
black hat
ASIA 2020
OCTOBER 1-2, 2020
BRIEFINGS
MAY THE TRUST BE WITH YOU:
EMPOWERING TRUSTZONE-M
WITH MULTIPLE TRUSTED ENVIRONMENTS
#BHASIA @BLACKHATEVENTS

ABOUT US

DANIEL OLIVEIRA (@danieljcoliv)

- PhD Student at University of Minho, Portugal
- Active member of ESRGv3
- Worked in a number of research projects from the automotive industry (Bosch) to embedded security (Hex Five Security, Inc)

SANDRO PINTO (@sandro2pinto)

- Research Scientist & Professor at University of Minho, Portugal
- Active and leading member of ESRGv3
- Working as an External Collaborator in Hex Five Security, Inc

BACKGROUND

- R&D with Arm TrustZone technology since 2013
- We have published 15+ papers/articles about TrustZone
- Highlights:

SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems

David Cerdeira, Centro Algoritmi, Universidade do Minho, david.cerdeira@dei.uminho.pt
Nuno Santos, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, nuno.santos@inesc-id.pt
Pedro Fonseca, Department of Computer Science, Purdue University, pfonseca@purdue.edu
Sandro Pinto, Centro Algoritmi, Universidade do Minho, sandro.pinto@dei.uminho.pt

***Abstract*—Hundreds of millions of mobile devices worldwide rely on Trusted Execution Environments (TEEs) built with Arm TrustZone for the protection of security-critical applications (e.g., DRM) and operating system (OS) components (e.g., Android keystore). TEEs are often assumed to be highly secure; however, over the past years, TEEs have been successfully attacked multiple times, with highly damaging impact across various platforms. Unfortunately, these attacks have been possible by the presence of security flaws in TEE systems. In this paper, we aim to understand which types of vulnerabilities and limitations affect existing TrustZone-assisted TEE systems, what are the main challenges to build them correctly, and what contributions can be borrowed from the research community to overcome them. To this end, we present a security analysis of popular TrustZone-assisted TEE systems (targeting Cortex-A processors) developed by Qualcomm, Trustonic, Huawei, Nvidia, and Linaro. By studying publicly documented exploits and vulnerabilities as well as by reverse engineering the TEE firmware, we identified several critical vulnerabilities across existing systems which makes it legitimate to raise reasonable concerns about the security of commercial TEE implementations.**

***Index Terms*—TEE, TrustZone, Security Vulnerabilities, Arm**

I. Introduction

Trusted Execution Environments (TEE) are a key security mechanism to protect the integrity and confidentiality of applications. By leveraging dedicated hardware, TEEs enable the execution of security-sensitive applications inside protected domains isolated from the platform's operating system (OS). Arm TrustZone [1] has become the de facto hardware technology to implement TEEs in mobile environments and has been employed in industrial control systems [2], servers [3], and low-end devices [4]. In the future, where trillions of TrustZone-enabled IoT devices are expected worldwide [5], TEEs can provide secure environments for data processing at the edge.

TrustZone-assisted TEEs are generally assumed to be more secure than modern OSes due to the hardware-based separation enforced by TrustZone technology and their smaller Trusted Computing Base (TCB), which is several orders of magnitude smaller than standard OSes'. For this reason, TEEs have become widely adopted for securing mobile devices against malware [6–10]. For instance, Android platforms incorporate TrustZone-assisted TEEs to secure application-specific operations involving, e.g., user authentication [11], online banking [12], or DRM [13]. Unfortunately, some of these systems have been exploited over the past years, which casts doubt on the real security guarantees that existing commercial TEEs can effectively provide.

In this paper, we perform a systematic study of publicly disclosed vulnerabilities in commercial TrustZone-assisted TEEs for Arm Cortex-A devices. Despite the existence of multiple security reports affecting such systems, this information tends to be scattered and, in certain cases, unverified, which makes it difficult to obtain a comprehensive understanding of the prevailing vulnerabilities and overall security properties of these systems. To fill this gap, we analyzed 207 TEE bug reports spanning a nearly 5 years, from 2013 until mid-2018, focusing on widely deployed TEE systems developed for Arm-based devices by five major vendors: Qualcomm, Trustonic, Huawei, Nvidia, and Linaro. We examined and categorized numerous vulnerabilities, in particular, some of those that have been leveraged to carry out successful attacks. From our analysis, along with the manual inspection of TEE firmware, we have gained multiple insights about the extent and causes of existing vulnerabilities, and about potential solutions to mitigate them.

One first observation is that TEE systems have a long history of *critical implementation bugs*. Numerous bugs have been (and continue to be) found inside TEE applications – named Trusted Applications (TAs) – and inside the trusted kernel responsible for managing the TEE runtime. Many bugs involve classic input validation errors, such as buffer overflows. As shown by multiple attacks, these bugs can be leveraged to hijack Android's Linux kernel or to entirely compromise the TEE kernel of devices featuring TEEs by Qualcomm [14, 15], Trustonic [16, 17], or Huawei [18].

Second, exploiting vulnerable TAs is facilitated by the *numerous architectural deficiencies* of TrustZone-assisted TEE systems. For instance, the memory protection mechanisms commonly found in modern OSes, e.g., ASLR or page guards, are almost absent or ill-implemented in most analyzed systems. TEE systems also tend to expose a large attack surface, including dangerous TEE kernel system calls that can be invoked by TAs. For example, on Qualcomm's TEE, any TA can map in memory regions of the host OS. As a result, by hijacking a vulnerable TA, e.g., leveraging a buffer overflow, an attacker can easily control Android [15].

Third, *important hardware properties are overlooked* in most TrustZone systems at the architectural and microarchitectural levels, which can compromise the security of the TEE. Some vulnerabilities are caused by unexpected behavior of trusted hardware components due to microarchitectural side-channels (e.g., in caches) [19–23]. Others are caused by components that can be leveraged to exfiltrate sensitive data from TEE-restricted

IEEE
Symposium on Security and Privacy
2020

Demystifying Arm TrustZone: A Comprehensive Survey

SANDRO PINTO, Centro Algoritmi, Universidade do Minho
NUNO SANTOS, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

The world is undergoing an unprecedented technological transformation, evolving into a state where ubiquitous Internet-enabled "things" will be able to generate and share large amounts of security- and privacy-sensitive data. To cope with the security threats that are thus foreseeable, system designers can find in Arm TrustZone hardware technology a most valuable resource. TrustZone is a System-on-Chip and CPU system-wide security solution, available on today's Arm application processors and present in the new generation Arm microcontrollers, which are expected to dominate the market of smart "things." Although this technology has remained relatively underground since its inception in 2004, over the past years, numerous initiatives have significantly advanced the state of the art involving Arm TrustZone. Motivated by this revival of interest, this paper presents an in-depth study of TrustZone technology. We provide a comprehensive survey of relevant work from academia and industry, presenting existing systems into two main areas, namely, Trusted Execution Environments and hardware-assisted virtualization. Furthermore, we analyze the most relevant weaknesses of existing systems and propose new research directions within the realm of tiniest devices and the Internet of Things, which we believe to have potential to yield high-impact contributions in the future.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Security and privacy → Systems security**; *Security in hardware*; Software and application security;

Additional Key Words and Phrases: TrustZone, security, virtualization, TEE, survey, Arm

ACM Reference format:
Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* 51, 6, Article 130 (January 2019), 36 pages.
https://doi.org/10.1145/3291047

1 INTRODUCTION

Arm TrustZone consists of hardware security extensions introduced into Arm application processors (Cortex-A) in 2004 [1, 63]. More recently, TrustZone has been adapted to cover the new generation of Arm microcontrollers (Cortex-M) [65, 113]. TrustZone follows a System-on-Chip (SoC) and CPU system-wide approach to security. This technology is centered around the concept of protection domains named *secure world* and *normal world*. The software executed by the

This work has been partially supported by COMPETE: POCI-01-0145-FEDER-007043, by COMPETE 2020/Portugal 2020/União Europeia within the project Mobile Security Ticketing (No. 11388), which is presented by Link Consulting Tecnologias de Informação SA, and by FCT—Fundação para a Ciência e Tecnologia—within the Project Scope: UID/CEC/00319/2013, UID/CEC/50021/2013, SFRH/BSAB/135236/2017, PTDC/EEI-SCR/1741/2014 (Abyss).
Authors' addresses: S. Pinto, Centro Algoritmi, Universidade do Minho, Campus de Azurém, Guimarães, 4800-058, Portugal; email: sandro.pinto@dei.uminho.pt; N. Santos, Rua Alves Redol, Lisboa, 1000-029 Lisboa, Portugal; email: nuno.santos@inesc-id.pt.

0360-0300/2019/01-ART130 $15.00
https://doi.org/10.1145/3291047

ACM
Computing Surveys
2019

AGENDA

01 **Introduction**
Current state of IoT security, motivation, and goals

02 **Arm TrustZone-M “101”**
TrustZone technology in a nutshell

03 **TrustZone Pitfalls**
What we have learned so far: the limitations of the dual–world model

04 **uTango**
Empowering TrustZone-M with multiple trusted environments

05 **The Numbers**
Tests, experiments, and results (Arm Musca-B1)

06 **“Live” Demo**
Demo of uTango multi-world IoT stack

07 **Conclusion**
Takeway points

Introduction

Current state of IoT security, motivation, and goals

03 Jan 2019

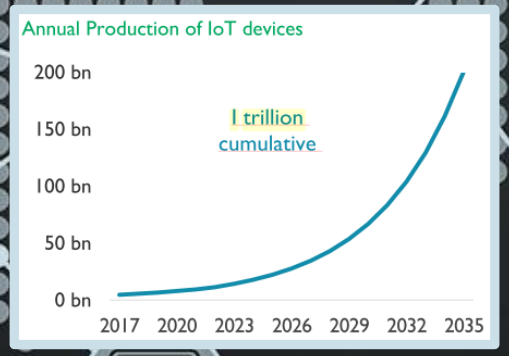
IDC Forecasts Worldwide Spending on the Internet of Things to Reach $745 Billion in 2019, Led by the Manufacturing, Consumer, Transportation, and Utilities Sectors

EGHAM, U.K., August 29, 2019

Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020

THE INTERNET OF THINGS

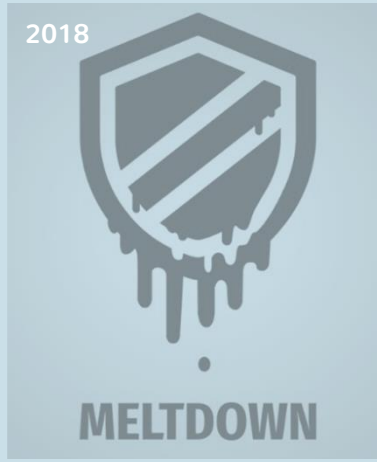
One trillion new IoT devices will be produced by 2035



Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated

Warning: All projections for the Internet of Things are subject to change

We need to address this **cat-and-mouse game** of increasing hacks and patches that **threatens** the IoT



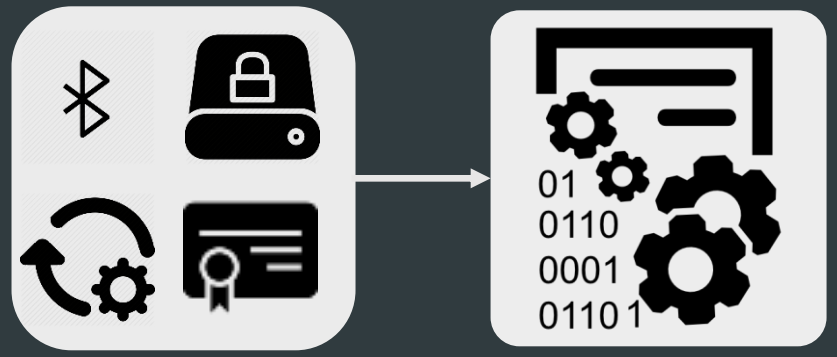
"We are witnessing cyberattacks on **critical** infrastructure, health services systems captured and **held to ransom** and home electronics devices used as **internet gateways** by **hackers**."

arm, in Security Manifesto, 2017.

IOT SECURITY PROBLEM

IoT-based devices requirements

- Connectivity (e.g WiFi, Bluetooh, ZigBee)
- Upgradability (e.g. firmware, features, security)
- Sensitive information (e.g. keys, certificates)



IoT systems are vertical silos!

All code blocks are **combined** into a single memory image. Therefore, if **one block** is **compromised**, the attacker can **most certainly** exploit any other component – there's **no separation** on this traditional model (e.g., FreeRTOS TCP/IP vulnerabilities) .

Isolate! TrustZone to the rescue

TrustZone addresses this problem by leveraging a system-wide architecture which is **split into two worlds**: a **secure** and a **non-secure** world. A trust barrier isolates the sensitive code from the rich and untrustworthy software.

But, is TrustZone enough?

KEY MOTIVATION POINTS

01

EXPLORE TRUSTZONE-M

TrustZone-M is a fairly **new** technology, with a **very few** number of products/works and very **little research**.

Which are the **key architectural differences** between the **two siblings**?

02

TRUSTZONE CLASSIC MODEL IS FALLING SHORT

We observed how the **dual-world model** is becoming **impractical** and **limited** to address the continuous growth of the attack surface in small IoT devices.

Which are the **main pitfalls** of TrustZone? And how can we **mitigate** them?

03

EMPOWER TZ WITH MULTIPLE TRUSTED WORLDS

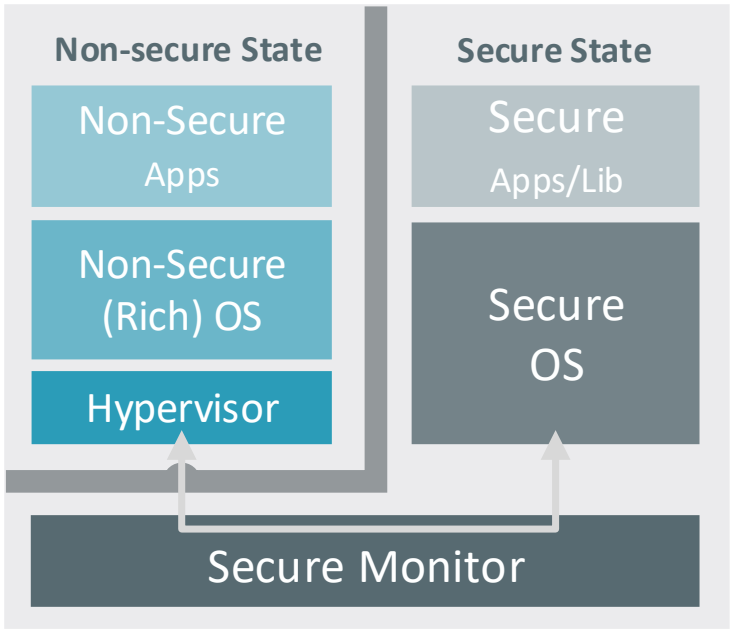
Apply a new **zero-trust model** to all pieces of code and isolate them in completely **isolated domains** (i.e. 'containers'), by following a complete **horizontal multi-domain** security schema.

