Reconciling Security with Virtualization: A Dual-Hypervisor Design for ARM TrustZone

Giorgiomaria Cicero*, Alessandro Biondi*, Giorgio Buttazzo* and Anup Patel†
*Scuola Superiore Sant'Anna, Pisa, Italy
†Individual Researcher, Bangalore, India
Email: {giorgiomaria.cicero, alessandro.biondi, giorgio.buttazzo}@santannapisa.it, anup@brainfault.org

***Abstract*—This paper proposes a novel design to enable the virtualization of both secure and non-secure worlds offered by ARM platforms with TrustZone technology. The design is based on a dual-hypervisor scheme that allows executing multiple two-world domains in isolation, where each of them can comprise both a standard (i.e., non-secure) execution environment, and a trusted execution environment (TEE). An implementation of the proposed design is presented and discussed by building upon Xvisor, a Type-1 open-source hypervisor. Experimental results to assess the performance of the implementation are finally reported and discussed.**

I. Introduction

Virtualization and security features are becoming of paramount importance in the design of modern cyber-physical systems. For instance, *hypervisors* (also called virtual machine monitors) represent a de-facto solution to share a common platform among multiple virtualized domains, each possibly executing different operating systems. Hypervisors provide isolation between independently developed components with different criticality levels, supporting fault-tolerance requirements. They can also achieve a higher system utilization while ensuring a proper control on temporal and spatial interference.

More recently, virtualization techniques have also been adopted as basic mechanisms to develop systems with *multiple independent levels of security* (MILS), which are designed to keep a strict separation between secure computing services that use confidential/sensible data (e.g., cryptographic keys) and other untrusted software components running on the same platform. However, due to the increase of software complexity and the exposure of modern systems to connectivity infrastructures, security became a crucial design objective, originating strong functional and reliability requirements that cannot generally be achieved with standard virtualization or other pure software techniques. To meet such requirements, chip makers are moving towards architectures that offer hardware-based solutions for realizing *trusted execution environments* (TEEs). One of the most popular solutions is the TrustZone technology proposed by ARM. ARM TrustZone provides hardware-based isolation of the computing resources shared between a *secure* and a *non-secure* world: the former is conceived to support the execution of a TEE, while the latter corresponds to the normal operating mode of the processor where *rich execution environments* (REE) (e.g., a software system running upon Linux) can be executed.

State of the art. No much effort has been devoted so far to develop virtualization infrastructures for the ARM TrustZone. Pinto et al. [1] proposed a centralized solution based on

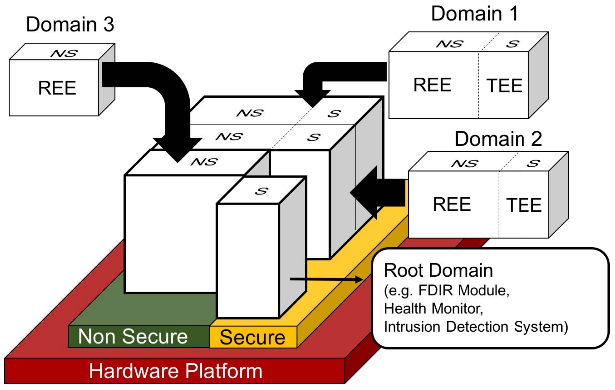

Figure 1. Illustration of the functionality provided by the proposed dual-hypervisor design. The design allows executing multiple domains with mixed criticality and security levels, where each of them can comprise both a REE and a TEE. The design also includes a root secure domain that can be used to host highly-privileged tasks such as fault detection, isolation and recovery (FDIR), health monitoring, or intrusion detection.

