*—From industry automation to smart home, embedded devices are already ubiquitous, and the number of applications continues to grow rapidly. However, the plethora of embedded devices used in these systems leads to considerable hardware and maintenance costs. To reduce these costs, it is necessary to consolidate applications and functionalities that are currently implemented on individual embedded devices. Especially in mixed-criticality systems, consolidating applications on a single device is highly challenging and requires strong isolation to ensure the security and safety of each application. Existing isolation solutions, such as partitioning designs for ARM-based microcontrollers, do not meet these requirements.**

In this paper, we present SafeTEE, a novel approach to enable security- and safety-critical applications on a single embedded device. We leverage hardware mechanisms of commercially available ARM-based microcontrollers to strongly isolate applications on individual cores. This makes SafeTEE the first solution to provide strong isolation for multiple applications in terms of security as well as safety. We thoroughly evaluate our prototype of SafeTEE for the most recent ARM microcontrollers using a standard microcontroller benchmark suite.

***Index Terms*—safety, embedded, ARM, TrustZone, TEE**

I. Introduction

Embedded devices play a key role in numerous application areas like automotive, industry 4.0 or smart home. The number of embedded devices is steadily increasing which is exemplified in the automotive industry. Electronics now account for over 40% of the production cost of a car, a figure that has doubled in just ten years [1].

Today, embedded devices are often used to run a single application. This exclusivity ensures proper isolation both for safety-critical applications (e.g., airbag or braking system) and security-critical applications (e.g., vehicle access or remote software update) but increases the production cost. Yet keeping costs low is an important goal across all industries.

To minimize cost, a more cost-effective approach to integrate the increasing number of applications and services is required, which enables the consolidation of multiple applications on a single embedded device in order to reduce the overall number of embedded devices. Additionally, the communication between applications is accelerated when they are consolidated since all communication happens on the same chip. However, when applications of mixed criticality share computing and memory resources, a number of challenges arise, as strong isolation between the applications needs to be guaranteed.

For security-critical applications, a shared system results in an increased attack surface. If one application is compromised, the system's shared nature potentially allows an attacker to also access the sensitive data of other applications on the system. Thus, a strong isolation between applications and their data must be provided by the system. Besides preventing run-time attacks on applications, mechanisms must be provided that allow the verification of the current state of an application, e.g., by using secure boot or local/remote attestation capabilities.

For safety-critical applications, a shared system can lead to interference between applications. A failure in one (non-safety-critical) application might lead to a fatal error, preventing the operating system (OS) from correctly executing a safety-critical application. In this case, it is not possible for the safety-critical application to mitigate the fatal failure since it is caused by a different application which does not adhere to high safety standards. Executing applications on individual processor cores does not solve the problem completely since all cores typically still share system resources, e.g., all cores have access to the same physical memory, and thus, interference might still occur.

To solve these challenges and enable mixed-criticality systems on ARM-based microcontrollers, an architecture is needed which fulfills both the requirements of safety-critical and security-critical applications. In recent years, multiple designs where proposed by academia to approach this problem, but no approach meets all requirements to provide isolation for safety and security applications sufficiently. While designs built for single-core devices fail to meet the isolation required by safety-critical applications [2], [3], designs with multi-core options only provide unidirectional isolation [3], [4]. The only previously proposed design to enable mixed-criticality systems on ARM-based processors relies on hardware-virtualization features, which are not available on microcontrollers [5].

In this paper, we propose SafeTEE, the first architecture

978-3-9819263-6-1/DATE22/©2022 EDAA

which combines safety and trusted execution environments (TEEs) to enable mixed-criticality systems on a single embedded device. SafeTEE isolates applications on ARM-based microcontrollers (containing ARM Cortex-M processors) which are most widely used for small embedded devices. SafeTEE targets multi-core devices and assigns cores exclusively to applications. In order to prevent any interference between applications, and to meet strong safety requirements, SafeTEE leverages the TrustZone-M technology available on ARM Cortex-M processors in a novel way to dynamically configure the memory regions each processor core is allowed to access. Moreover, SafeTEE still allows to use the TrustZone-M features in their intended way to offer additional security services to applications and thus, also meets the requirements of security-critical applications. We implemented a prototype of SafeTEE on a simulated ARM MSP2+ board and evaluated our prototype using the embench benchmarking suite [6]. We compare SafeTEE to an unmodified system and show that SafeTEE achieves a 2.5% higher benchmark score on average.