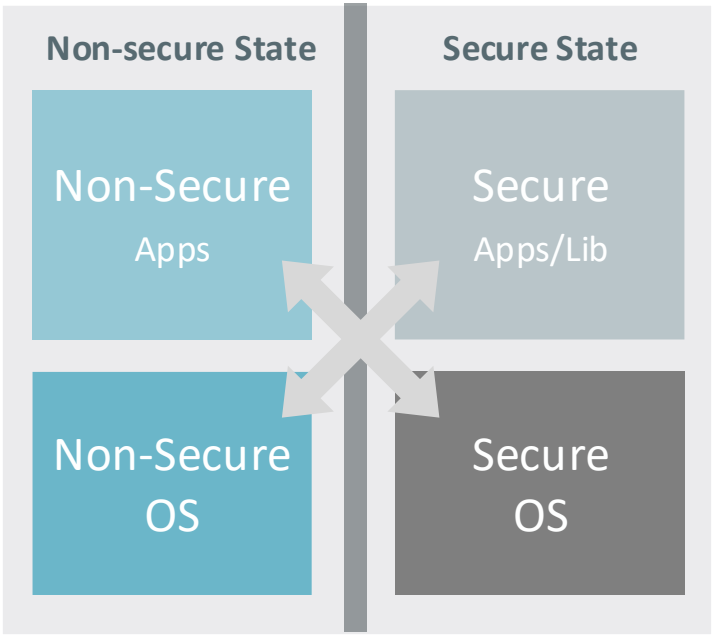
How to leverage the **TrustZone-M hardware features** to implement a **multi-world** approach**?**

Arm TrustZone-M “101”

TrustZone technology in a nutshell

TRUSTZONE VS TRUSTZONE-M





TrustZone (for Cortex-A)

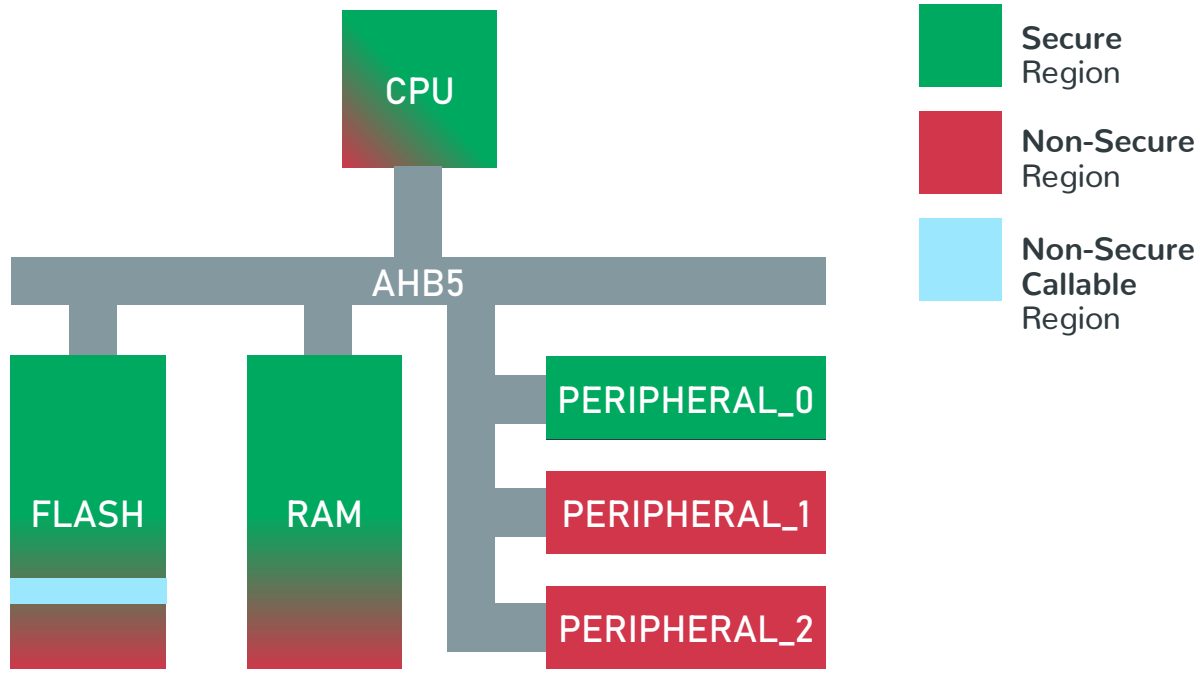
- Introduced in 2004
- Execution state is determined by the NS bit
- Monitor mode / SMC instruction
- Memory and devices: TZASC and TZPC
- Interrupts: GIC - IRQs vs FIQs

TrustZone-M (for Cortex-M)

- Introduced in 2018
- Execution state is memory mapped based
- No Monitor mode / SG instruction
- Memory and devices: SAU and IDAU
- Interrupts: NVIC – secure and non-secure IRQs

1 2 3 4

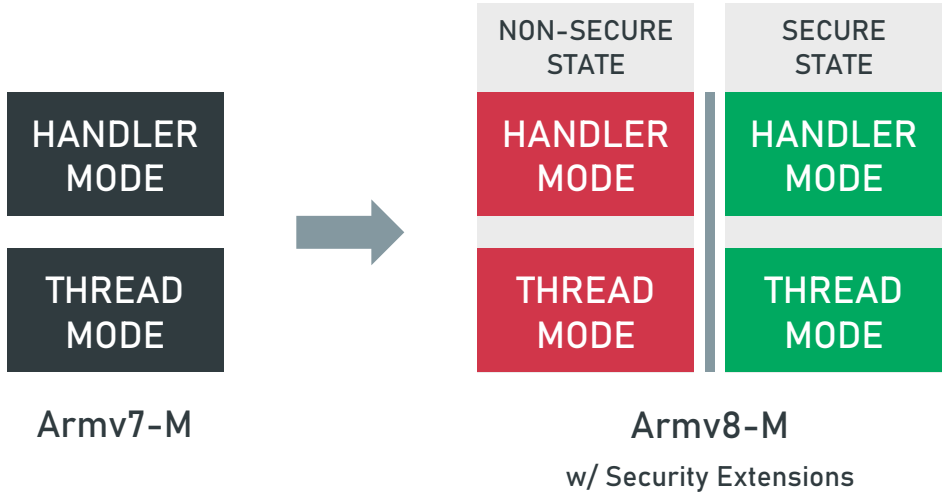
EXECUTION STATE IS MEMORY MAP BASED



On Cortex-M MCUs, SoC resources (FLASH, RAM, peripherals) are mapped to a **memory address**

- The 4GB memory space is **partitioned** into:
  - **Secure** (software can access secure or non-secure resources)
  - **Non-Secure** (software can only access non-secure resources)
  - **Non-Secure Callable** (memory area only used for state transition)
- The **processor state** is **dependent** on the memory space definition: when the processor is running code in a **Secure** region it is in **secure state**, otherwise it is in **Non-Secure state**

SECURITY STATES
TRANSITION/SG INST

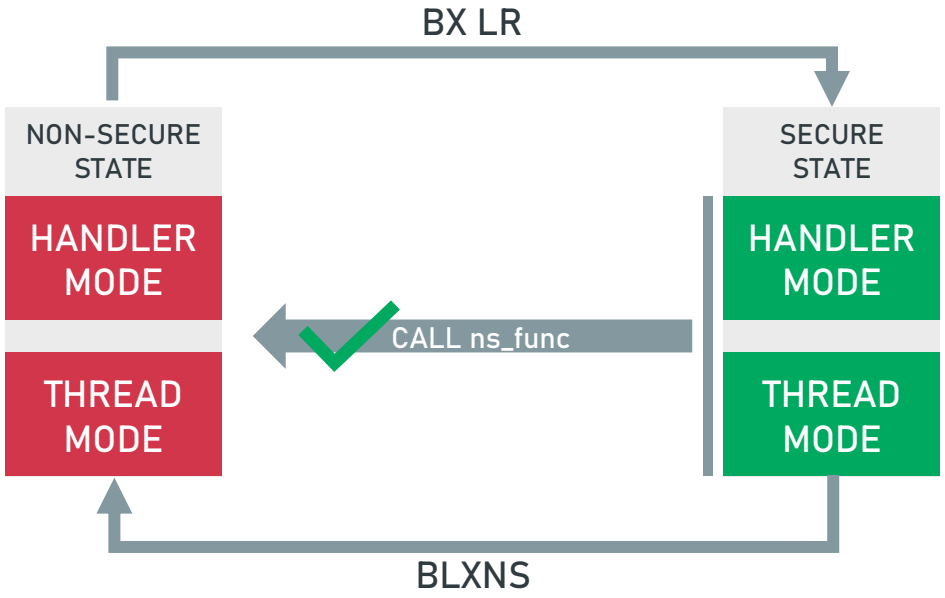


Cortex-M MCUs have two operation modes: **Handler** (for exception handling) and **Thread** (for normal application code – priv/unpriv)

- TrustZone-M **replicates** operation modes for the secure and non-secure states - the security states are **orthogonal** to the existing processor modes

1 2 3 4

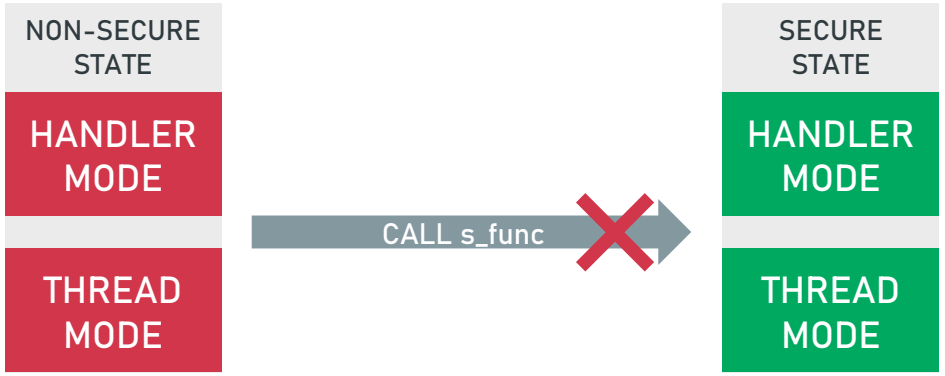
SECURITY STATES
TRANSITION/SG INST



- **Secure** software can call non-secure software using the **BLXNS <Ri>** inst.
  - Return address is pushed onto the **secure stack**
  - **Link Register** is set to a special value called **FNC_RETURN**
  - Performing a **branch** to this **LR** automatically triggers the unstacking of the **true return address** from the secure stack and a return transition to the **S** world

1 2 3 4

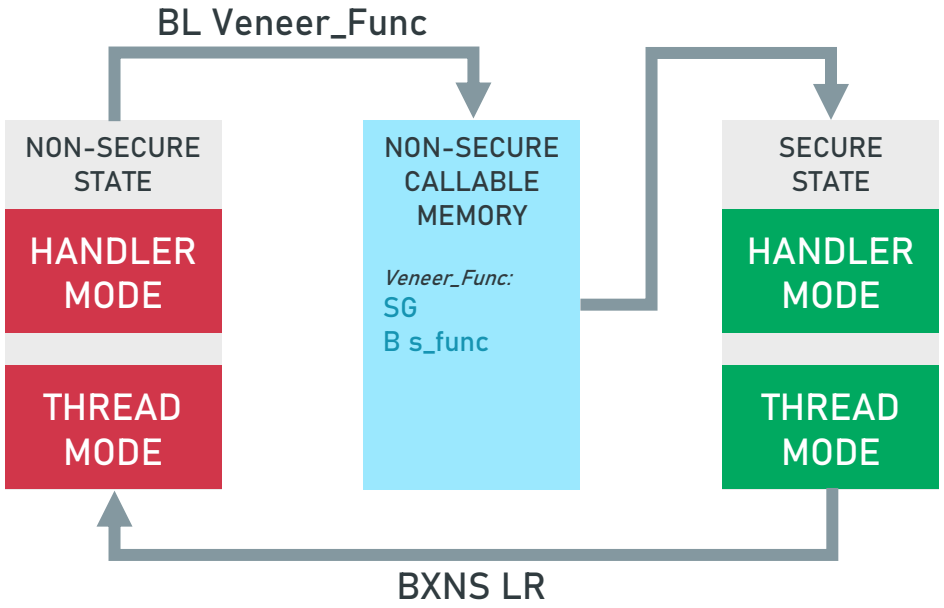
SECURITY STATES
TRANSITION/SG INST



- Non-Secure software **cannot directly** call the secure software
  - A **SecureFault** (or a HardFault) **exception** will be triggered
  - Prevents direct calls into protected code

