Multi Zone Security for Arm Cortex-M Devices

Sandro Pinto
Universidade do Minho
Guimarães, Portugal
sandro.pinto@dei.uminho.pt

Cesare Garlati
Hex Five Security
Redwood City, CA, USA
cesare.garlati@hex-five.com

***Abstract* – Developing secure Internet of Things devices is becoming more and more difficult. Complex functional requirements are increasingly met with the addition of untrusted 3rd party components. The resulting monolithic firmware presents vastly larger code base, greater attack surface, and increased system vulnerability. In addition, cost and low-power requirements lead to resource-constrained microcontroller architectures. These lack basic hardware security mechanisms and the ability to separate multiple trusted applications from less critical components. A new zero-trust model is required to address the intrinsic security threat posed by the resulting multi-source monolithic firmware. In this paper, we propose a novel approach to embedded security based on hardware-enforced, software-defined separation of multiple, equally secure, functional domains. We start by analyzing why the traditional "two-worlds" model is no longer suitable for modern IoT applications. We then introduce the concept of a lightweight, multi zone, trusted execution environment capable of enforcing security and separation for a multitude of equally-secure functional domains. Finally, we explain the details of the actual implementation of this model in Arm Cortex-M7 processors.**

Keywords— IoT, Security, Isolation, Separation, Trusted Execution Environment, TEE, Firmware, Arm, Cortex-M, TrustZone.

I. Introduction

The Internet of Things (IoT) is comprised of billions of interconnected devices that, by definition, are exposed to remote attack – potentially resulting in the most damaging type of cyber threats: distributed denial of service or DDOS. While early design concerns were mostly related to connectivity and interoperability, a multitude of recent high-profile cyberattacks has shown that the success of this new Internet era is heavily dependent on the trust and the security built into these devices [1, 2].

However, the attack surface in IoT devices is growing exponentially as sophisticated connectivity requirements are increasingly met with the integration of a multitude of 3rd party software libraries, open source real-time operating systems, and proprietary black-box binaries – i.e. peripherals drivers [2]. The problem is exacerbated by the lack of simple and reliable mechanisms to enforce separation among these multi-source mixed-criticality components. This inexorably leads to unsafe systems that indiscriminately run all software components at the same level of privilege sharing the same memory space and peripherals. Due to the monolithic nature of the firmware embedded into these systems, the exploit of a vulnerability in one low-criticality component is usually enough to lead to privilege escalation, lateral movement to higher-criticality modules, and to complete system failure – see for example the recent high-profile attacks to two of the most widespread embedded operating system, FreeRTOS and VxWorks, that were compromised by exploiting vulnerabilities in network-related components [3, 4].

More expensive and power-hungry processors provide some features intended to control access to memory-mapped resources – i.e. virtual memory and Memory Management Unit (MMU). However, these mechanisms are far from optimal as they present significant drawbacks with regard to the implementation of virtual memory management and MMU. Firstly, they require more complex hardware (e.g., silicon gates and TLBs) and software (e.g., 2-stage translation tables) that add to cost and power consumption. Secondly, the additional complex software layers necessary to drive the MMU vastly increase the total codebase (TCB) of the system, resulting in larger attack surface and ultimately in less secure systems. As all products of the human intellect, software is intrinsically subject to defects and statistically likely to present unexpected behaviors – generally referred as "bugs". Thus, the resulting increase in TCB inevitably leads to a higher number of potential vulnerabilities and in the end to a less resilient system. Resource-constrained microcontrollers used in IoT applications typically have simplified versions of the MMU – i.e. Memory Protection Unit (MPU) – that are equally exposed to the complexity of the additional software required to operate them – in Brandon Lewis' words: "the design complexity associated with properly implementing these technologies often results in them not being used at all" [5].

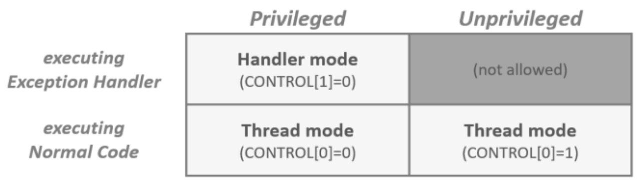
We reject the very idea that simply throwing more silicon area at the security problem - as in adding more hardware security blocks - makes any system more "secure". In fact, we argue the exact opposite: when it is comes to security "less is more". Simpler hardware means simpler software, less lines of code, statistically less defects, and in the end more resilient systems. Traditional Trusted Execution Environments that "carve-out" one safe area across the various hardware components - commonly referred as the "secure world" – have been available in the market since 2004 but never achieved widespread adoption outside mobile telephony because of two significant limitations. Firstly, they rely on proprietary hardware extensions typically not available across all vendors and platforms. Secondly, they are admittedly overly complex, very difficult to understand, and even more difficult to properly implement across disparate silicon architectures. Finally, over the last few years, confidence in these systems has been shaken due to the systematic discovery of many critical vulnerabilities [6, 7]. For example, a recent vulnerability study of popular commercial TrustZone-assisted TEE systems has revealed that these systems have (i) a long history of critical implementation bugs, (ii) numerous architectural deficiencies, and (iii) important hardware properties that are overlooked [6].