a single hypervisor, denoted as LTZVisor, which runs in secure world to manage the whole platform. This design choice implies that all the interactions between the virtualized non-secure domains and the hypervisor (e.g., hypercalls) require a world switch. Very recently, Cicero [2] presented some preliminary results on a dual-hypervisor solution in the context of embedded systems that led the foundations for the present paper. Later, Hua et al. [3] also evaluated a similar dual-hypervisor solution to virtualize TrustZone in the context of cloud computing. However, this solution has not been considered and implemented by the authors since they found a centralized approach with a single hypervisor in non-secure world more suitable for their objectives. To the best of our knowledge, these are the only solutions that provide virtualization of ARM TrustZone. Other works support the execution of a TEE, running in secure world, together with a virtualized environment executing in non-secure world. In particular, Paolino et al. [4] proposed a solution, denoted as T-KVM, in which a hypervisor in non-secure world enables virtual machines on calling trusted applications (placed in the secure world) but running upon a single TEE kernel. Lucas et al. [5] proposed VOSYSmonitor, a low-level software layer running in secure world to enable predictable execution of a TEE in the presence of a virtualized T-KVM environment executing in non-secure world.

Contribution. The work presented in this paper aims at reconciling virtualization and hardware-based security capabilities

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on December 02,2024 at 11:32:22 UTC from IEEE Xplore. Restrictions apply.

by proposing a dual-hypervisor design for ARM platforms equipped with the TrustZone technology. The proposed design acts as a virtualization layer for both secure and non-secure worlds, enabling the execution of multiple isolated domains, each comprising both a REE and a TEE (or only one of them), as illustrated in Figure 1. REEs are managed by a hypervisor that exploits the ARM *virtualization extensions* (ARM-VE) in non-secure world, while TEEs are handled with a *minimal* hypervisor that relies on para-virtualization in secure world. The two hypervisors are jointly configured to enable a transparent communication between REEs and TEEs.

The proposed design carries considerable benefits in terms of performance, reliability, and security. First, being each TrustZone world managed by a dedicated hypervisor, the proposed solution does not require to undertake a switch between secure and non-secure world to handle the virtualization of REEs, thus limiting the runtime overhead suffered by the domains. Second, it avoids the presence of a single point of failure—if the virtualization mechanisms for the non-secure world crash, the TEEs running in the secure world continue to operate. Third, it limits hyperjacking by reducing the attack surface from the non-secure world, because the virtualization mechanisms for the TEEs are only exposed to the latter, and because the hypervisor running in secure world is minimal as it does not have to handle complex REEs.

The proposed dual-hypervisor scheme has been implemented with Xvisor [6], a monolithic Type-1 open-source hypervisor. The most relevant implementation issues that have been encountered are discussed together with a set of practical considerations that emerged during the implementation. Finally, the paper reports a set of experimental results to evaluate the impact of the proposed approach in terms of run-time overhead, latency, and memory footprint.

II. Essential Background

To make the paper self consistent, this section briefly describes the main tools and technologies adopted in this work.

a) ARM TrustZone: ARM TrustZone [7] is a hardware-based technology that aims at providing strong isolation between two execution environments within the same platform. The main concept is that the processor is physically split into two worlds, namely secure and non-secure world, supporting safe communications between them by introducing a super privileged mode, called *monitor mode*, which belongs to the secure world.[1] The architecture includes hardware protection for isolation of memory, bus transactions, interrupts, and peripherals. The technology introduces a special instruction, called Secure Monitor Call (SMC), as a communication primitive between secure a non-secure world. This instruction is basically a synchronous exception handled by the secure code running in monitor mode.

b) Xvisor: Xvisor [6] is an open-source Type-1 hypervisor (i.e., native), which aims at providing a monolithic, light-weight, portable, and flexible virtualization solution. It provides high performance and low memory footprint virtualization capabilities for various ARM architectures (with and without virtualization extensions) and for other CPU architectures, including x86. The hypervisor allows executing multiple domains (also referred to as virtual machines or guests), where each of them can run a different instance of an OS (e.g., Linux, which is the primary OS supported by Xvisor). Each domain disposes of a set of virtual CPUs (VCPU), which are assigned to physical CPUs by the hypervisor scheduler. Virtualization of peripheral devices is achieved via emulation. Pass-trough access is also available for some devices.