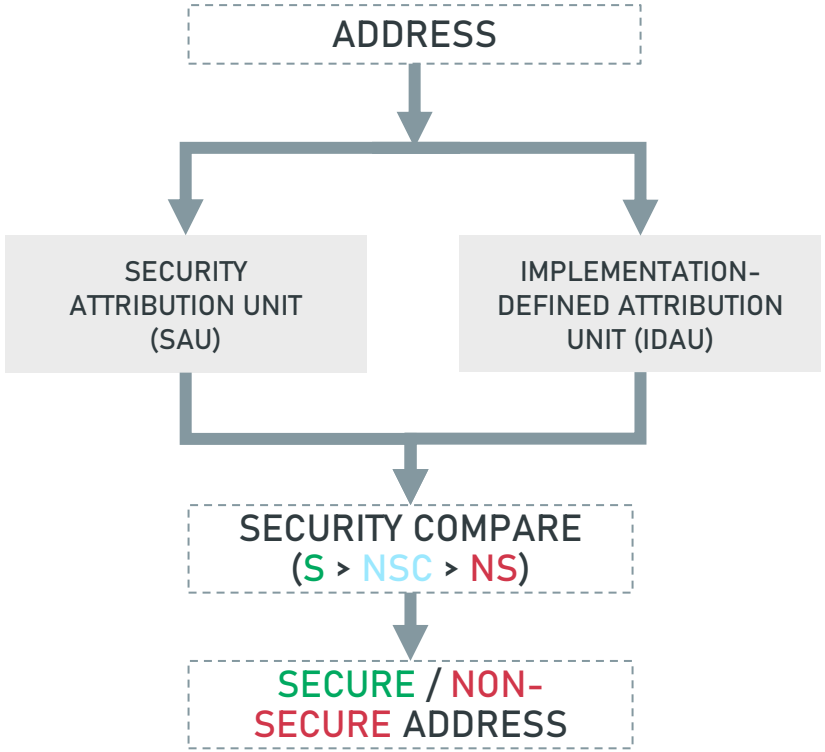
1 2 3 4

SECURITY STATES
TRANSITION/SG INST



- Non-Secure software can **only** call secure software through specific entry points located in a Non-Secure Callable (NSC) memory region
  - The first instruction in the API must be an **SG** (secure gateway)
  - The **SG** instruction must be in an NSC region
  - The last instruction must be a **branch** to the secure function
  - After completion, the secure function returns to the non-secure state by using a **BXNS** instruction

MEMORY & DEVICES PARTITIONING

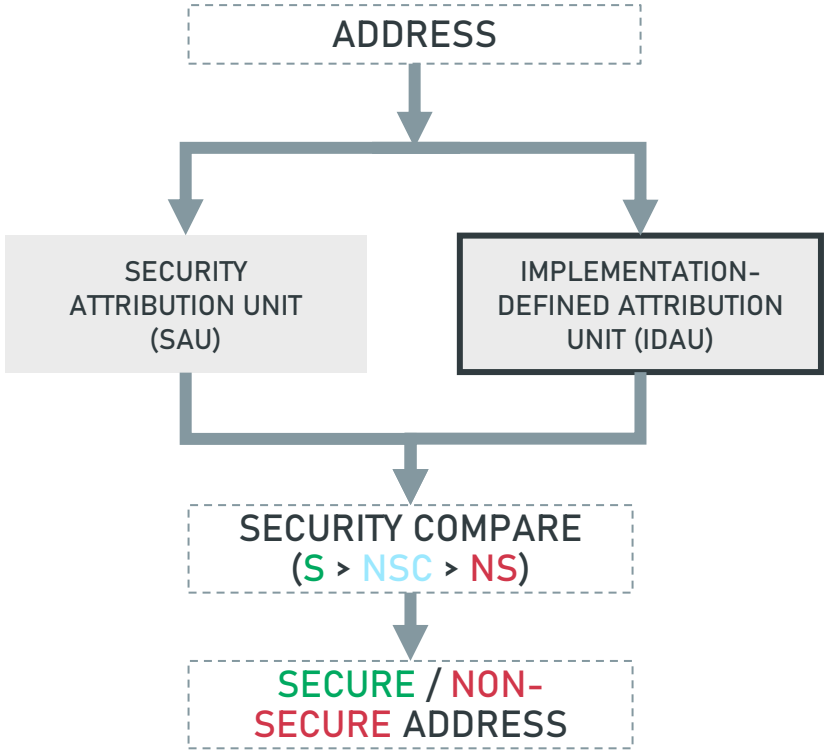


Memory and devices partitioning is defined by address

- The security state of a memory region is selected by a combination of the **IDAU** and **SAU** attribution units
  - The result is the **logical OR operation** between the **security configuration** of both memory controllers

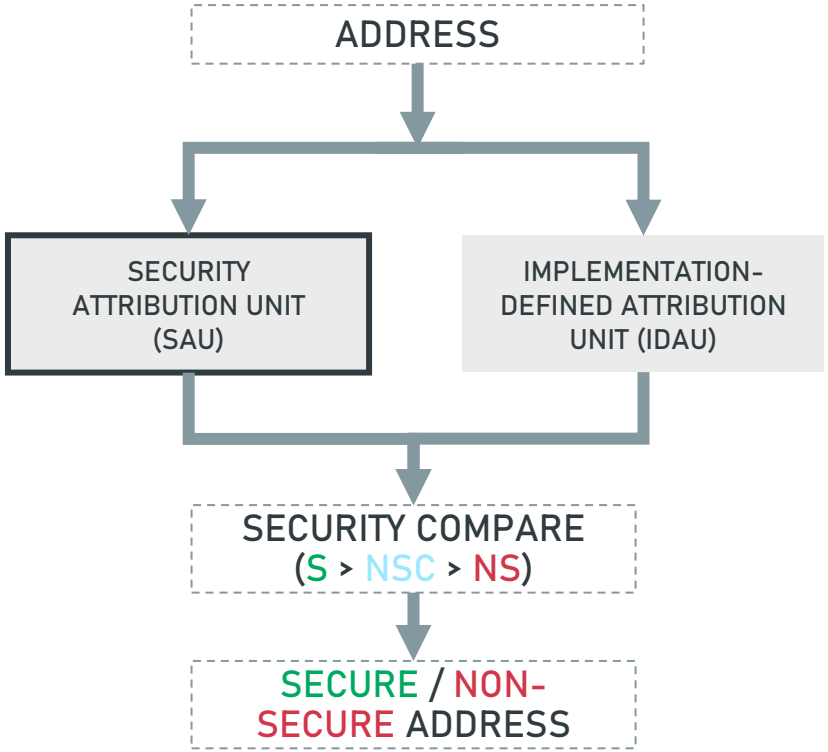
1 2 3 4

MEMORY & DEVICES
PARTITIONING



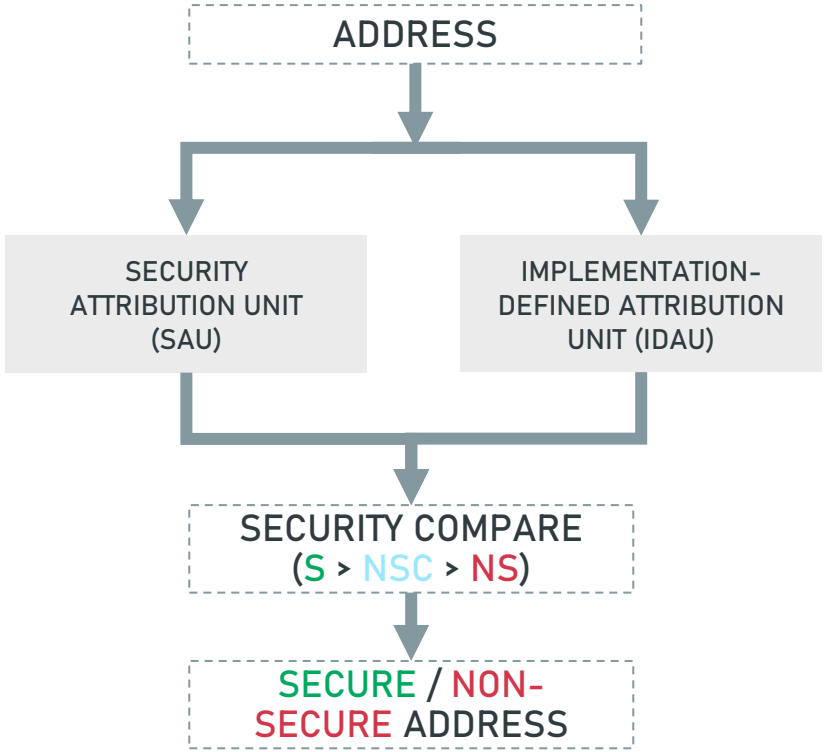
- **IDAU**
  - Provides **static** address **partitioning**
  - **Non-Programmable**
  - **Customized** by the **silicon vendor** to define a fixed memory map
  - Supports up to 256 regions

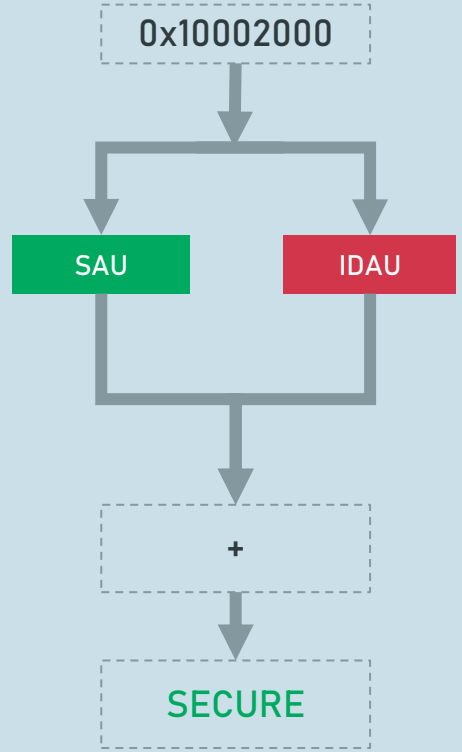
MEMORY & DEVICES
PARTITIONING



- **SAU**
  - Provides **dynamic** address **partitioning**
  - **Programmable** in secure state
  - Always available on all chips with TrustZone support (even if the number of regions is configured to zero)
  - The **number of regions** is defined by the chip designer (typically 8 regions)

MEMORY & DEVICES
PARTITIONING

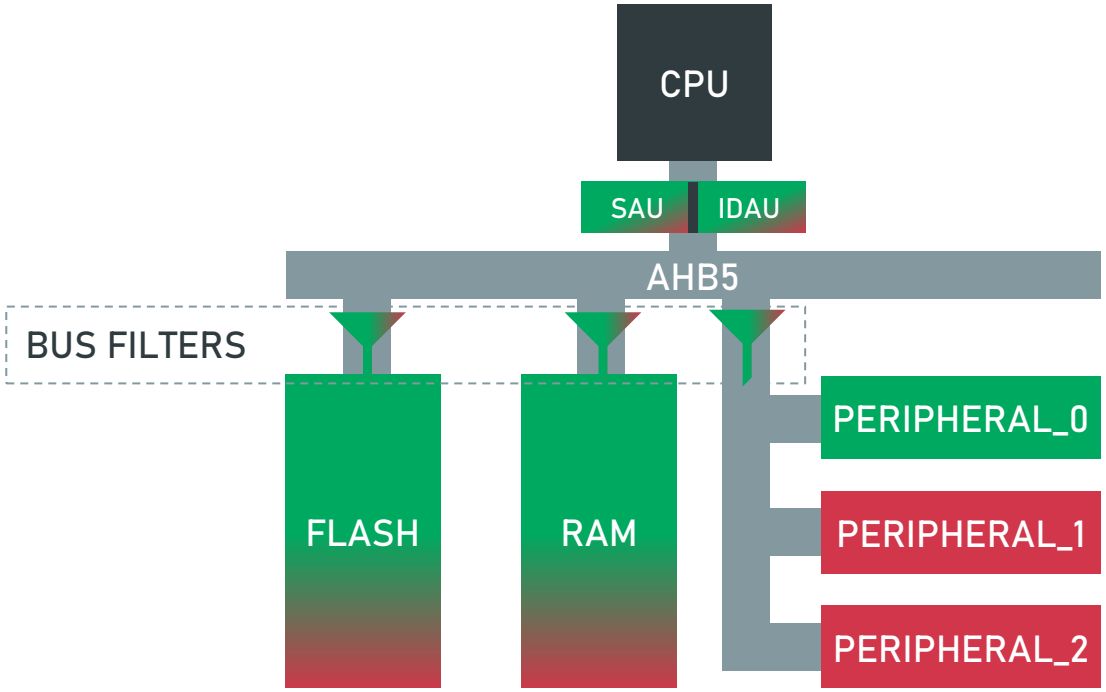




The result is the **logical OR operation** between the **security configuration** of SAU and IDAU

SAU	IDAU	Security State
Non-Secure	Non-Secure	Non-Secure
Secure	Non-Secure	Secure
Non-Secure	Secure	Secure
Secure	Secure	Secure
NSC	Secure	Secure
NSC	Non-Secure	NSC

MEMORY & DEVICES PARTITIONING

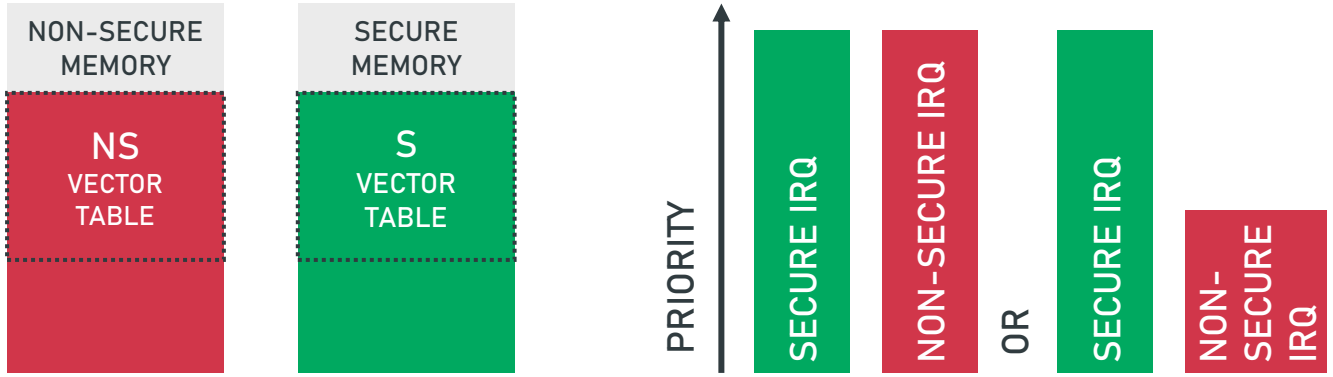


The Security state is extended to other SoC resources through "gate-keepers" (bus filters)

- **SAU** and **IDAU** will affect the output from the **CPU to the bus**, whereas **bus filters** affect the accesses from other **bus masters to the resources**

- e.g. Arm's SSE-200 subsystem specification provides two components that configure **system-wide access rights** to **memory** and **peripherals**:
  - MPC: Memory Protection Controller
  - PPC: Peripheral Protection Controller