In this paper, we propose a novel approach to hardware security based on hardware-enforced, software-defined separation of multiple, equally secure, functional domains. We start by explaining why the "secure world" model is no longer enough to meet the requirements of modern IoT applications and why a new zero trust model is required. Then, we share the details of the research work underpinning the development of the first commercial multi zone TEE for Arm(v7-M) specifically designed to bring security and separation to resource-constrained Cortex-M devices. Finally, we present a system evaluation and a reference implementation targeting an industry widespread Cortex-M7 microprocessor.

II. A New Zero Trust Model

The vast majority of the electronic devices in the market are not completely developed in-house by any single vendor. They are rather composed of a myriad of 3rd party hardware and software components combined with some proprietary IP to meet the specific functional and budgetary constraints of the particular product. Third party software components are often available in the form opaque object code libraries or black-box binaries. The lack of transparency poses a significant threat vector for embedded firmware, as the product may ship with intentional or unintentional vulnerabilities, leading to exploit and attacks – see Cisco routers "unauthorized" firmware incidents. When the 3rd party code is linked into the monolithic firmware image without any functional separation, a single defect or vulnerability can compromise the whole system. We believe this practice is flawed at its core. All systems components, and especially the ones provided by 3rd parties, should be assumed defective and therefore untrusted. This leads to a new design paradigm based on the concept of Zero Trust: no single functional block should have indiscriminate access to all system resources and, therefore, no single functional block should be able to intentionally or unintentionally compromise the CIA - Confidentiality, Integrity, Availability - of the whole system.

The Zero Trust Model described above requires hardware-enforced separation between each primary threat vector and the core of the system. If we take into consideration the basic functional blocks typically present in embedded connected devices (e.g., smart sensor and IoT endpoints), we can typically identify the following elements: (i) a Real Time Operating System (RTOS) running a set of software threads (tasks); (ii) a set of sensors and actuators to control external physical systems; (iii) a network interface exposed to remote attack; and (iv) a set of cryptographic algorithms used to secure data at rest and in motion and to provide attention services to a host. Each of these elements is likely to contain 3rd party code. It should be untrusted and isolated from the other parts of the system.



	Privileged	*Unprivileged*
executing Exception Handler	**Handler mode** (CONTROL[1]=0)	(not allowed)
executing Normal Code	**Thread mode** (CONTROL[0]=0)	**Thread mode** (CONTROL[0]=1)

Fig. 1 - Armv7-M operation modes and privileged levels

Implementing the Zero Trust Model above requires at least four different blocks of functional separation - or "zones". MMUs allow separation only between user and kernel functions – several of these blocks are traditionally built inside a monolithic infrastructure and thus very difficult to separate and protect using virtual memory and MMU. In addition, traditional TEEs based on hardware primitives such as Arm TrustZone are designed to isolate only one functional block. This is commonly referred as the "secure world" and typically used to control access to cryptographic keys and security-critical operations such as secure boot. The main RTOS and all its tasks run in the remaining single non-secure world – with no mechanism left for further separation. The intrinsic binary limitation of this antiquated architecture simply fails to provide enough levels of separation for a modern Zero Trust design.

III. Leveraging Armv7-M Hardware Primitives

Oftentimes the implementation of an embedded TEE requires additional specialized hardware, internal or external to the application processor. In practice, this makes the TEE implementation very difficult and prone to defects and vulnerabilities. The Armv7-M architecture specifies a set of hardware security primitives that make the TEE hardware enablers available out-of-the-box in almost all Cortex-M microcontrollers – with the notable exception of the tiny Cortex-M0.

A first group of security primitive available in Armv7-M cores is represented by the **privilege levels**. At any time, an Armv7-M MCU runs at a specific mode and privilege level. According to the Armv7-M Architecture Reference Manual, Cortex-M processors can have run in two modes: Handler and Thread. While the Handler mode is always privileged, the Thread mode can have privileged and unprivileged access levels (Fig. 1). The Handler mode is intended to execute exception handling code, and a bit more privileged than the Privileged Thread mode, i.e. some registers are just accessible in Handler mode (e.g., *IPSR*). The separation of privileged and unprivileged access levels allows for the development of more robust systems. It provides a basic security mechanism by controlling memory accesses to specific regions.