c) ARM Fast Models and Cycle Models: ARM Fast Models [8] is a tool that enables users to build, execute, and debug a collection of virtual platforms. The tool enables software development for ARM platforms, such as drivers, operating systems, and firmware, without disposing of the actual hardware. ARM Fast Models allows full control over the simulation, including profiling, fine-grained debug, and trace. ARM Cycle Models [9] is another tool that can be used to precisely analyze the software performance as latencies, bandwidth, cache metrics, and transaction counts, by means of cycle-accurate simulations of ARM processors. The tool enables the analysis of many classes of software and provides software and system debugging features to quickly identify issues during software development.

III. Dual-hypervisor Design

As anticipated in the introduction, the proposed design consists of two jointly-configured hypervisors, providing virtualization in the non-secure and secure worlds, respectively, as offered by ARM TrustZone. To ease the presentation, the two hypervisors will also be referred as non-secure hypervisor (NSH) and secure hypervisors (SH). For the purpose of this section, no specific hypervisor is taken in consideration, as the proposed design is general and can be applied to several proposals. An implementation of the design is later presented in Section IV.

The NSH handles the execution of a set of domains each hosting a REE; consequently it must provide a rich set of functionality that are typical of virtualization mechanisms for complex operating systems such as Linux or Android. The NSH can also make use of the ARM virtualization extensions to realize virtualization. In a dual manner, the SH provides virtualization for a set of secure domains each hosting a TEE. Considering the typical limited complexity of the software infrastructures of TEEs, which at most comprise a small real-time operating system (e.g., see [10]), the SH provides a restricted set of functionality with respect to the NSH. Nevertheless, since today's TrustZone-enabled processors do not provide the secure world with virtualization extension, the SH must implement para-virtualization.

To enable a transparent virtualization of the platform, the design handles the interactions between secure and non-secure world with dedicated mechanisms and a joint configuration of the two hypervisors. Specifically, the hypervisors must support the communication between the REE and the TEE of the same domain: to this end, SMC instructions executed by a REE are intercepted by the NSH to notify the corresponding TEE in the secure world. The dual case holds for SMC instructions issued by a TEE. Furthermore, the configuration of the two hypervisors must also take into account possible shared memory buffers between a REE and the corresponding TEE, which are handled with a matching configuration of the virtual address space of the involved domains.

[1]This paper adopts the terminology introduced by ARMv7 platforms. Note that a different terminology is adopted for new ARMv8 platforms.

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on December 02,2024 at 11:32:22 UTC from IEEE Xplore. Restrictions apply.

Finally, the two hypervisors are orchestrated by a bare-metal firmware, denoted as X Monitor, which handles the system boot, the switch and the communication between secure and non-secure world, and is in charge of dispatching interrupt signals. The X Monitor executes in monitor mode, which is the highest privileged mode offered by TrustZone.

A. *Example of usage*

To better explain the practical usage of the proposed design, consider a set of four domains, denoted as Domain 1, 2, 3 and 4, to be executed upon a TrustZone-enabled platform. Domains 1 and 2 include both a REE and TEE, while Domain 3 consists of only a REE, and Domain 4 consists of only a TEE. As it is illustrated in Figure 2, Domains 1 and 2 are split into two sub-domains to handle their REEs and TEEs with the corresponding hypervisors. Conversely, Domains 3 and 4 are not split, and execute upon the NSH and the SH, respectively.