INTERRUPT HANDLING

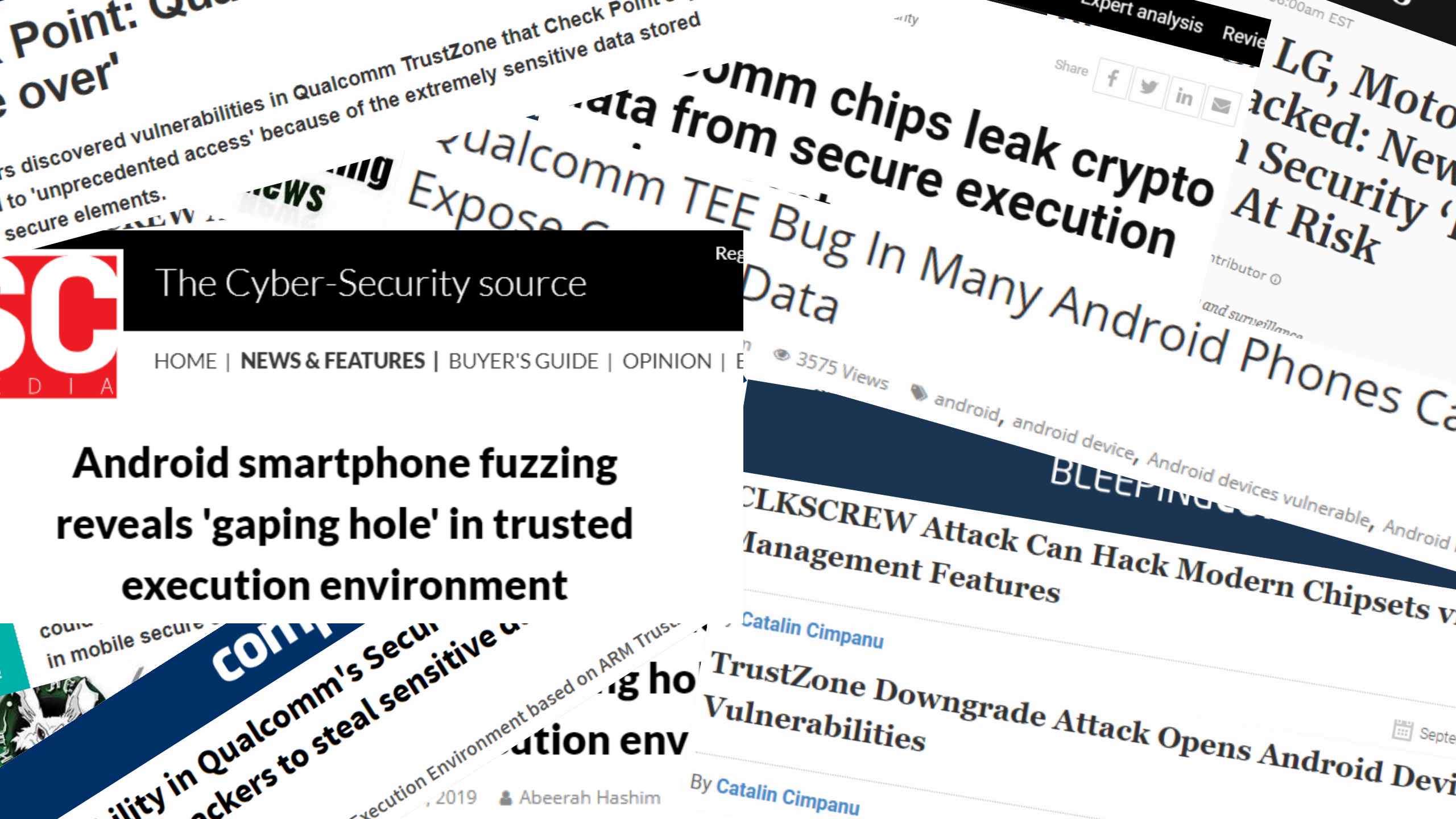


Interrupt management is similar to ARMv7-M, where a subset of the core registers is pushed automatically into the stack

- Supports two separated exception vector tables (**Secure** and **Non-Secure**)
- Each interrupt can be assigned to the **secure** or **non-secure** state through a new NVIC register (ITNS – Interrupt Target Non-Secure)
- **S** or **NS** interrupts can share the **same priority levels**, or **secure** interrupts can have higher priority – i.e. DoS (bit PRIS on AIRCR register)

TrustZone Pitfalls

What we have learned so far: the limitations of the dual–world model


Point: Qua
e over'
rs discovered vulnerabilities in Qualcomm TrustZone that Check Point
to 'unprecedented access' because of the extremely sensitive data stored
secure elements.
ews
Qualcomm TEE Bug In Many Android Phones Ca
Expose
Data
3575 Views
android, android device, Android devices vulnerable, Android
BLEEPING
omm chips leak crypto
ata from secure execution
Expert analysis
Revie
Share
LG, Moto
acked: New
Security '
At Risk
tributor
and surveillance
The Cyber-Security source
HOME | NEWS & FEATURES | BUYER'S GUIDE | OPINION |
Android smartphone fuzzing reveals 'gaping hole' in trusted execution environment
CLKSCREW Attack Can Hack Modern Chipsets v
Management Features
Catalin Cimpanu
TrustZone Downgrade Attack Opens Android Devi
Vulnerabilities
By Catalin Cimpanu
Septe
in mobile secure
com
ility in Qualcomm's Secu
ckers to steal sensitive
g ho
ution env
Execution Environment based on ARM Trust
2019
Abeerah Hashim

WHAT HAVE WE LEARNED SO FAR

SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems

David Cerdeira, Centro Algoritmi, Universidade do Minho, david.cerdeira@dei.uminho.pt
Nuno Santos, INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, nuno.santos@inesc-id.pt
Pedro Fonseca, Department of Computer Science, Purdue University, pfonseca@purdue.edu
Sandro Pinto, Centro Algoritmi, Universidade do Minho, sandro.pinto@dei.uminho.pt

***Abstract*—Hundreds of millions of mobile devices worldwide rely on Trusted Execution Environments (TEEs) built with Arm TrustZone for the protection of security-critical applications (e.g., DRM) and operating system (OS) components (e.g., Android keystore). TEEs are often assumed to be highly secure; however, over the past years, TEEs have been successfully attacked multiple times, with highly damaging impact across various platforms. Unfortunately, these attacks have been possible by the presence of security flaws in TEE systems. In this paper, we aim to understand which types of vulnerabilities and limitations affect existing TrustZone-assisted TEE systems, what are the main challenges to build them correctly, and what contributions can be borrowed from the research community to overcome them. To this end, we present a security analysis of popular TrustZone-assisted TEE systems (targeting Cortex-A processors) developed by Qualcomm, Trustonic, Huawei, Nvidia, and Linaro. By studying publicly documented exploits and vulnerabilities as well as by reverse engineering the TEE firmware, we identified several critical vulnerabilities across existing systems which makes it legitimate to raise reasonable concerns about the security of commercial TEE implementations.**

***Index Terms*—TEE, TrustZone, Security Vulnerabilities, Arm**

I. Introduction

Trusted Execution Environments (TEE) are a key security mechanism to protect the integrity and confidentiality of applications. By leveraging dedicated hardware, TEEs enable the execution of security-sensitive applications inside protected domains isolated from the platform's operating system (OS). Arm TrustZone [1] has become the de facto hardware technology to implement TEEs in mobile environments and has been employed in industrial control systems [2], servers [3], and low-end devices [4]. In the future, where trillions of TrustZone-enabled IoT devices are expected worldwide [5], TEEs can provide secure environments for data processing at the edge.

TrustZone-assisted TEEs are generally assumed to be more secure than modern OSes due to the hardware-based separation enforced by TrustZone technology and their smaller Trusted Computing Base (TCB), which is several orders of magnitude smaller than standard OSes'. For this reason, TEEs have become widely adopted for securing mobile devices against malware [6–10]. For instance, Android platforms incorporate TrustZone-assisted TEEs to secure application-specific operations involving, e.g., user authentication [11], online banking [12], or DRM [13]. Unfortunately, some of these systems have been exploited over the past years, which casts doubt on the real security guarantees that existing commercial TEEs can effectively provide.

In this paper, we perform a systematic study of publicly disclosed vulnerabilities in commercial TrustZone-assisted TEEs for Arm Cortex-A devices. Despite the existence of multiple security reports affecting such systems, this information tends to be scattered and, in certain cases, unverified, which makes it difficult to obtain a comprehensive understanding of the prevailing vulnerabilities and overall security properties of these systems. To fill this gap, we analyzed 207 TEE bug reports spanning a nearly 5 years, from 2013 until mid-2018, focusing on widely deployed TEE systems developed for Arm-based devices by five major vendors: Qualcomm, Trustonic, Huawei, Nvidia, and Linaro. We examined and categorized numerous vulnerabilities, in particular, some of those that have been leveraged to carry out successful attacks. From our analysis, along with the manual inspection of TEE firmware, we have gained multiple insights about the extent and causes of existing vulnerabilities, and about potential solutions to mitigate them.

One first observation is that TEE systems have a long history of *critical implementation bugs*. Numerous bugs have been (and continue to be) found inside TEE applications – named Trusted Applications (TAs) – and inside the trusted kernel responsible for managing the TEE runtime. Many bugs involve classic input validation errors, such as buffer overflows. As shown by multiple attacks, these bugs can be leveraged to hijack Android's Linux kernel or to entirely compromise the TEE kernel of devices featuring TEEs by Qualcomm [14, 15], Trustonic [16, 17], or Huawei [18].

Second, exploiting vulnerable TAs is facilitated by the *numerous architectural deficiencies* of TrustZone-assisted TEE systems. For instance, the memory protection mechanisms commonly found in modern OSes, e.g., ASLR or page guards, are almost absent or ill-implemented in most analyzed systems. TEE systems also tend to expose a large attack surface, including dangerous TEE kernel system calls that can be invoked by TAs. For example, on Qualcomm's TEE, any TA can map in memory regions of the host OS. As a result, by hijacking a vulnerable TA, e.g., leveraging a buffer overflow, an attacker can easily control Android [15].

Third, *important hardware properties are overlooked* in most TrustZone systems at the architectural and microarchitectural levels, which can compromise the security of the TEE. Some vulnerabilities are caused by unexpected behavior of trusted hardware components due to microarchitectural side-channels (e.g., in caches) [19–23]. Others are caused by components that can be leveraged to exfiltrate sensitive data from TEE-restricted

Main Contributions:

Extensive s

- More than
- Reverse e
- All major T

Classificat

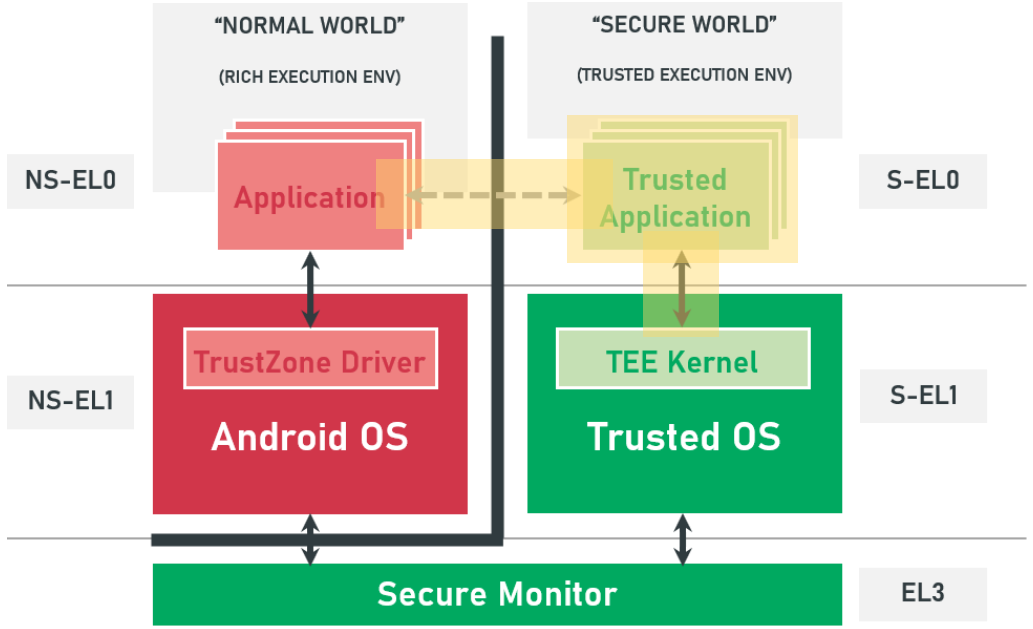
- Architectu
- Implemen
- Hardware

System	Critical	Severe	Medium	Low	Total
Qualcomm TEE	52	19	12	9	92
Trustonic TEE	1	-	0	4	5
Huawei TEE	-	2	-	1	3
Nvidia TEE	-	5	1	4	10
Linaro TEE	-	-	2	1	3
Other	-	1	7	3	11
TEE Total	53	27	22	22	124
FreeRTOS	-	-	5	8	13
VxWorks	2	2	5	1	10
Linux	242	254	393	758	1647

Architetural Issues
(TEE attack surface)

1 **Wide interfaces in TEE componentes**

2 **Excessively large TEE TCBs**



These wide interfaces are presented in **TAs**, the **Trusted-OS**, and even on **device drivers**

- The **Widevine TA** (DRM tool) implements 70 different commands (many of them manipulate complex media data structures)
- The **Qualcomm's TEE** kernel provides to TAs 69 syscalls (e.g. mapping normal world memory)
- The **Trustonic TEE** shares the fingerprint device driver with almost all TAs, which might compromise user authentication

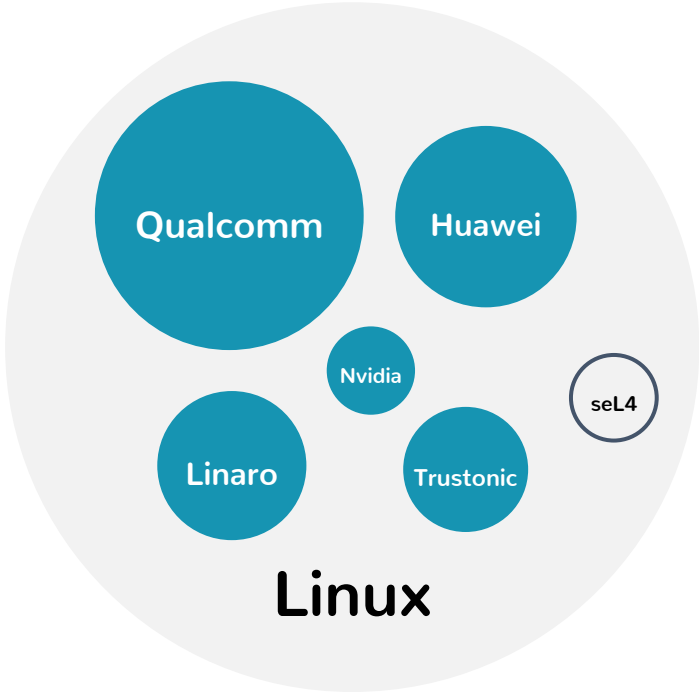
Wide interfaces in TEE components promote designs more **prone** to **vulnerabilities** *

*Jo Van Bulck et al. “A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes”, ACM CCS, 2019.