II. Background on ARM TrustZone-M

In this section, we introduce the TrustZone technology for ARM Cortex-M processors since it is a key technology in SafeTEE's design. TrustZone is the prevalent technology used to protect security-critical applications on today's ARM-based systems, first in Cortex-A processor and recently on the smaller Cortex-M processors [7], [8]. On an abstract level, TrustZone-M, depicted in Figure 1, splits the system into two separated worlds: A *secure* world and a *normal* world. The secure world is strongly isolated from the normal world, which prevents applications running in the normal world from manipulating or interfering with the secure world.

A Cortex-M processors' two execution modes called *thread* mode and *handler* mode are available both in the secure and normal world. The execution state of a processor is defined through a combination of the *world* the processor currently executes in and the execution *mode*.

In a TrustZone-enabled microcontroller, a security state is assigned to each memory address. TrustZone prevents access to secure memory addresses from the normal world. This guarantees not only the isolation of secure data and memory-mapped devices, but also prevents normal-world applications from jumping to code belonging to a secure application.

Even though the normal and secure world are strongly separated, secure-world applications can provide security services to normal-world applications by using a third security state, called non-secure callable (NSC). The normal world can call secure functions whose entry point is within NSC memory.

The assignment of a security state to a memory region is done using the Secure Attribution Unit (SAU) hardware component, which is included in every TrustZone-enabled ARMv8-M processor. The SAU can define up to eight memory regions of configurable size. These memory regions can either be marked non-secure (belonging to normal the world) or NSC. All other memory regions are implicitly assigned the secure state (belonging to the secure world). The SAU configuration registers can only be modified from within the secure world.

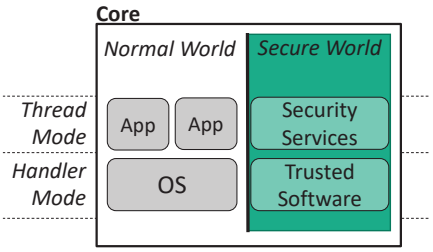

Fig. 1. A typical TrustZone-M system on a single processor core, running an OS with its applications in the normal world and trusted software and security services in the secure world.

Figure 1 shows a typical TrustZone setup on a single-core device. Applications are executed in the normal world, using the handler mode for privileged parts of an OS or real time operating system (RTOS) and the thread mode for unprivileged applications. Security services like attestation are running in the thread mode of the secure world. The secure-world handler mode contains the trusted software, which is the first software running after boot. The trusted software configures the SAU and other system resources before any normal-world software is executed. Moreover, the trusted software enables the implementation of further security features like secure boot.

III. System and Adversarial Assumptions

In the following section, we describe the system we target in SafeTEE, and we detail our assumed adversary model.

A. System Assumptions

SafeTEE targets ARM microcontrollers that contain multiple Cortex-M cores. The availability of multi-core chips is a practical assumption since the number of processor cores per chip has been steadily increasing on ARM-based chips in the last decade; an example of a platform implementing dual-core Cortex-M processors is the ARM Musca Board [9]. We further assume that each processor core is TrustZone-M enabled.

We assume that SafeTEE is used to consolidate applications of mixed-criticality on one embedded device. Some applications are assumed to be safety critical, while others are security-critical. Taking an example from the manufacturing industry, the safety-critical application could be represented by the automation software of an industrial robot and the security-critical application by software which transmits sensitive system data from the robot to a server using encrypted communication.

We assume that all safety-critical applications on the system are built in compliance with industry standards, such as IEC 61508 [10], which defines generic functional safety guidelines for electrical, electronic, and programmable electronic systems, or industry specific standards, such as ISO 26262 [11] (automotive) or IEC 61511 [12] (manufacturing industry). For the security-critical applications we do not assume that they are built in accordance to such standards.

B. Adversary Model

Our adversary model adheres to the one commonly assumed for TEE architectures in general and for TrustZone-based architectures in particular [7], [13]–[17]. We assume a strong

software-only adversary that can compromise all software in the normal world (thread and handler mode). The software in the secure world however, which represents the system's trusted computing base (TCB) and which is provided by the system vendor, is inherently trusted and verified with secure boot.