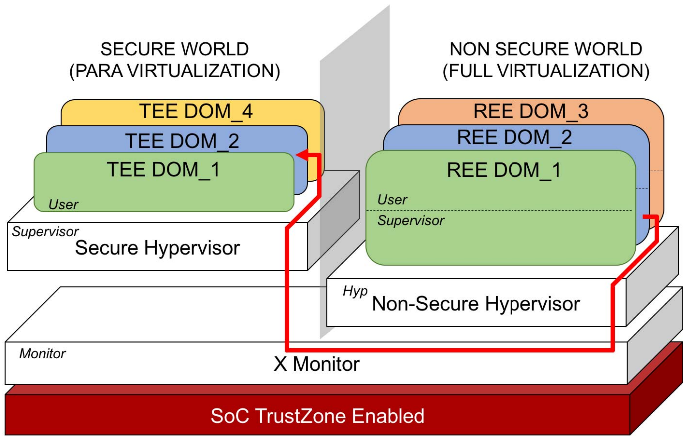

Figure 2. Illustration of the software layers for the proposed dual-hypervisor design. The figure also illustrates the allocation of four domains. Domain 1 and Domain 2 comprise both a REE and a TEE: the former is deployed as a domain managed by the non-secure hypervisor, while the latter is deployed as a domain managed by the secure hypervisor. The red arrow depicts the communication path between the REE and the TEE of Domain 2.

B. *Scheduling the two hypervisors*

The coexistence of two hypervisors on the same platform is favored by the separate hardware contexts offered by Trust-Zone. Nevertheless, it requires introducing a further scheduling layer to handle their dispatching, which in the proposed design is realized by the X Monitor.

Two events may trigger the switch between secure and non-secure worlds: **(i)** the execution of SMC instructions, and **(ii)** the arrival of asynchronous interrupt signals in non-secure world. The first event corresponds to an explicit request of world switch, e.g., similarly to a cooperative preemption. In this case, one of the two hypervisors invokes the X Monitor that will undertake the actual world switch. Concerning interrupt signals, the X Monitor implements a *fixed-priority scheme*: interrupts belonging to the software running in the secure world (denoted as secure interrupts) have higher priority with respect to the ones related to the non-secure world (denoted as non-secure interrupts). As a result, non-secure interrupts that arrive in secure world are masked and marked as pending. Conversely, secure interrupts that arrive when the processor is in non-secure world result in a preemption to switch to the secure world. The resulting behavior is illustrated in Figure 3.

Note that, since the secure world cannot be preempted by the non-secure world, this solution gives higher predictability to the secure world at the cost of degrading the perfomance of the non-secure world. Nevertheless, this policy allows shielding the secure world from denial-of-service attacks (e.g., in the case where the system is flooded with interrupts) and allows guaranteeing timing requirements for the software running upon secure world (e.g, a real-time operating system).

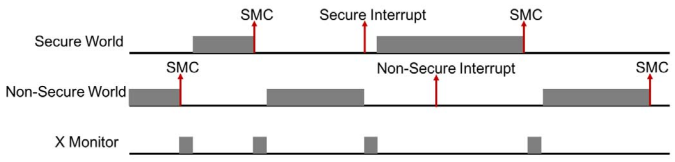

Figure 3. Example scheduling of secure and non-secure worlds. SMC instructions trigger an explicit world switch. Secure interrupts that arrive in non-secure world trigger a world switch, while non-secure interrupts that arrive in secure world are masked.

IV. Implementation

This section presents an implementation of the design introduced in the previous section. The implementation has been carried out by building upon Xvisor [6]. Among the limited number of open-source hypervisors, Xvisor has been selected for its low memory footprint, which represents a relevant requirement in a design with two hypervisors. In fact, although some platforms allow reserving a portion of the main memory for the secure world, which is accomplished by means of the TrustZone Address Space Controller (TZASC), other platforms require that the secure world software has to fit in a dedicated small memory with a typical size of 32 or 64 MB. Furthermore, the adoption of a small hypervisor also helps in reducing the attack surface in secure world.

Starting from the release v. 0.2.9 of Xvisor, two new versions have been developed to support the proposed dual-hypervisor design, namely Secure Xvisor (S-Xvisor) and Non-Secure Xvisor (NS-Xvisor), while the X Monitor has been implemented from scratch. These components are described in the following sub-sections. The implementation has been carried out with the ARM Fast Models tool emulating the FVP ARM Versatile Express platform equipped with a single Cortex-A15 processor. The current version does not allow multiple pending interactions between a REE and a TEE at the same time, i.e., the TEEs that comprise SMC instructions are passive software components invoked by REEs to provide on-demand secure services.