Architetural Issues
(TEE attack surface)

1 Wide interfaces in TEE componentes

2 Excessively large TEE TCBs



System	Core (bin / src)	TAs
Qualcomm TEE (Google Pixel XL)	1.61MB / –	2.71MB
Trustonic TEE (Samsung S7)	350KB / –	5,02MB
Huawei TEE (Huawei P8 Lite)	744KB /–	479KB
Nvidia TEE (Nvidia Tegra)	97KB / 123Kloc	80KB
Linaro TEE (Hikey960)	365KB /210Kloc	-
Linux (4.14.rc7)	19MB / 15Mloc	-
seL4 (kernel)	166.5KB / 19Kloc	-

We have measured **binary sizes** and the number of **lines of code** of **5 commercial TEEs**

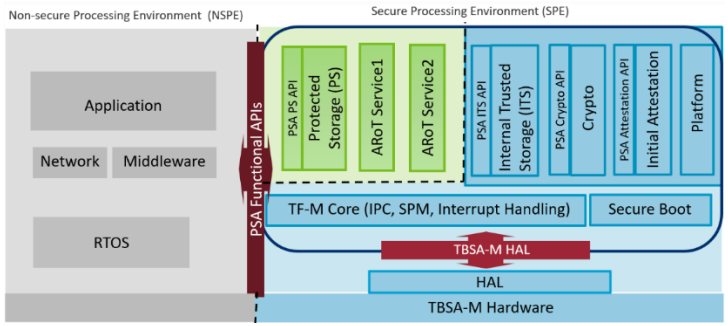
- **TEEs** are fairly **bigger** than similar software systems, such as the **seL4** (security-driven microkernel)
- **TAs** can even **increase the TCB** with megabytes of code that need **also** to be **trusted** alongside the TEE

TEEs are too **big** and too **complex** to be secure!

… AND WHAT ABOUT TRUSTZONE-M?

Example 1

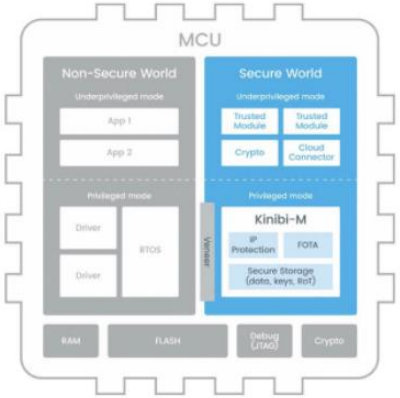
Trusted Firmware-M



Open governance project, that "provides a **reference implementation** of **secure** world **software** for (…) Armv8-M."

Example 2

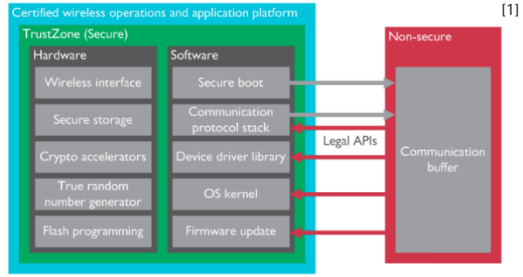
Trustonic Kinibi-M

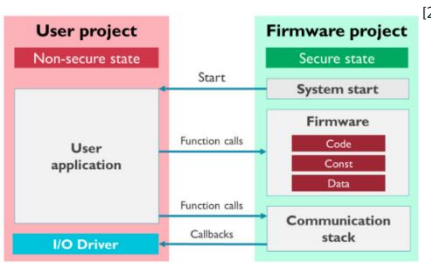


Commercial "**secure TEE** operating system", where "the platform is **adapted** from Trustonic's existing **Kinibi technology** that already secures more than 1.7 billion **smartphones** and **tablets**."

Other Examples

Across Arm Documentation





[1] Arm, "TrustZone technology for the ARMv8-M architecture", 2017.

[2] Keil, "Using TrustZone on ARMv8-M", 2016.

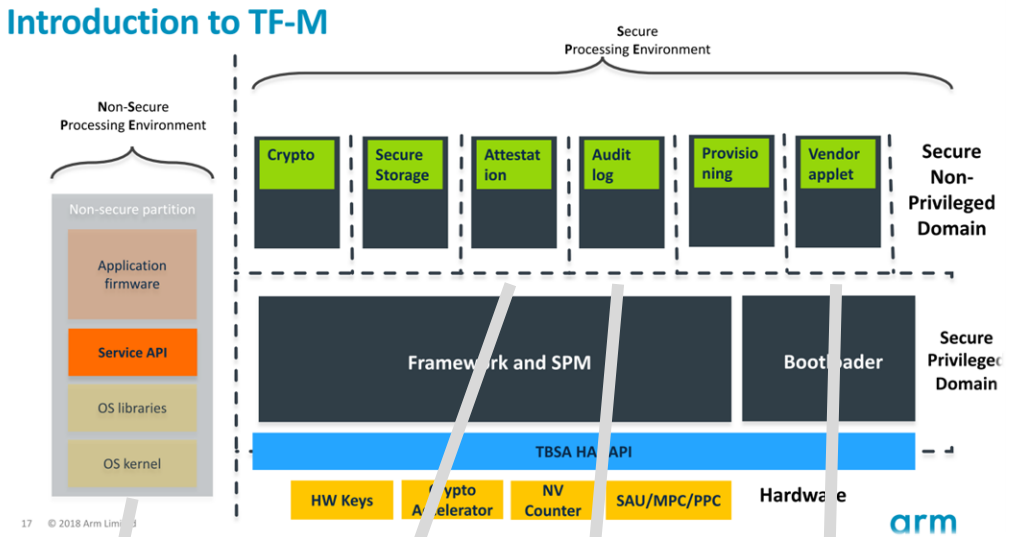
We by no means intend to diminish the value of these solutions…

… but with what we have seen so far, it is very likely that the same problems will happen again in the near future.

uTango

Empowering TrustZone-M with multiple trusted environments

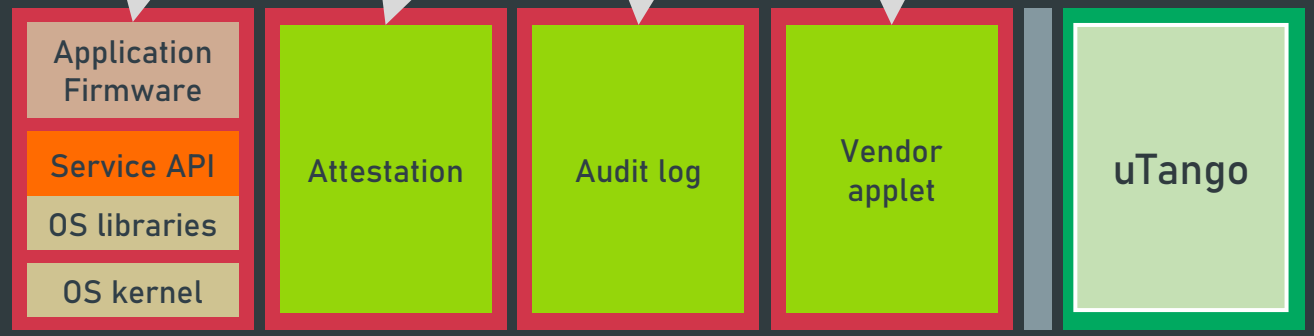
is there a better solution?



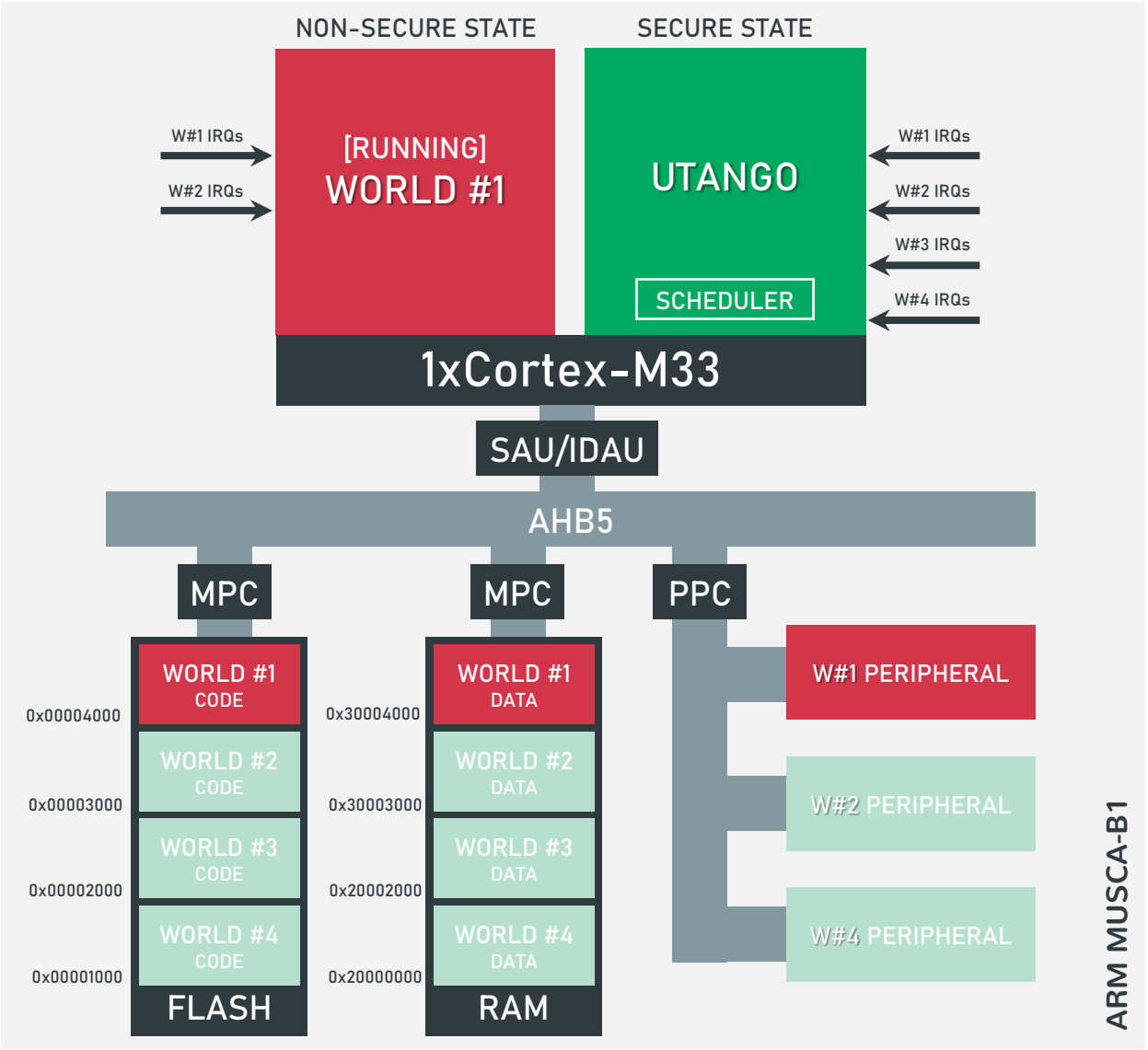
expands the **LIMITED**
TrustZone-M dual-world model

applies a **ZERO-TRUST**
model to all pieces of code

implements a **MULTI-WORLD**
security schema with equally-secure worlds



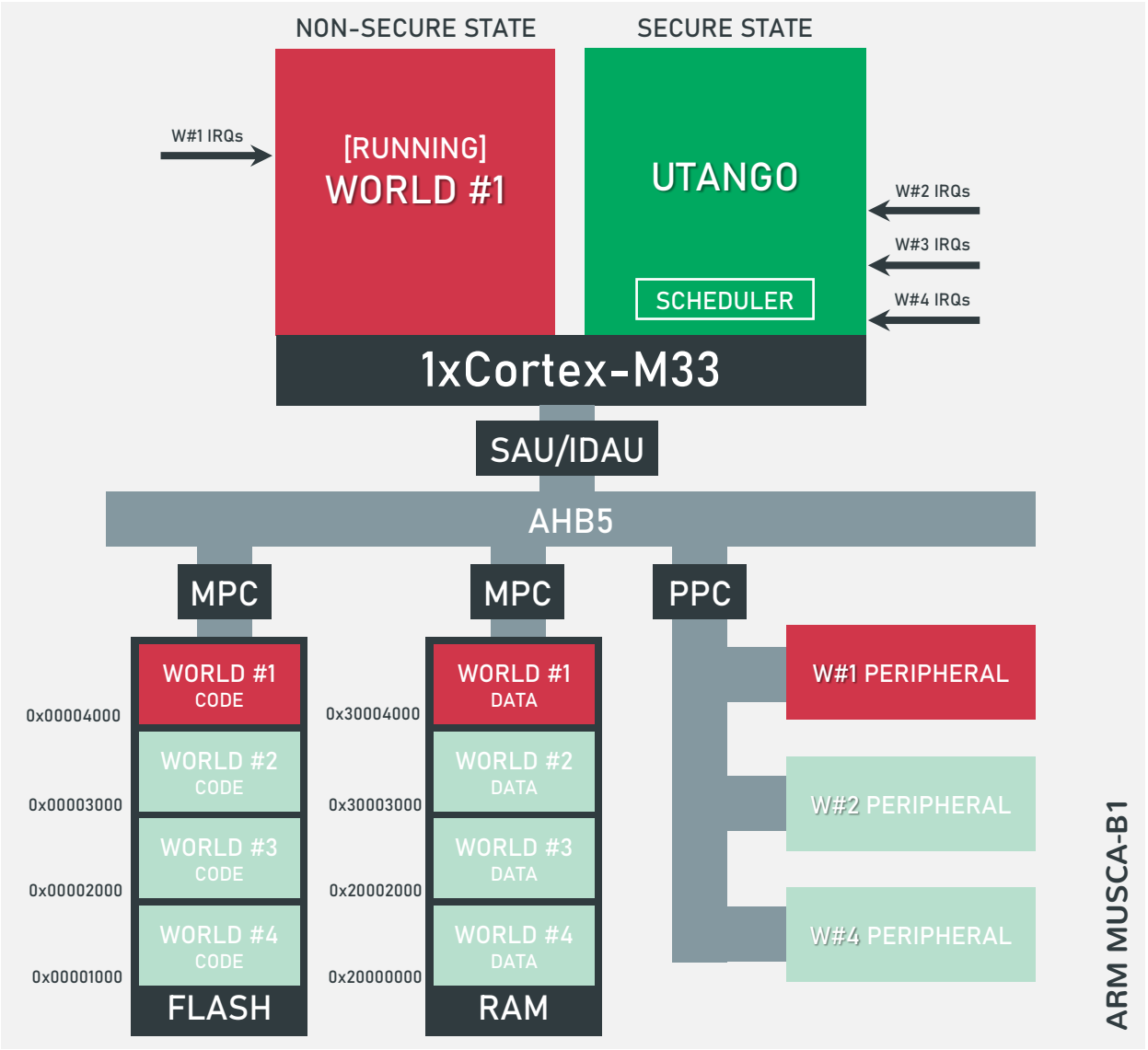
ARCHITECTURE & TOP-10 HIGH-LEVEL DETAILS



You can view **uTango** as a **TEE µkernel** that provides the mechanisms that allow the consolidation of **multiple, equally secure, trusted environments** on a TrustZone-M-enabled platform