The goal of the adversary is to leak secret data from security-sensitive applications and/or the TCB, or to provoke failures of safety-critical applications. The adversary is able to inject own code into the OS running in handler mode, and thus, also able to deploy malicious applications. Unlike most other works, we also consider denial of service (DoS) attacks, which the attacker might use to starve a safety-critical application. As attack vectors for DoS attacks we consider manipulation of the safety-critical application's memory and attacks on the processor core which is executing the safety-critical application, e.g., by provoking a fatal error to prevent execution.

We assume the underlying hardware to be correct and trusted, and hence, do not consider attacks that exploit hardware flaws [18], [19]. Further, we assume that the attacker does not have physical access to the device, and thus, fault injection attacks [20] and physical side-channel attacks [21], [22] are out of scope. As standard for TEE architectures, SafeTEE does not protect from memory corruption vulnerabilities inside a security-critical application but prevents their exploitation from compromising other applications on the system. Since SafeTEE targets ARM-based microcontrollers which typically do not include caches or direct memory access (DMA) capable devices, we do not consider cache side-channel attacks [23], [24] or DMA attacks [25].

IV. Design

In this section, we first show the high-level design of SafeTEE. Next, we give details on the underlying hardware primitives used in the design and the work flow of setting up a safety- or security-critical application.

A. High-level Overview

SafeTEE provides a novel design for a security and safety architecture for ARM microcontrollers which meets the safety and security requirements of mixed-criticality systems.

In SafeTEE, whose high-level design is shown in Figure 2 exemplarily for a dual-core setup, all applications run on dedicated physical processor cores in the normal world, whether they are safety critical (NWP 0) or security critical (NWP 1). Each application, which we call a normal-world partition (NWP), can thus utilize both processor modes, the thread and handler mode. Dedicating each core to only one NWP is not considered a drawback, as SafeTEE aims to consolidate applications else implemented on individual embedded devices. Running all applications unprivileged in the normal world keeps the secure world usable for security services, e.g., key management or remote attestation. In SafeTEE, an NWP can access security services the same way they are used on existing TrustZone-enabled microcontrollers. Thus, existing security-critical applications can easily be ported to SafeTEE.

TrustZone's secure world can only provide a single protected environment, thus, applications running in the secure world

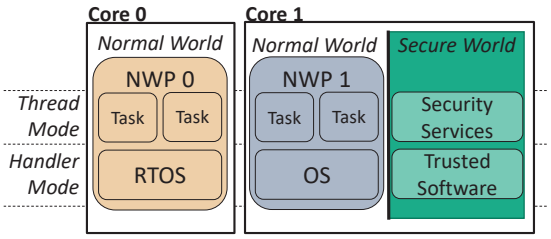

Fig. 2. SafeTEE on a dual-core device, highlighting different usage possibilities for NWPs. The first core and its NWP execute an RTOS. The second core executes an NWP running a general-purpose OS with access to security services managed by the trusted software, executing in the same core's secure world.

cannot be strongly isolated from each other, which is a much-debated general limitation of TrustZone [14], [15], [26], [27]. In SafeTEE, we therefore extract all security-critical applications from the secure world and run them in unprivileged NWPs in the normal world. The secure world only contains security services and SafeTEE's firmware (trusted software in Figure 2) provided by the system vendor.

In Figure 2, we show that only the security-critical application (NWP 1) has access to the depicted security services. The safety-critical application (NWP 0) does not require security services during runtime and thus, never switches to the secure world. As its execution is never interrupted, the safety-critical application can meet its timing requirements in the same way as an implementation on an exclusive microcontroller. Instead of including an RTOS in an NWP which executes multiple tasks, safety-critical applications can also be run bare-metal.

Since every NWP runs exclusively on a single core or multiple cores, the NWP's computing resources are physically isolated from each other. This is necessary for enabling safety-critical applications, as they require strong partitioning to prevent interference between applications. Partitioning of memory resources is achieved by utilizing TrustZone's hardware primitives which we describe next.

B. Hardware Primitives

Each NWP's memory is stored in a separate partition consisting of multiple memory regions, memory-mapped IO (MMIO) devices are exclusively assigned to an NWP. SafeTEE defines memory access policies, so that every NWP can only access its own memory regions. Partitioning ensures the integrity and confidentiality of memory belonging to an NWP, meeting the requirement for security-critical applications and also for safety-critical applications for which partitioning is also recommended by industry standards [10].