A second built-in security feature is the **memory protection unit (MPU).** It is optional but widely available in all Cortex-M0+/M3/M4/M7 processors. The MPU is a programmable hardware block that can be used to define permissions to specific memory regions according to privilege levels. This allows the partitioning of functionality between execution environments and the misuse of some particular resources - i.e. define RAM

Instruction	Description
MRS	Read from Special register (e.g., PRIMASK)
MSR	Write to Special register (e.g., PRIMASK)
CPSIE	Enable Interrupts/Faults
CPSID	Disable Interrupts/Faults
WFI	Wait for Interrupt
WFE	Wait for Event

Fig. 2 - List of privileged instructions that do not cause any privileged violation exception when executed in Unprivileged Thread mode

space as non-executable (eXecute Never, XN) to limit buffer misuse and prevent code injection attacks.

IV. Implementation Challenges on Armv7-m

TEEs aim at enforcing hardware isolation of multiple software components within the system while preserving Confidentiality, Integrity, and Availability - i.e., the CIA triad. A modern TEE must provide [9]: (i) isolation of code, data, interrupts, and other resources; (ii) preemptive temporal separation; and (iii) trap and emulate functionality for privileged instructions to allow transparent execution of legacy applications, which are typically not designed for running in a secure unprivileged mode. TEE requirements were probably not a major design criteria for the Armv7-M architecture. As a result, the TEE designer needs to address a few specific shortcomings.

NAPOT MPU Regions. The MPU implemented in Armv7 Cortex-M microcontrollers can support either eight or sixteen programmable memory regions, each with their programmable starting addresses (MPU_RBAR) as well as sizes and attributes (MPU_RASR). For a proper TEE implementation, we identified and addressed two limitations in the MPU design: (i) the MPU region size is encoded through a set of fixed naturally aligned power of two (NAPOT) ranges from 32 Byte to 4 GiB; and (ii) the base address of the region must be aligned to the size of the region. For example, for a region size of 32 KiB, the base address must be aligned to 15 (i.e., base address [31:15]). From a system developer perspective, this imposes hard limitations that either lead to wasting of precious memory space or, even worse, to improper security settings that leave unprotected "holes" extremely difficult to detect and correct.

Special Privileged Instructions. The Armv7-M ISA defines a well-defined set of privileged instructions. These instructions are intended to be executed at the higher level of privilege (i.e., Privileged Thread or Privileged Handler modes) and typically used to access special registers that control for example interrupt settings and power-down operations. Fig. 2 lists the identified privileged instructions from the Armv7-M ISA. The main problem with these instructions is that when executed in Unprivileged Thread mode, they fail silently instead of raising privilege violation exceptions. This behavior constitutes a significant issue for TEEs aiming at providing complete trap and emulation support for code designed to operate in unprotected memory space.

Imprecise Bus Faults. The Armv7-M architecture supports a predefined 32-bit address space, with subdivision for code, data, peripherals, and regions for on-chip and off-chip resources. According to the system address map, there is a region, so-called *System* area (0xE000_0000 - 0xFFFF_FFFF), which is reserved for system-level use. Within this predefined 512MiB range, there is a special 4KiB subregion named System Control Space (SCS), which provides registers for system configuration and status reporting and control. For example, the System Timer (SysTick) and the Nested Vectored Interrupt Controller (NVIC) are mapped to this 4KiB memory area. As a memory-mapped area, read and write operations to the SCS registers are performed through normal load and store instructions (and not through *MSR* and *MRS* instructions). However, the full *System* region is special in the sense it is a privileged memory area where MPU policies are not enforced. This means that even if the MPU is properly configured to prevent unprivileged access to this region, unprivileged read and write operations will trigger Bus faults, instead of Memory Management (MemManage) faults. The issue with Bus faults is that, depending on the specific microarchitectural implementation, they might raise imprecise exceptions – in contrast to MemManage faults that are always precise. Imprecise faults are difficult and expensive to handle as the core throws the Bus fault some cycles later, without recording the exact address and/or instruction that violated the privileged memory and that needs to be emulated.

V. Proposed Multi Zone TEE for Arm(v7-m)

Fig. 3 shows a multi zone reference implementation for Arm(v7-M) processors. The multi zone separation kernel runs at the highest privilege level (Privileged Handler Mode). Application code and interrupt handlers run in separated zones at the lowest level of privilege (Unprivileged Thread mode).