A. *Secure Xvisor*

Due to the lack of the ARM virtualization extensions in secure world, the S-Xvisor is configured to work with para-virtualization, implementing a classical trap-and-emulate paradigm for most instructions, and requiring to patch the domains for the other sensitive instructions that do not generate an exception in user mode. Luckily, these features were already supported by Xvisor. The main modifications that were required to support our design concern the idling mechanism of Xvisor, the virtualization of the SMC instruction, and the hypervisor configuration.

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on December 02,2024 at 11:32:22 UTC from IEEE Xplore. Restrictions apply.

Idling mechanism. Natively, whenever there is no ready workload to execute, Xvisor executes a wait for interrupt (WFI) instruction so that the processor can switch to the sleep mode in order to save energy until it will be awaken by an interrupt signal. To handle the co-existence of the two hypervisors, this behavior has been modified by enforcing a world switch whenever the S-Xvisor goes idle. That is, instead of executing the WFI instructions, the S-Xvisor executes a SMC instruction to pass the baton to the NS-Xvisor. This logic has been implemented in an infinite loop executed by the Idle Orphan, which is a special VCPU of Xvisor that executes with the lowest priority. Before calling the SMC instruction, the loop starts by saving the context of the secure world with the program counter pointing to the instruction after the SMC. After the SMC, the loop checks which event resumed the execution of the secure world, i.e., an SMC executed from non-secure world or a secure interrupt. In the first case, the execution of the secure world will restart from the code following the SMC in the loop of the Idle Orphan, which is responsible of handling the requests issued by the non-secure world: further details on this phase will be provided in Section IV-D.

In the second case, once the execution will be resumed, the Idle Orphan is immediately preempted by the interrupt service routine (ISR) to be served. The ISR could trigger the activation of some secure domains that will also prevent the Idle Orphan to execute until they terminate to execute. To distinguish from the case in which the Idle Orphan is awaken by a SMC instruction, the X Monitor changes the saved context of the Idle Orphan (by modifying some registers) to notify the absence of requests issued from the non-secure world.

SMC virtualization. To allow the coexistence of multiple two-world domains, the SMC instruction must be properly virtualized. In fact, if SMC instructions are handled with a pass-through scheme, their execution originates a world switch *independently* of the execution state of other secure domains. This strategy would generate additional temporal interference to secure domains, contradicting the fixed-priority scheme introduced in Section III-B. To address this issue, whenever a TEE executes a SMC, the SMC is trapped by the S-Xvisor that stores it as pending and deschedules the corresponding VCPU. The actual SMC will then be executed by the Idle Orphan when all the secure domains complete their execution.

Configuration. To enable a joint configuration of the two hypervisors, Xvisor has been enriched with a software module that handles the communications between the domains. The module allows specifying pairs of domains running in secure and non-secure world, respectively, that are allowed to communicate by means of SMC instructions, and is in charge of ensuring that such communications do not violate the configuration. Furthermore, by building upon the memory virtualization capabilities offered by Xvisor, it is possible to specify shared memory buffers that are mapped in the virtual address space of two communicating sub-domains, e.g., by automatically generating a symmetrical configuration entry for both the hypervisors.

B. Non-secure Xvisor

Different from the S-Xvisor, the NS-Xvisor can leverage the ARM virtualization extensions, and hence do not require to support para-virtualization. The modifications required for the NS-Xvisor are analogous to the ones discussed for the S-Xvisor, with the exception of the idling mechanism that is not required in non-secure world, i.e., when the non-secure world is idle, then there is nothing else to execute and the processor can sleep by means of WFI instruction. Since inter-world communications are initiated by the non-secure world, the logic of the SMC virtualization is a bit more complex. This aspect is discussed in Section IV-D.