In order to create and isolate an NWP, the SAU hardware component is used, which is included in every TrustZone-enabled Cortex-M processor. In Figure 3, we show in more detail how the SAU is configured for each processor core and how the general boot process and setup phase of SafeTEE work. Each core's SAU defines only the memory regions belonging to the NWP running on that core as non-secure (belonging to the normal world). Thus, NWP 0 can only access the memory regions defined by the SAU on core 0 (NWP Code, Data and Device as shown in Figure 3). Memory accesses to

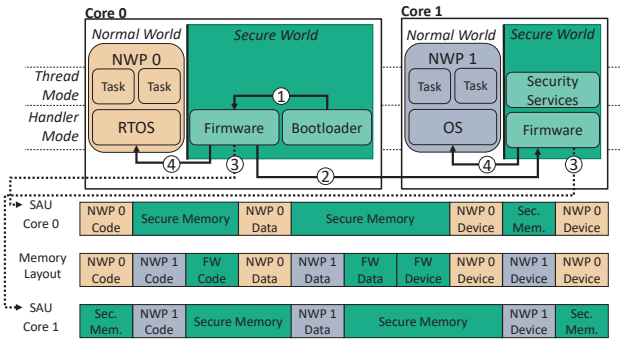

Fig. 3. The setup steps for SafeTEE shown on a dual-core system. First, the bootloader is executed on core 0, afterwards the firmware boots the second core. On both cores the firmware (FW) then configures the SAU regions before starting the NWPs.

addresses belonging to NWP 1 are prevented. Based on core 0's SAU configuration, NWP 1's memory regions and all memory regions of the secure world (FW Code, Data and Device) are regarded as secure memory. If it is accessed from NWP 0, TrustZone triggers a SecureFault [7].

The configuration of the SAU is part of the SafeTEE setup phase. Its steps are highlighted in Figure 3. After an initial secure bootloader has loaded and verified all images (including all safety- and security-critical applications), a small firmware is executed to configure the system ①. The firmware and bootloader together form the trusted software (Figure 2). As the initial boot process can be performed on a single core, other cores only start after the bootloader is finished. Booting the other cores is done by the firmware ②. The next steps are executed on all cores. First, the firmware configures each core's SAU ③. The firmware can also assign MMIO devices to a particular NWP (NWP 0 Device, NWP 1 Device) or the secure world (FW Device). The final step of the system setup is to start the NWPs ④. After the setup phase is complete, the firmware is responsible for managing the security services.

V. Security Considerations

SafeTEE provides isolation for applications encapsulated in NWPs. For each NWP, a memory region is configured to be exclusively assigned to it, i.e., an NWP's private memory region is only accessible by that NWP and the secure world. The secure world is trusted (cf. Section III-B). Thus, the memory of an NWP provides integrity and confidentiality for all data and code of an NWP during run time.

The initial integrity and confidentiality of NWP's code and data is ensured by the secure loading procedure of SafeTEE, i.e., NWPs are loaded by the trusted software (firmware + bootloader) at boot time. Before any NWP can execute, the memory isolation for the NWPs is configured and SafeTEE's memory isolation is active before any untrusted NWP component is allowed to execute, hence, a NWP can never access another NWP's code or data. This prevents memory manipulation as an DoS attack vector.

NWPs are bound to dedicated processor cores, thus, the actions of one NWP can only impact the execution of that NWP itself, e.g., invoking security services will cause that NWP's processor to switch to the secure world. All other NWPs can execute independently on their own processor cores, thus, SafeTEE prevents interference between NWPs. All resources that are shared between processor cores are handled by SafeTEE's firmware, preventing for instance inter-processor interrupts. Through this strict resource separation, SafeTEE provides DoS protection for the attack vectors mentioned in Section III-B. Even if the adversary has control over the handler mode of one NWP, the adversary cannot impact the execution of any other NWP on the platform.

VI. Implementation

We implemented a prototype of SafeTEE on an ARM-based microcontroller. Two currently available hardware platforms by ARM Ltd. meet the requirements described in Section III-A. Both implement the SSE-200 subsystem [28] with two Cortex-M33 processors supporting TrustZone. The Musca-B1 test chip board and the Arm MPS2+ FPGA prototyping board [29], where the AN521 FPGA image implements the subsystem on the prototyping board [30]. A white paper presented in 2013 by ARM Ltd. already contained design guidelines for a multi-core microcontroller design [31]. While ARM Ltd. has been leading the way for dual-core microcontrollers, other vendors have also introduced dual-core Cortex-M-based boards [32].