TEE Configurator. To make the system more secure, we limit the possibility for human error by encapsulating and hiding the whole complexity of managing the underlying hardware blocks [8]. The only interaction of the system developer with the TEE is a simple flat format policy definition file. No coding, compilation, linking and debugging is necessary – and in fact even allowed. Instead, we provide a simple command line utility to be used at the last step of the development cycle in the form of a JAR file toolchain extension. This small utility is written in java to make it portable across any operating system and development environment. The configurator utility combines the fully linked binaries of each zone with the pre-built TEE runtime, applies the security and separation policies defined in the configuration file, and produces the secure boot firmware image for target upload. In addition, the configurator provides (i) binary translation to address the system limitations highlighted in Section IV, (ii) full support for trap and emulation without modifying existing source code, and (iii) a

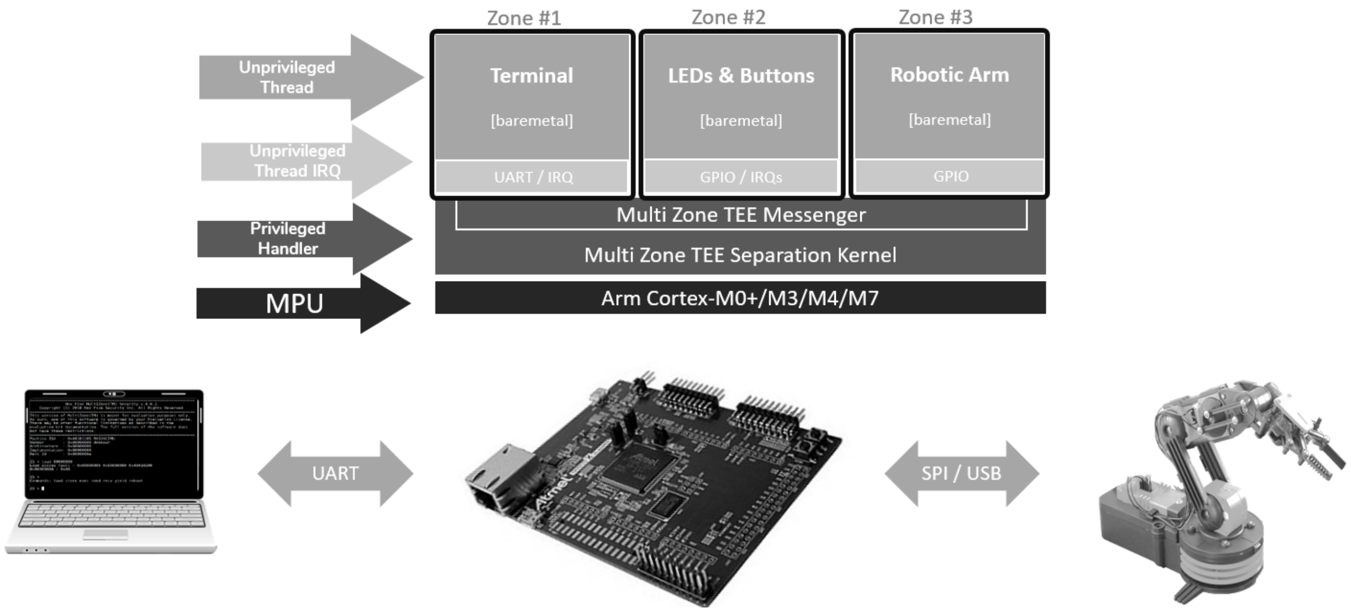


Fig. 3 - Multi Zone Trusted Execution Environment - Reference Implementation Architecture.

sophisticated MPU optimization algorithm that allows any user-defined range and size of memory mapped resources.

TEE Secure Boot Process. The TEE implements a 2-stage secure boot loader to verify the integrity and authenticity of the firmware image (SHA-256) and to minimize runtime memory footprint and target attack surface.