C. X Monitor

The X Monitor is a bare-metal firmware that has been developed from scratch to orchestrate the two hypervisors and to bootstrap the platform. It executes in monitor mode, which is the highest privileged mode belonging to the secure world, and is mainly responsible for handling the world switch by means of SMC instructions and for dispatching the interrupts.

At the system startup, the X Monitor initializes the Generic Interrupt Controller (GIC). The GIC is composed of a distributor and a CPU interface: in the presence of the TrustZone technology, the distributor allows grouping interrupts in secure and non-secure, while the CPU interface allows distinguishing between fast interrupts (FIQ) and standard interrupts (IRQ). As typical of interrupt controllers, the interrupts have priorities. In the realized implementation, the GIC is configured to assign the upper half of the priority values to secure interrupts, and the remaining ones to non-secure interrupts, thus matching the fixed-priority scheme introduced in Section III-B.

As a best practice, ARM recommends to handle secure interrupts as FIQs [11] and non-secure interrupts as IRQs, consequently disabling FIQ masking from non-secure world and configuring the processor for catching FIQs in monitor mode. However, Xvisor is not conceived to handle FIQs, thus originating a problem for the S-Xvisor. To avoid heavy modifications of Xvisor, the interrupt routing strategy illustrated in Figure 4 has been adopted. As it can be observed from the figure, while the processor is executing in non-secure world, secure interrupts are configured as FIQs, and are hence handled in monitor mode by the X Monitor. Conversely, while the processor in executing in secure world, secure interrupts are configured as IRQs and can be directly handled by the S-Xvisor. In correspondence to each secure interrupt raised in non-secure world, the X Monitor undertakes a world switch and sets the secure context for the GIC in which secure interrupts are configured as IRQs. This reconfiguration automatically forwards interrupts from FIQs to IRQs. Finally, non-secure interrupts are always configured as IRQs: when the processor is executing in non-secure world, they are regularly handled at hypervisor mode, while they are masked (and stored as pending) in secure world.

The X Monitor provides two types of world switch depending on the triggering event: a *lite world switch*, which is performed in correspondence to the execution of SMC instructions, and *full world switch*, which is performed when the X Monitor handles a secure interrupt. This distinction is required because SMC instructions are generally used for inter-world communication purposes, where part of the context of one world (i.e., some registers) is used for exchanging the information and must hence be preserved. More specifically, the ARM SMC Calling Convention [12] specifies that the first eight general purpose registers (`r0-r7`) are reserved to enable communications between the two worlds. As a consequence,

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on December 02,2024 at 11:32:22 UTC from IEEE Xplore. Restrictions apply.

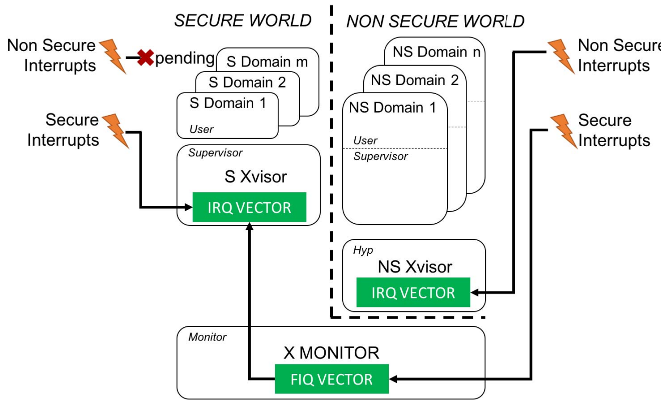

Figure 4. Overview of the interrupt routing strategy.

during a lite world switch, the X Monitor preserves the values of r0-r7, while it saves and restores the entire context of both worlds in the presence of a full world switch. To allow the Idle Orphan of S-Xvisor to identify that it has been resumed by an interrupt, the full world switch also forces the value of the first four registers (r0-r3) to zero.