Unfortunately, physical versions of the suited boards could not be obtained on the market. However, the MPS2+ with AN521 image can be simulated using QEMU, which is a widely-used open-source tool for system emulation. In our prototype implementation, we use Zephyr v2.5 [33] and multiple tasks for representing the safety- and security-critical applications (NWPs) which run in the normal world of the MPS2+ AN521 board.

In order to achieve SafeTEE's memory partitioning design, the secure software has to run on both processor cores and configure the correct SAU regions of each core. We modified the Trusted Firmware-M (TF-M) v1.2.0 to use it as the firmware in our prototype. The TF-M performs the secure boot and system initialization, including the SAU configuration on each core. Moreover, the TF-M manages the security services. We implement a key management security service in our prototype which can be used by an NWP. The security service stores a key for a one-time-pad in its secure memory. When an NWP calls the security service with a plaintext for encryption, the service returns the ciphertext while keeping the key secured.

VII. Evaluation

In our evaluation, we first verify whether SafeTEE provides the required isolation between multiple applications running on one system. Then, we evaluate SafeTEE's performance using the microcontroller benchmark suite embench [6].

For the isolation evaluation, we implement a setup were one application performs a failure and then observe whether it impacts other applications on the system. We compare SafeTEE, running two applications in own NWPs, with two variants of a system that does not follow our design approach. In the first variant, we implement each application as an own thread within

TABLE I
RESULTS OF EVALUATION TESTS

	Legacy OS single-core	Legacy OS AMP	SafeTEE NWPs
Fault Isolation	×	✓	✓
Memory Isolation	×	×	✓

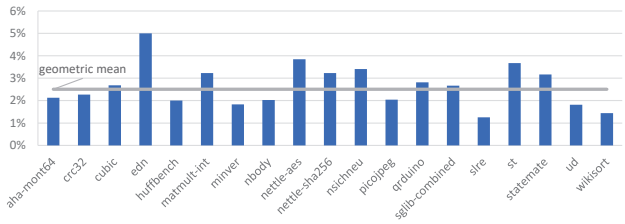

Fig. 4. Relative increase of benchmark score when using SafeTEE.

Zephyr, running on a single core. As a second variant, we use an asymmetric multiprocessing (AMP) implementation of Zephyr to run each application on an exclusive processor core. We evaluate two scenarios which test the possibility of interference between the applications. We show the results in Table I.

a) Scenario - Isolation for Safety: In the first scenario, we purposely generated a fatal error in one application to see if the other application is still able to execute afterwards. Isolation from failures in other applications is necessary for safety-critical applications. As visible in Table I, isolation is only achieved by the AMP dual-core system and SafeTEE. The results show that processor core sharing should be avoided for safety-critical applications.

b) Scenario - Isolation for Security: The second scenario is relevant for both safety- and security-critical applications. In a shared system, applications should not be able to access other application's memory. For security-critical applications, this could result in attacks on the confidentiality and integrity of sensitive data. While confidentiality is not a big concern for safety-critical applications, data integrity must be ensured. Manipulations might corrupt the application's state and lead to incorrect behavior. In the context of our evaluation, one application tries to modify data belonging to the other application. As shown in Table I, only SafeTEE can provide adequate memory protection. The other systems both allowed reading and writing data belonging to the other application. When using SafeTEE, memory accesses are prevented, a fault is generated and the application's execution is halted. This is useful because illegal memory accesses are only expected by damaged or malicious applications. Both of which should not continue their execution after an illegal behavior is observed.

c) Performance Evaluation: For evaluating the performance of SafeTEE, we performed ten runs of the benchmarks provided by embench v1.0 [6], as used in related work [34], on the same QEMU simulation setup both with and without SafeTEE. The relative differences between the benchmark scores of both runs are presented in Figure 4. SafeTEE's runs achieved better benchmark scores than runs without SafeTEE. On average, SafeTEE increases the benchmark scores by 2.5% geometric mean. However, we attribute the increased scores to internal optimizations in the simulation process of a dual-core QEMU setup. As SafeTEE only produces a constant overhead (during system setup), the run time of NWPs does not differ from systems without SafeTEE. Hence, neither a performance increase nor decrease are expected on physical boards.

VIII. RELATED WORK

In this section, we present and compare relevant existing approaches for partitioning on ARM-based platforms and highlight why none of them meet the requirements for mixed-criticality systems on embedded devices.