1. **uTango** sits on the Secure side and, therefore, is the **first** software component to run, working as the secure bootloader of the system.
2. **Worlds** are guaranteed to run in a **round-robin fashion**, supported by the uTango **scheduler**
3. The extended multiple **trusted environments** (or worlds) run in the Non-Secure side, and are preserved in the secure side when suspended
4. **SAU** is **reconfigured** at each scheduling point, to guarantee the **isolation** of each world and its resources (running or suspended)
5. **Interrupts** security assignment and **state** (enable/disable) is reconfigured at each scheduling point, depending on the world resumed or suspended

ARCHITECTURE & TOP-10 HIGH-LEVEL DETAILS



You can view **uTango** as a **TEE µkernel** that provides the mechanisms that allow the consolidation of **multiple, equally secure, trusted environments** on a TrustZone-M-enabled platform

6	The **execution state** of each world is **saved** or **restore** during the context switching into/from a **control block structure**
7	uTango **scheduler** is supported by the **architectural system timer** (SysTick), which is **banked** between states
8	**MPC** and **PPC** bus filter controllers are configured in **boot time** to match the worlds memory space.
9	uTango is **statically** configured (specify which resources will be used by each world, memory area, interrupts assigned, vector table)
10	uTango is supporting **Arm Musca-B1** and other commercial platforms are in the roadmap.

UTANGO: A "TOY" USE-CASE

WORLD #1

Bare-metal application featuring a serial terminal

Memory Regions:

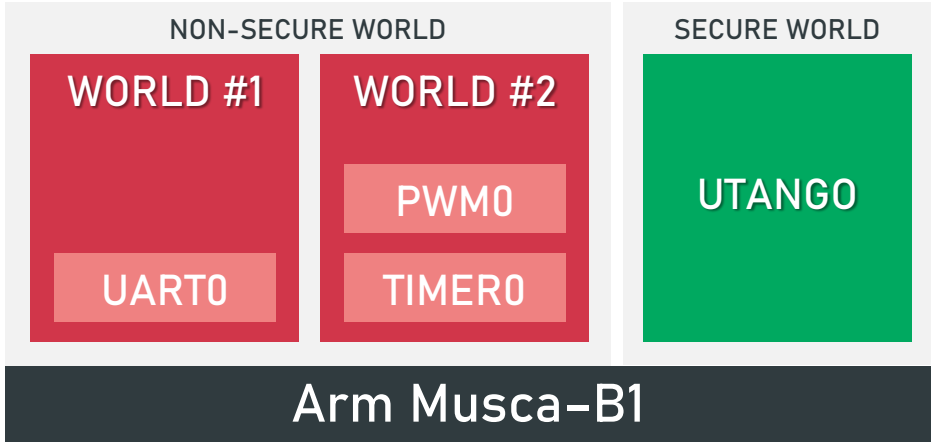
- Code: FLASH
- Data: Internal SRAM1
- Devices: UART0

WORLD #2

Bare-metal application blinking a LED at each timer tick interrupt (1ms)

Memory Regions:

- Code: FLASH
- Data: Internal SRAM2
- Devices: PWM0, TIMER0



UTANGO

Configurations:

- N worlds: 2
- Tick: 10 ms

Memory Regions:

- Code: Internal SRAM0 (TCM)
- Data: Internal SRAM0 (TCM)

BOOT SEQUENCE

CPU CONFIGURATION

- Enable **all** fault handlers (UsageFault, MemFault, BusFault, SecureFault)
- **Disable** system **reset** from non-secure state
- **Prioritize** secure exceptions
- **Disable** CPU deep sleep on non-secure side

SYSTEM TIMER INIT

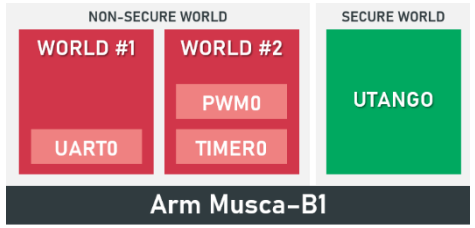
- Set SysTick **reload** value (=10 ms)
- Enable **interrupt** (secure)
- Set clock source

WORLDs CONTROL BLOCK INIT

- Set **each** world control block:
  - **CPU** registers
  - **SCB** register (e.g. VTOR)
  - Memory regions (**SAU** configuration)
  - Interrupts
- Set **exception stack frame**
- Set **1st** world to run **regs** (VTOR, MSP, NVIC)

MEMORY & DEVICES PARTITION

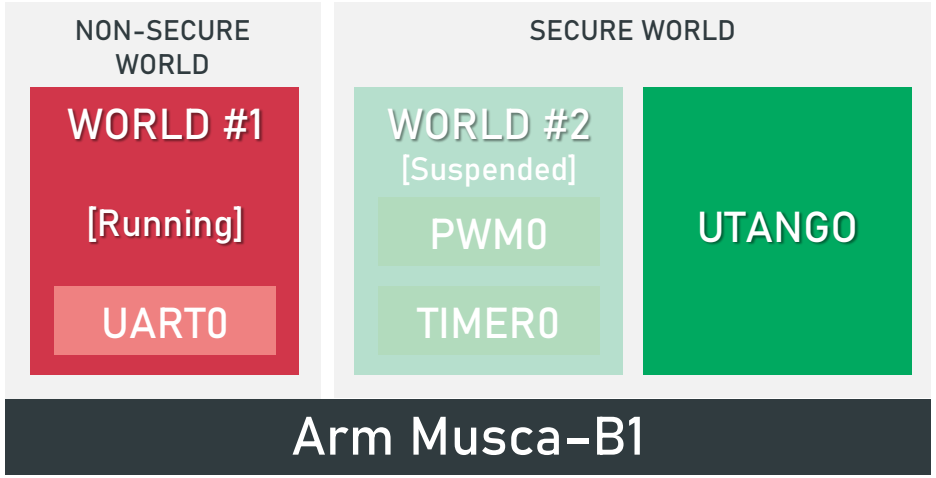
- **Configure** SAU for **1st** world to run
- Enable SAU
- **Configure** bus filters (MPC and PPC)

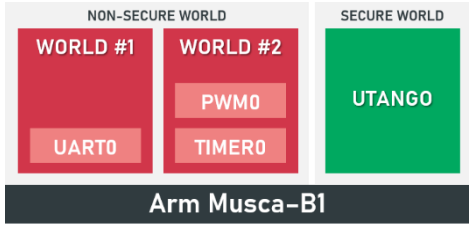


BOOT SEQUENCE

UTANGO START

- Enable **System Timer**
- **KICKOFF!** Jump to 1st world by means of a BLXNS instruction (non-secure function call, that switches from secure to non-secure state)

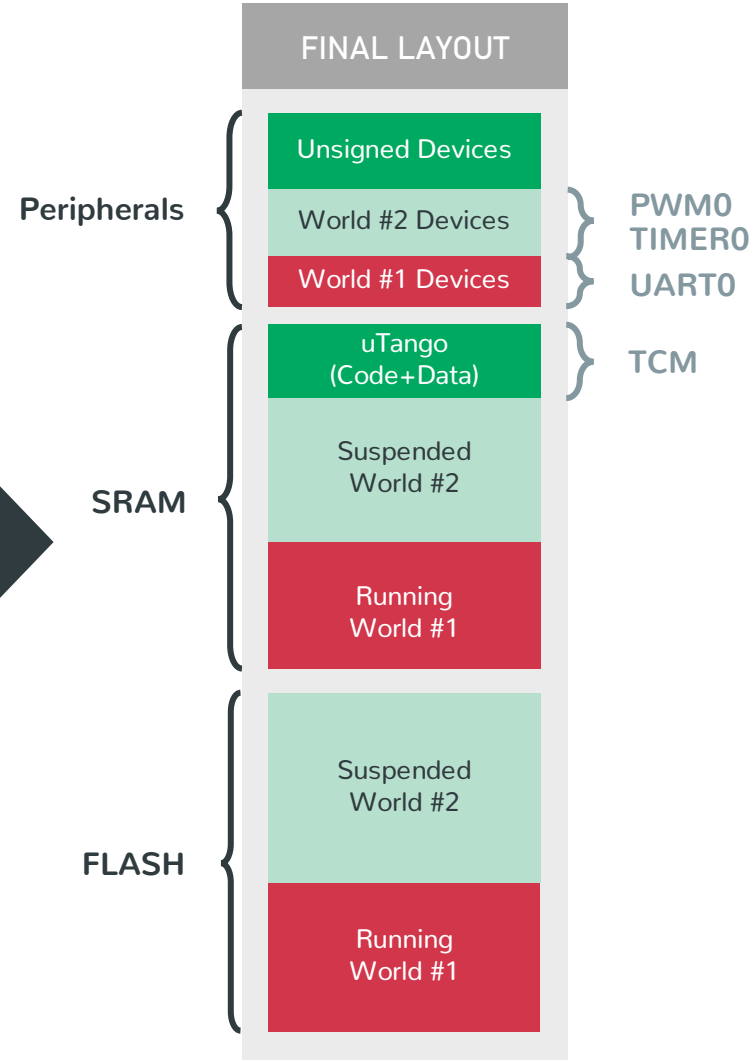


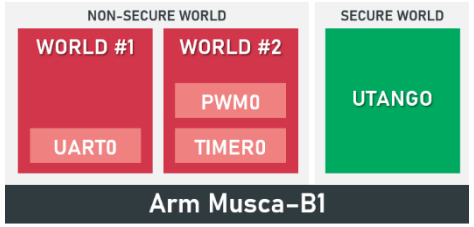


MEMORY & DEVICES PARTITION

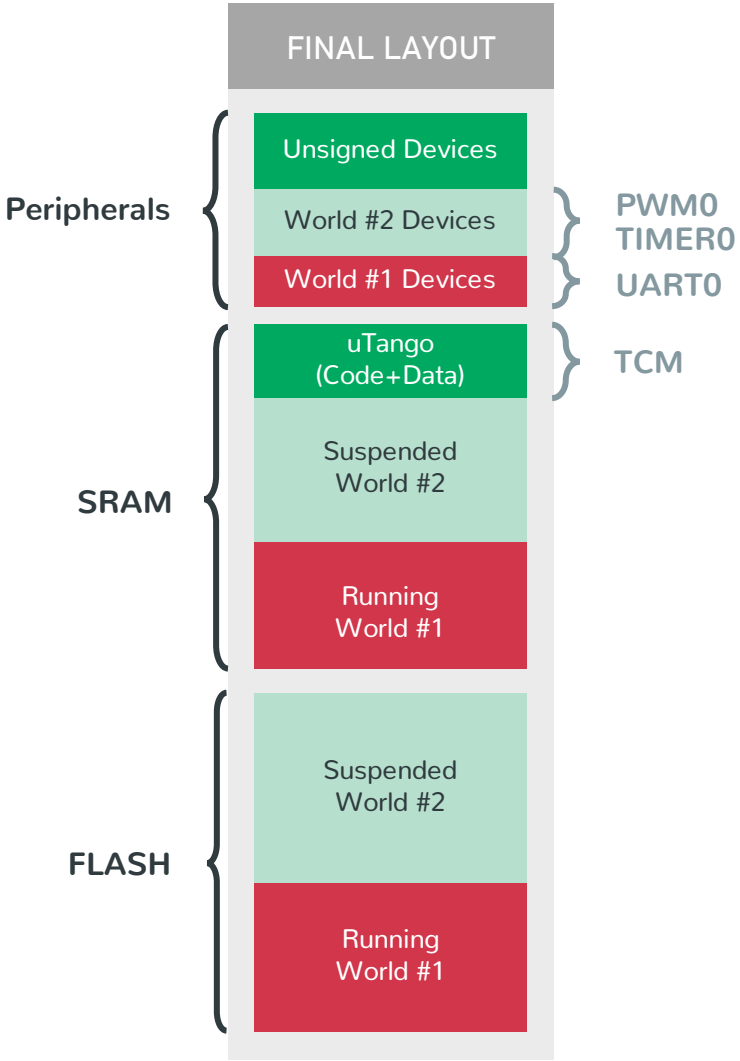
ADDRESS	TYPE	IDAU
0xFFFFFFFF		
0xE0000000	System	Banked
0xA0000000	External Device	Secure / Non-secure
0x80000000	External RAM	Secure / Non-secure
0x60000000		
0x40000000	Peripherals	Secure / Non-secure
0x20000000	SRAM	Secure / Non-secure
0x00000000	Code	Secure / Non-Secure