D. Inter-world communication

The domains that include a TEE need to establish communications between secure and non-secure world. The proposed implementation comes with a communication protocol that follows the ARM SMC Calling Convention [12]. This convention mainly defines that arguments and return values are passed in the first eight general purpose registers, while the immediate value of the SMC instruction shall be kept to zero. The first register (r0) is reserved for storing the so called SMC Function ID, which defines the type of request issued to the TEE, while the remainder seven registers can be managed by the designer. In the realized protocol, the second register (r1) is used to identify the TEE service invoked by non-secure world, and registers r2-r5 are used to identify two shared-memory buffers for realizing the communication. Registers r2-r3 contain the address and the size of the source buffer, respectively, which is populated by non-secure world, while r4-r5 contain the address and the size of the destination buffer, which is managed by secure world. As it is illustrated in Figure 5, these registers are modified at the stage of SMC virtualization. First, since the dual-hypervisor design allows the coexistence of multiple virtualized TEE, the TEE identifier in r1 is modified by the NS-Xvisor to match the proper secure domain managed by S-Xvisor (which is determined as a function of the configurations described in Sections IV-A and IV-B). Second, the addresses of the source and destination buffers are also modified to cope with the different virtual address spaces with which the two communicating domains can be configured. For simplicity, if secure domains do not use memory virtualization, this phase simply consists in replacing virtual memory addresses with the corresponding physical memory addresses. Third, the NS-Xvisor also stores in r7 the identifier of the non-secure domain that issued the SMC. Finally, when returning to the non-secure world after a secure execution, the first four registers are reserved to store values returned from the secure world, and the NS-Xvisor undertakes a dual address translation to restore the correct virtual address in r4.

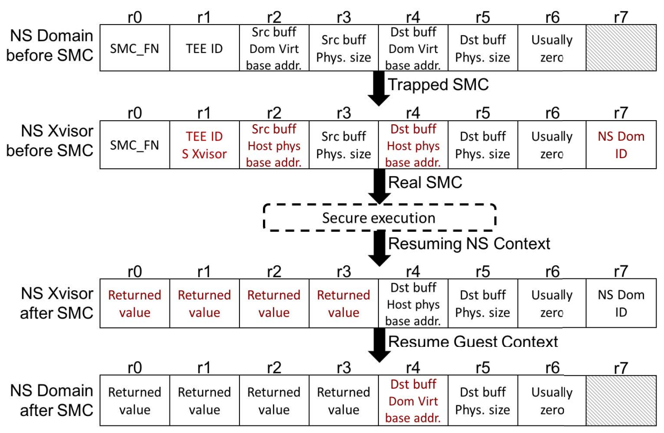

Figure 5. Modification of the general purpose registers due to SMC virtualization.

When the secure execution is resumed by a SMC instruction, the Idle Orphan of the S-Xvisor starts parsing the request and performs sanity checks based on the content of registers r0-r7, i.e., checking the compliance with the established convention, or verifying the correctness of the identifiers.

Memory map. Figure 6 illustrates the memory map adopted for the proposed implementation. The RAM address space is simply split into a secure and a non-secure part: this split can be realized with the TZASC or is implicitly adopted whenever a dedicated RAM is used for the secure world. As it can be noted from the figure, the secure world is able to access the entire RAM address space. The code of the X Monitor is loaded at the beginning of the secure RAM, while a shared region is reserved at the end of the non-secure RAM to realize the shared buffers for inter-world communication.

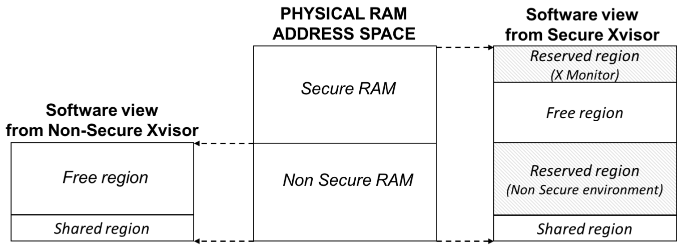
PHYSICAL RAM ADDRESS SPACE
Software view from Secure Xvisor
Software view from Non-Secure Xvisor
Secure RAM
Non Secure RAM
Reserved region (X Monitor)
Free region
Reserved region (Non Secure environment)
Shared region
Free region
Shared region

Figure 6. Illustration of the adopted memory map and the memory view from both secure and non-secure world.