While our design is built for embedded systems with Cortex-M processors, a variety of partitioning designs exist for the more powerful Cortex-A platforms. The architectures of the different ARM processor families share similarities, so general design principles could be applied to and compared between Cortex-A and Cortex-M processors.

A. TrustZone- and MPU-based Isolation

One design possible on all processors with TrustZone uses the isolation between normal and secure world to isolate one application in the secure world, while another one is executed in the normal world. This is implemented by LTZVisor for Cortex-A [4]. Similar designs for Cortex-M devices exist [3], [34]. However, switching between secure world and normal world on the very same core is problematic in real-time scenarios, as the scheduling deadlines cannot be enforced any more when in secure world. Diverging from this concept and assigning secure and normal world to individual cores can address this issue, but is inflexible. Since TrustZone only offers two isolated domains, this approach would be limited to two applications even if more cores were available. Using the secure world for an application further prevents usage of TrustZone for actual security services.

MultiZone is a security architecture for Cortex-M systems [2]. Applications are scheduled by the MultiZone kernel and executed on a single core. Instead of virtualization, MultiZone uses the Memory Protection Unit (MPU) available on the processors to restrict the memory to the currently scheduled application. However, MultiZone's processor sharing implies that a crashing application impacts the other applications, making it unsuitable for safety-critical scenarios. Further, using the MPU for isolation also limits applications to using only the thread mode, which is a limitation SafeTEE does not have.

B. Virtualization-based Isolation

An alternative are virtualization-based approaches on Cortex-A processors, e.g., Hafnium [35] or Siemens' Jailhouse [36]. These approaches allow the execution of applications on individual logical cores. While Jailhouse targets the normal world, Hafnium can be used for normal-world and secure-world virtualization (SEL-2) to isolate applications in virtual machines (VMs). These VMs can be scheduled dynamically, managed by a primary VM. Thus, the designs can only guarantee availability of the primary VM. This prioritization is not suitable for the

use cases targeted by this work's design. The static design of SafeTEE instead guarantees availability of all applications.

Bao [5] is a static, low-overhead, processor-exclusive design for Cortex-A processors. Similar to SafeTEE, Bao also targets safety-critical applications. Like other designs for Cortex-A processors, Bao relies on virtualization for isolation between applications. Porting the design of Bao to embedded Cortex-M systems is not possible because the smaller processors lack the necessary virtualization features.

To conclude, existing designs for Cortex-M systems do not meet the requirements of safety-critical applications, and thus, are not suited for mixed-criticality systems targeted by SafeTEE. In contrast, Cortex-A approaches either use temporal isolation and cannot meet safety requirements, or rely on a hypervisor—which is not present on Cortex-M processors. SafeTEE is the first architecture for Cortex-M system which addresses these issues and enables isolation of mixed-criticality applications on a single embedded device.

IX. Conclusion

We presented the design and implementation of SafeTEE, the first architecture to offer strong isolation for consolidation of safety-critical applications on a single platform, thus, enabling mixed-criticality systems on Cortex-M microcontrollers. SafeTEE prevents interference between safety- and security-critical applications by encapsulating them in normal-world partitions (NWPs), which are assigned to individual processor cores, have separated memory regions assigned and exclusive access to devices. Our evaluation shows that the performance of applications isolated within NWPs is not negatively impacted.

Acknowledgment

We thank our anonymous reviewers for their valuable and constructive feedback. This work was funded by the Federal Ministry of Education and Research in the StartUpSecure funding program (16KIS1417).

References

[1] Deloitte, "Semiconductors – the Next Wave," press release. [Online]. Available: https://www2.deloitte.com/cn/en/pages/about-deloitte/articles/pr-semiconductors-the-next-wave-2019.html

[2] S. Pinto and C. Garlati, "Multi zone security for arm cortex-m devices," 2020. [Online]. Available: https://hex-five.com/wp-content/uploads/2020/02/Multi-Zone-Security-White-Paper-20200224.pdf

[3] S. Pinto, H. Araujo, D. Oliveira, J. Martins, and A. Tavares, "Virtualization on trustzone-enabled microcontrollers? voilà!" in *RTAS*, 2019.

[4] S. Pinto, J. Pereira, T. Gomes, A. Tavares, and J. Cabral, "Ltzvisor: Trustzone is the key," in *ECRTS*, 2017.

[5] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *NG-RES*, 2020.