After **uTango boot** and configuring the **SAU**





MEMORY & DEVICES PARTITION



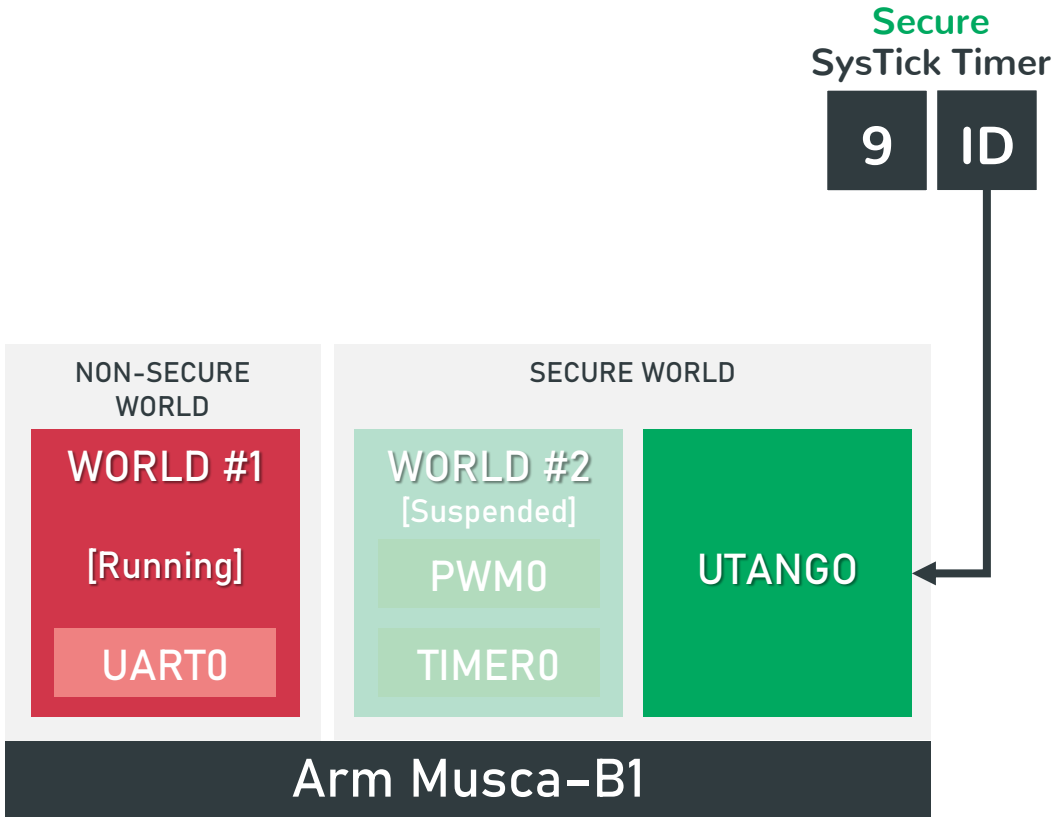
uTango sets up the **memory space** and creates **isolation boundaries** between **worlds**.

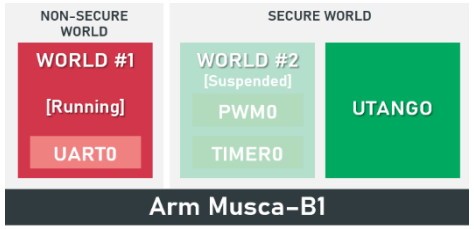
- The SAU is **enabled**
- The **non-secure execution context** for the **1st world** to run is prepared
- Remaining worlds are **suspended** in the **secure** side until a **next context switching**

Bus filters are also configured to match the **memory** and **devices** partition settings

- The **assumption** is that other **bus master** besides the CPU (e.g. DMA, crypto engines) are **sandboxed** into the **secure** side
- This **prevents** the **reconfiguration** of each bus filter during the context switching and the inherent performance penalties

what happens during a SCHEDULING point?





SCHEDULER

1

Save on the **WCB** structure the **context** of:

- CPU registers
- SCB registers
- Interrupts state

Save **Suspended World** Context

2

Schedule **Next World** to be Resumed

Effortless scheduling algorithm

- Round-robin fashion

3

Set **SAU** with the **memory regions** of the **world** to be **resumed**.

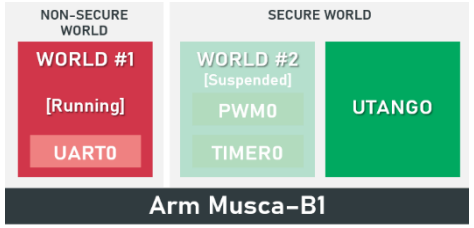
Set new **Memory Security** settings

4

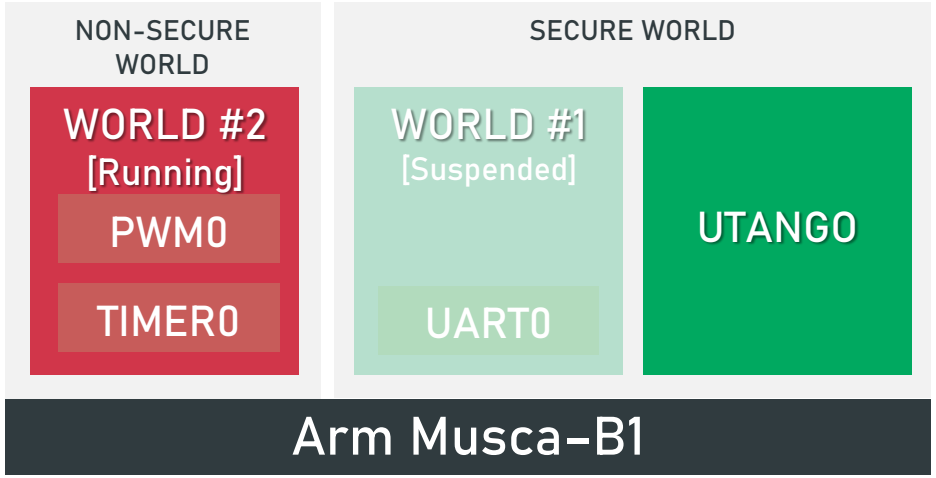
Restore **Resumed World** Context

Restore from the **WCB** structure the **context** of:

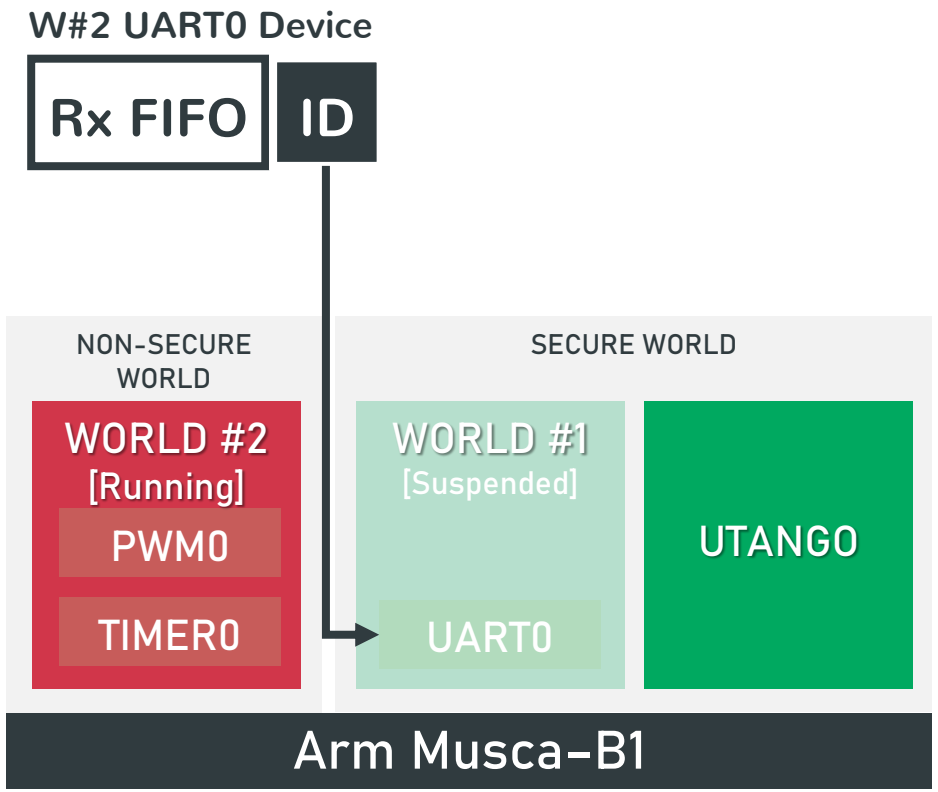
- CPU registers
- SCB registers
- Interrupts state


NON-SECURE WORLD
SECURE WORLD
WORLD #1
[Running]
UART0
WORLD #2
[Suspended]
PWM0
TIMER0
UTANGO
Arm Musca-B1

SCHEDULER


NON-SECURE WORLD
SECURE WORLD
WORLD #2
[Running]
PWM0
TIMER0
WORLD #1
[Suspended]
UART0
UTANGO
Arm Musca-B1

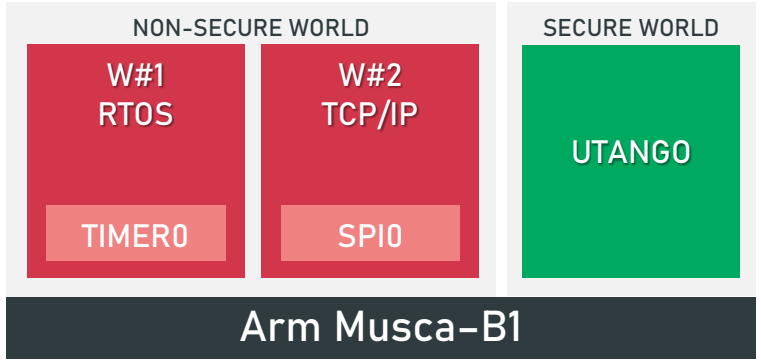
and what happens
if an interrupt of a
SUSPENDED WORLD
takes place?



INTERRUPT HANDLING

For now...

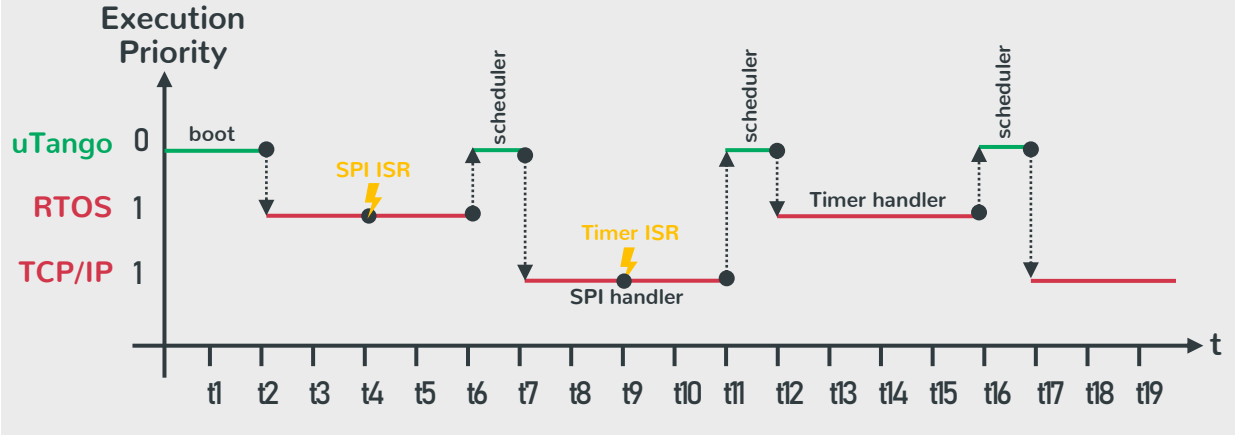
when an **interrupt** of a **suspended world** takes place, the interrupt will only be **served** when the **world is resumed** on a next scheduling point



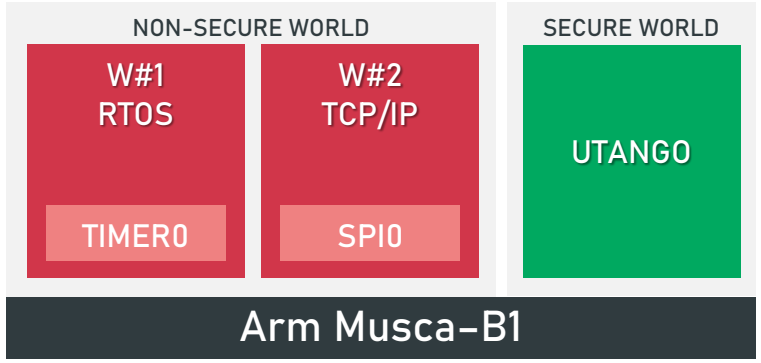
W#1: RTOS
no priority
W#2: TCP/IP
no priority
uTango
tick: 4t

- **Impracticable**, but "workable" at least to materialize uTango proof of concept
- In rough terms, the worst time in which an interruption may take to be served is given by the following equation:

$$(worlds - 1) * tick$$



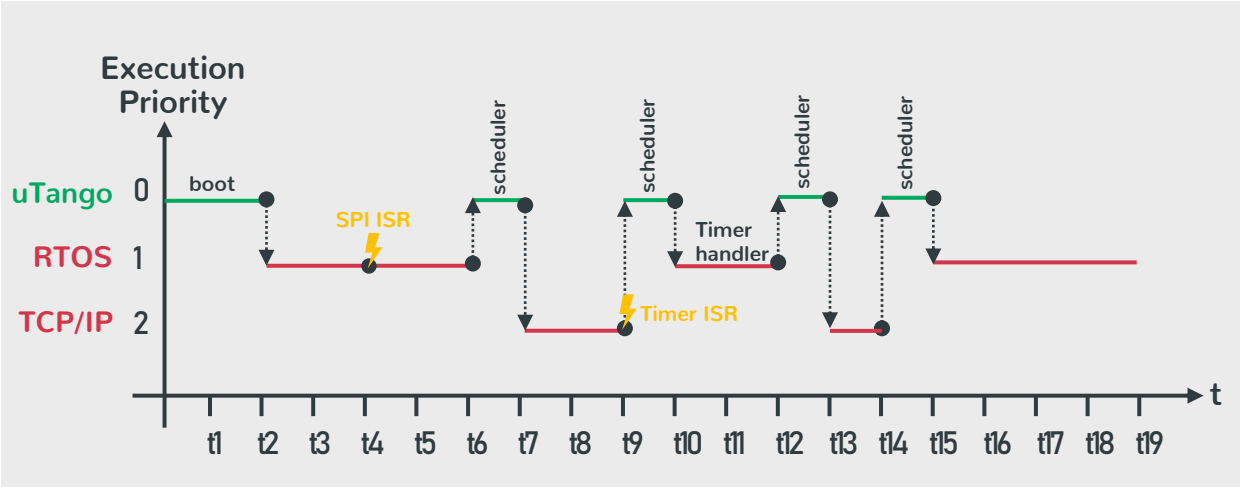
INTERRUPT HANDLING



W#1: RTOS
priority 1
W#2: TCP/IP
priority 2
uTango
tick: 4t

Currently, we are working ...

to allow an interrupt from a **suspended world** to be able to **preempt** uTango and **force** a new **context switch** to **resume** that world



- The preemption behavior will be **priority-driven.**
- A world with **higher criticality** can preempted a **lower criticality** world, but the opposite is not possible!