TEE Separation Kernel. At runtime, the tiny separation kernel provides a formally verifiable, self-contained facility for time and spatial isolation of multiple secure threads – zones [9]. The kernel supports an unlimited number of separated TEEs, called zones. Through the policy definition file, the system designer assigns a set of resources to each zone. These include any memory mapped resource such as RAM, ROM, I/O, interrupts, and relative read/write/execute access policies. The TEE kernel implements a preemptive real-time scheduler suitable to safety-critical applications with configurable round-robin and/or cooperative scheduling policies. The kernel also provides independent soft timers for each zone consistent with the Armv7-M System Timer (a.k.a. SysTick). There is full support for low-latency vectored interrupts (NVIC) and wait for interrupt (WFI) low power / suspend mode. A unique security aspect of the multi zone TEE is the ability to securely execute interrupts handlers in unprivileged Thread mode in the context of the mapped zone. To provide complete and transparent support for unmodified binaries, the kernel implements trap & emulation for most privileged system registers – including those accessible through privileged *MRS* and *MSR* instructions (e.g., MSP, BASEPRI) and those that are memory-mapped (e.g., VTOR, ICSR, CFSR in the SCS region).

TEE API. Trap and emulation is great for software quality, development costs, and time to market as it doesn't require any change to existing software. However, this technique may have a slight performance impact on throughput and interrupt latency. To optimize throughput and latency, a completely optional API is available to wrap privileged instructions into functionally equivalent TEE calls. The API is provided in the form of a C header file and uniquely comprised of small and efficient inline assembly code – no library or stack overhead required.

TEE Messenger. The TEE runtime provides a self-contained facility for inter-zone secure communications. It allows zones to exchange secure messages – protected bytes streams - on a non-shared memory basis. The TEE API offers two system calls for sending and receiving messages – *ECALL_SEND* and *ECALL_RECV*.

VI. Reference Application And Evaluation

Fig. 3 shows our reference application. The proof of concept controls the movements of a small robotic arm via a local terminal console. It includes also a set of built-in bare-metal commands to test security and separation of the system and to measure performance overhead and interrupt latency. Zone1 connects to a PC terminal via serial port (UART) to demonstrate peripheral mapping, secure user-mode drivers, enforcement of isolation policies, performance statistics, soft-timer, and inter-zone communications. At the same time, Zone2 blinks a LED and interfaces with local buttons to demonstrate secure user-level interrupt handling and secure messaging. Zone3 operates the robotic arm connected via GPIO. Commands are received from Zone1 and the status of the robot reported back via secure messaging.

The multi zone TEE was evaluated on a Microchip SAM E70 Xplained Evaluation Kit. The Microchip SAM E70 is equipped with a Cortex-M7 ATSAME70Q21 processor clocked at

240MHz. In addition to the extensive tests for security, separation, and reliability accessible via zone #1, we measured TCB size and performance overhead.

TCB Size. To minimize the attack surface and to allow for formal verification the multi zone runtime is completely written in assembly and self-contained with zero dependencies on compiler libraries - typical of system level software written in C. The TCB of the system comprises a total amount of (approx.) 2.5KiB. At least one order of magnitude smaller than any system level software documented in publicly available literature. On the basis of the proven correlation between TCB and number of defects, we can safely conclude that the proposed TEE is at least ten times less exposed to vulnerabilities or more simplistically "ten times more secure".

Performance Overhead. To assess the performance overhead we measured zone context switch time. For the system under test configured for four zones, a complete context switch takes 146 clock cycles – or 608ns @240MHz. For a system configured with a preemption time of 10ms, the worst-case performance overhead amounts to 0.006%, which is practically neglectable in any real world application.

REFERENCES

[1] A. Sadeghi, C. Wachsmann and M. Waidner, "Security and privacy challenges in industrial Internet of Things," 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, 2015.

[2] Chris Conlon and Cesare Garlati. "A New Zero-Trust Model for Securing Embedded Systems". In Proceedings of the Embedded World Conference, Nuremberg, Germany, 2019.

[3] Ori Karliner, "FreeRTOS TCP/IP Stack Vulnerabilities Put A Wide Range of Devices at Risk of Compromise: From Smart Homes to Critical Infrastructure Systems." Zimperium, 18 October, 2018.

[4] Lily Hay Newman, "An Operating System Bug Exposes 200 Million Critical Devices." Wired, 29 July, 2019.

[5] Brandon Lewis, "Secure IoT devices from the microcontroller, up." Embedded Computing, 18 July, 2018.

[6] David Cerdeira, Nuno Santos, Pedro Fonseca, Sandro Pinto. " SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems." IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, 2020.

[7] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes". ACM Conference on Computer and Communications Security (CCS), NY, USA, 2019.

[8] Hex Five Security Inc. "MultiZone® Security SDK for Arm Cortex-M". GitHub, 2020. [Online]: https://github.com/hex-five/multizone-sdk-arm

[9] Cesare Garlati and Sandro Pinto. "A Clean Slate Approach to Embedded Linux Security: RISC-V Enclaves". In Proceedings of the Embedded World Conference, Nuremberg, Germany, 2020.