[6] Free and Open Source Silicon Foundation. Embench: A Modern Embedded Benchmark Suite. [Online]. Available: https://www.embench.org/

[7] Arm Ltd., *Arm TrustZone Technology for the ARMv8-M Architecture*. [Online]. Available: https://developer.arm.com/documentation/100690/0201/

[8] P. Burr. (2018). [Online]. Available: https://community.arm.com/developer/ip-products/processors/trustzone-for-armv8-m/b/blog/posts/nordic-announce-first-cortex-m33-based-chip-with-trustzone

[9] *Musca-B1 Test Chip Board*, Arm Ltd. [Online]. Available: https://developer.arm.com/tools-and-software/development-boards/iot-test-chips-and-boards/musca-b-test-chip-board

[10] International Electrotechnical Commission. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements. [Online]. Available: https://webstore.iec.ch/publication/5515

[11] International Organization for Standardization. ISO 26262-1:2018(en) Road vehicles – Functional safety – Part 1: Vocabulary. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-2:v1:en

[12] International Electrotechnical Commission. Functional safety - Safety instrumented systems for the process industry sector - ALL PARTS. [Online]. Available: https://webstore.iec.ch/publication/5527

[13] Intel, "Intel Software Guard Extensions Programming Reference," https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, 2014.

[14] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *SOSP*, 2017.

[15] F. Brasser, D. Gens, P. Jauernig, A. Sadeghi, and E. Stapf, "Sanctuary: Arming trustzone with user-space enclaves," in *NDSS*, 2019.

[16] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX Security*, 2016.

[17] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "CURE: A Security Architecture with CUstomizable and Resilient Enclaves," in *USENIX Security Symposium*, 2021.

[18] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, 2014.

[19] A. Tang, S. Sethumadhavan, and S. Stolfo, "Clkscrew: exposing the perils of security-oblivious energy management," in *USENIX Security*, 2017.

[20] I. Biehl, B. Meyer, and V. Müller, "Differential fault attacks on elliptic curve cryptosystems," in *CRYPTO*, 2000.

[21] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *CRYPTO*, 1996.

[22] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008.

[23] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *RSA Conference*, 2006.

[24] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack." in *USENIX Security*, 2014.

[25] A. T. Markettos, C. Rothwell, B. F. Gutstein, A. Pearce, P. G. Neumann, S. W. Moore, and R. N. Watson, "Thunderclap: Exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals." in *NDSS*, 2019.

[26] H. Sun, K. Sun, Y. Wang, J. Jing, and H. Wang, "Trustice: Hardware-assisted isolated computing environments on mobile devices," in *DSN*, 2015.

[27] S. Wan, M. Sun, K. Sun, N. Zhang, and X. He, "Rustee: developing memory-safe arm trustzone applications," in *Annual Computer Security Applications Conference*, 2020, pp. 442–453.

[28] Arm Ltd., *Arm® CoreLink™ SSE-200 Subsystem for Embedded - Technical Reference Manual*. [Online]. Available: https://developer.arm.com/documentation/101104/0100/

[29] *Arm MPS2+ FPGA prototyping board*, Arm Ltd. [Online]. Available: https://developer.arm.com/tools-and-software/development-boards/fpga-prototyping-boards/mps2

[30] Arm Ltd., *Application Note AN521 SMM-SSE-200 for MPS2+*, 2017. [Online]. Available: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/DAI0521A_example_sse200_subsystem_for_v2m_mps2.pdf

[31] J. Yiu and I. Johnson, "Multi-core microcontroller design with cortex-m processors and coresight soc," ARM Ltd., Tech. Rep., 2013.

[32] NXP Semiconductors. i.MX RT1170 Crossover MCU Family - First GHz MCU with Arm Cortex-M7 and Cortex-M4 Cores. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus/:i.MX-RT1170

[33] Zephyr Project members and individual contributors. Zephyr Project. [Online]. Available: https://docs.zephyrproject.org

[34] D. Oliveira, T. Gomes, and S. Pinto, "utango: an open-source tee for the internet of things," *arXiv preprint arXiv:2102.03625*, 2021.

[35] Linaro Ltd. (2021). [Online]. Available: https://www.trustedfirmware.org/projects/hafnium/

[36] Siemens AG. Jailhouse - Linux-based partitioning hypervisor. [Online]. Available: https://github.com/siemens/jailhouse