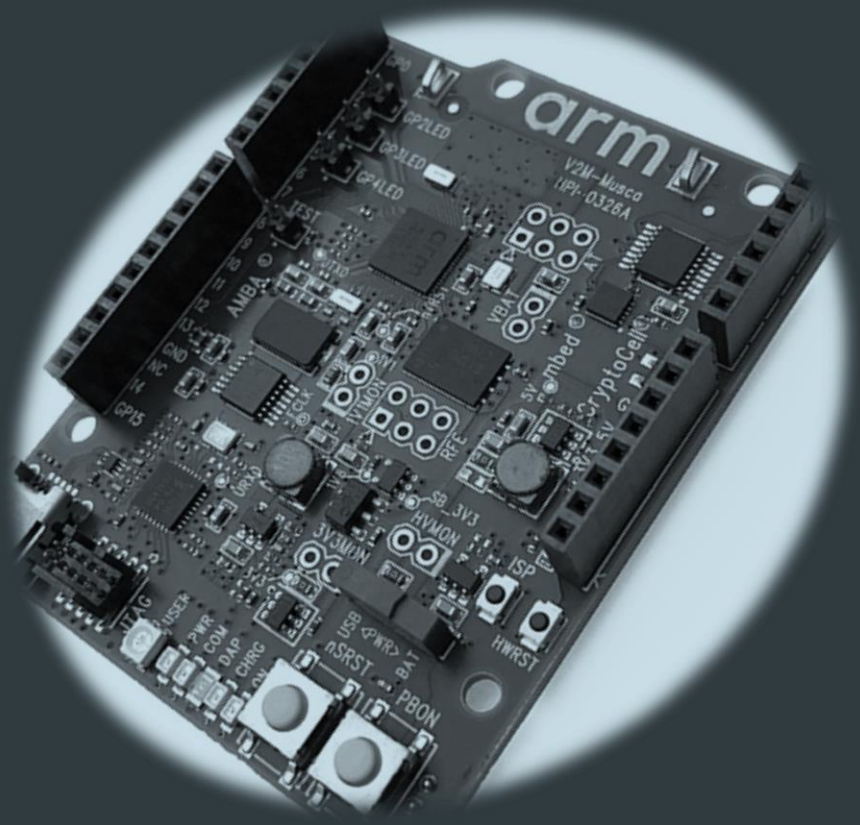
The Numbers

Tests, experiments, and results (Arm Musca-B1)

EXPERIMENTAL SETUP

Hardware

- **Arm Musca-B1 board (SSE-200)**
- **CPU0 @40MHz 2KB_ICache**
- **CPU1 @40(-160)MHz 2KB_ICache**



System Configs

- **uTango v0.2**
  - W1: LED Blinking
  - W2: Zephyr (control. a servo motor)
  - W3: Serial Terminal
  - W4: TCP/IP Stack

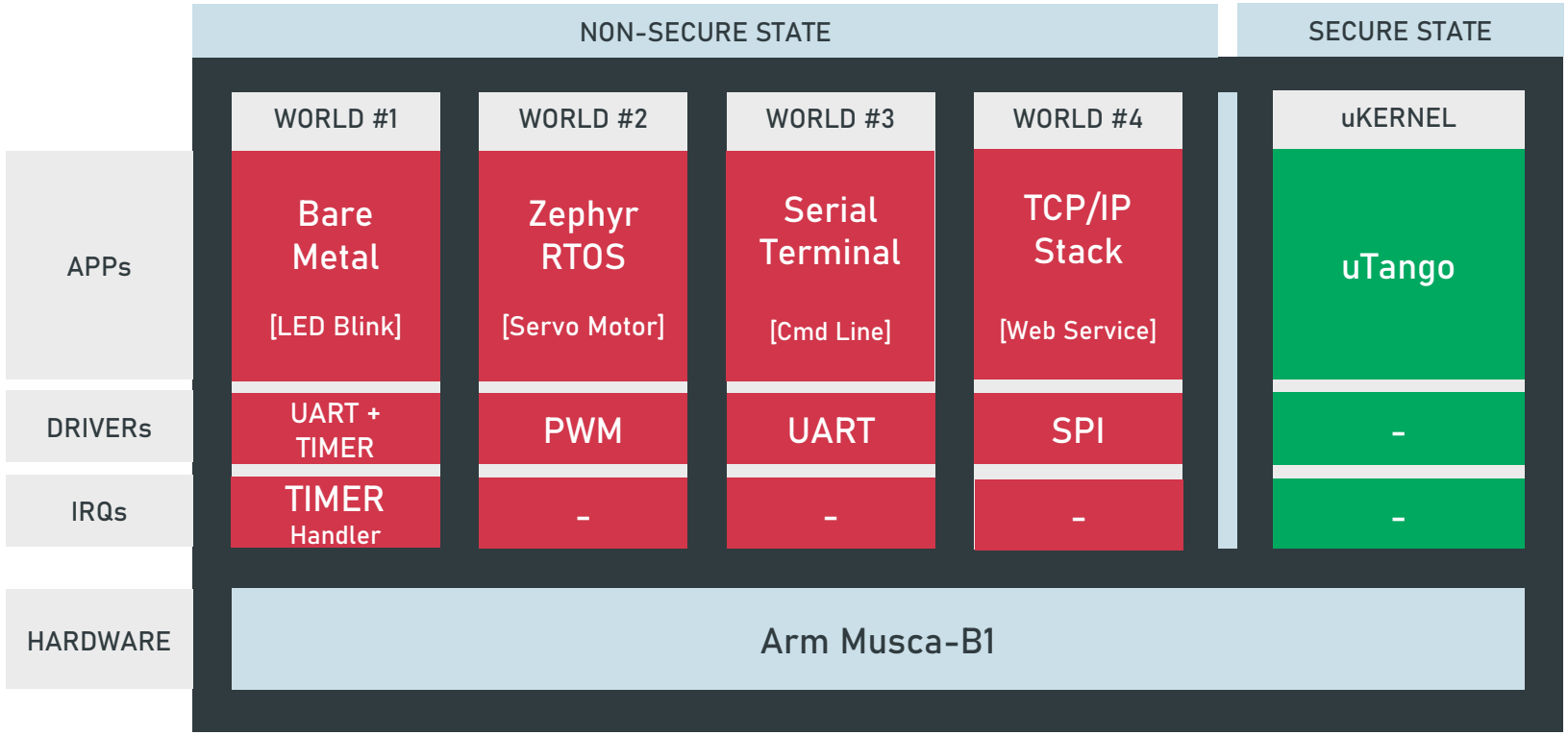
Software

- **Compiler:**
  - GNU Arm Embedded Toolchain (9-2020-q2-update)
  - Optimization: -O0
- **Measuring Unit:**
  - Data Watchpoint and Trace (DWT) unit

Metrics

- **Performance Overhead**
  - Time that uTango kernel spends while switching between worlds
- **Trusted Computing Base (TCB)**
  - Memory footprint (size)
  - Code Complexity (lines of code)

UTANGO CONFIGURATION



MEMORY FOOTPRINT

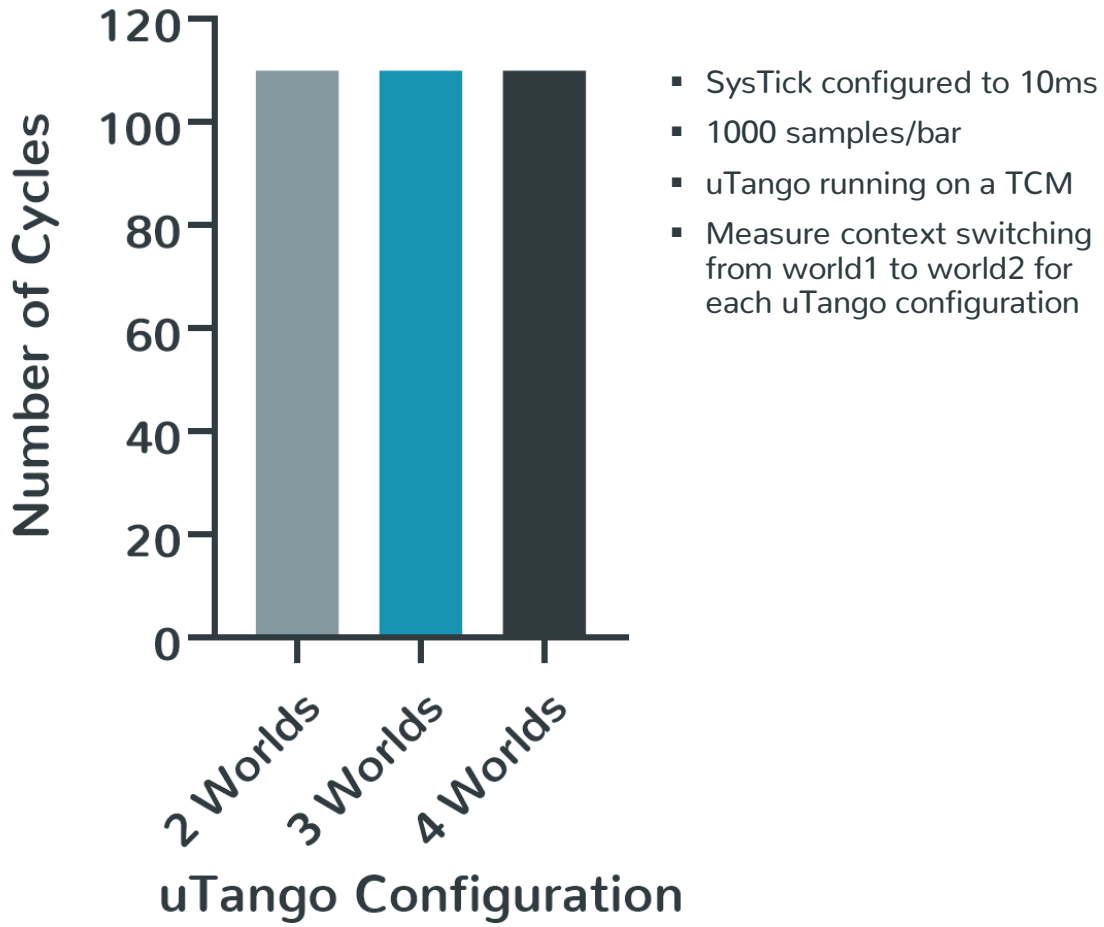
text	data	bss	total
4618	244	1056	5918 bytes

file extension	total code
.h	1248
.c	574
.S	181
.s	142

uTango relies on the **principle of minimal implementation**, striving to **reduce** the amount of **code** that executes at the **highest privilege level**, which is reflected in a TCB size of around **6 KB** with less than **900 lines of code**.

PERFORMANCE OVERHEAD

	Context Switch Time		
uTango Configuration	2 Worlds	3 Worlds	4 Worlds
Clock Cycles	110	110	110



- SysTick configured to 10ms
- 1000 samples/bar
- uTango running on a TCM
- Measure context switching from world1 to world2 for each uTango configuration

We consider **performance overhead** as the **amount of time** that uTango kernel spends while switching between worlds, i.e. the **context/world switch time**.

The assessed results show that for all **uTango configurations**, the time taken during the context switch is precisely **110 clock cycles**.

PERFORMANCE OVERHEAD

CPU Freq (MHz)	Context Switch Time (ns)		
40	2750	2750	2750
160	688	688	688

uTango Tick (ms)	Performance Overhead (%)		
1	0.07	0.07	0.07
5	0.01	0.01	0.01
10	0.007	0.007	0.007

160 MHz

For all system configurations (i.e. 2, 3, 4 worlds), the performance **degradation**
(CPU frequency at 160 MHz and uTango running in a TCM memory) is **null** or **very small** (**<~0.1%**).

Conclusion

Takeaway Points

TAKEAWAYS

1 We demonstrate why the **traditional dual-world** security model provided by TrustZone is becoming **impractical** and **inherently insecure** as the number and complexity of software integrated on embedded devices is increasing at a rapid pace.

2 We presented a **disruptive** approach that **solely** relies on the hooks of the elephant in the room: **Arm TrustZone-M**. We demonstrate how to take advantage of the same TrustZone hardware security primitives to **extend** the **dual-world model** to a **multi-world approach**.

3 We evaluate our system in a **real-world platform** and targeting a **real use-case scenario** to demonstrate that **security** doesn't always need to come at a **significant cost** (uTango achieve values of **performance overhead < 1%**).

We have presented the **1st Multi-World TEE** for modern **TrustZone-M** devices which are expected to be deployed on **billions of tomorrow's Arm MCUs**.

ONGOING WORK

1. Wrapping up the **first release version** of uTango as soon as we finish the new interrupt handling mechanism.

   github.com/danielRep/utango

2. Support for other **commercial platforms** is on the way (e.g. Nuvoton M2351, NXP LPC55S69).

3. Next major steps will focus on **multi-core support** and a **schedulability analysis** of uTango to improve **predictability** and extend the target to **mixed-criticality systems**.

THANK YOU!

Daniel Oliveira (UMinho) | Sandro Pinto (UMinho)

daniel.oliveira@dei.uminho.pt

LinkedIn - https://www.linkedin.com/in/danieljcoliv/
ResearchGate - https://www.researchgate.net/profile/Daniel_Oliveira12
Github - https://github.com/danielRep/

sandro.pinto@dei.uminho.pt

LinkedIn - https://www.linkedin.com/in/sandro-pinto-phd-40535455/
ResearchGate - https://www.researchgate.net/profile/Sandro_Pinto2
Github - https://github.com/sandro2pinto/

Q&A