V. Experimental results

The realized implementation has been evaluated by measuring the memory footprint and the timing latencies introduced by the dual-hypervisor scheme. The evaluation has been carried out on ARM Fast Models and, whenever possible, on ARM Cycle Models, thus obtaining cycle-accurate measurements. The Linaro GCC 5.3 compiler has been adopted.

Memory footprint. Table I reports the memory footprint (in bytes) for the implemented components, which has been measured by using the GNU size tool. As it can be observed from the table, the total footprint is considerably limited and amounts to less than 3.3 MB, although both hypervisors include the default drivers and emulated devices that come

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on December 02,2024 at 11:32:22 UTC from IEEE Xplore. Restrictions apply.

with Xvisor. Also note that the footprint of the X Monitor is very minimal (less than 3 KB). It is worth mentioning that, at run-time, Xvisor frees temporary memory areas (e.g., those containing initialization procedures), hence the actual steady-state footprint is even less.

Table I. MEMORY FOOTPRINT

Component	*Size [bytes]*			
	.text	*.data*	*.bss*	*Total*
S-Xvisor	1570648	139164	188528	1898340
NS-Xvisor	1218872	89164	146460	1454496
X Monitor	2994	0	0	2994
Total (without domains)				**3355780**

Latencies. The major latencies introduced by the realized implementation are due to the bootstrap phase of the X Monitor, the world switches, and the crossing of the virtualization layers of the two hypervisors. At the system startup, before launching S-Xvisor, the bootstrap phase of the X Monitor initializes the monitor context (interrupts and memory stacks). After S-Xvisor is started, the control is passed back to the X Monitor that enables the virtualization extensions, initializes the GIC and the timers for NS-Xvisor, and configures the access rights of the non-secure world. The maximum net execution times to execute these two phases are reported in the first two rows of Table II. The table also reports the maximum overhead introduced by the two world switches (lite and full) discussed in Section IV-C. As it can be noted from the table, the largest overhead is introduced by a full world switch from non-secure to secure world and amounts to about 22 μs. Note that these results further highlight the advantages in adopting our dual-hypervisor design with respect to a centralized (i.e., single-hypervisor) solution, where a much intense world switch ratio would be present to support the virtualization of REEs.

Table II. CYCLE-ACCURATE LATENCIES

Phase	***Cycles***	***μs@180MHz***
X Monitor initialization	1011	5.6167
X Monitor init. for NS-Xvisor	1512	8.4003
Lite world switch (NS-S)	3186	17.7
Lite world switch (S-NS)	3186	17.7
Full world switch (NS-S)	3893	21.6278
Full world switch (S-NS)	3563	19.794

Another important latency is the duration of the entire chain triggered by a non-secure domain that performs a request to a TEE in secure world via SMC, thus involving the virtualization layers of the two hypervisors (the chain is discussed in Section IV-D and is illustrated in Figure 5). Due to technical issues, Cycle Models prevented us to measure this latency, which instead has been evaluated with Fast Models by means of an emulated timer. The maximum measured latency over 1000 invocations of the chain resulted equal to 249.74 μs on an emulated Versatile Express board running at 60 Mhz. The tested scenario consisted in a single domain with one REE and one TEE. The time needed by the latter to produce data in response to a service request is not accounted in the measurement. Finally, thanks to the fixed-priority scheme adopted in the configuration of the GIC, note that the latency with which the system reacts to secure interrupts is always bounded by the duration of a full world switch from non-secure to secure world.

VI. CONCLUSION AND FUTURE WORK

This paper proposed a dual-hypervisor design for ARM TrustZone to handle multiple two-world domains (secure and non-secure world) in virtualization. An implementation of the design has been then presented by building upon Xvisor, an open-source hypervisor. Two modified versions of Xvisor, namely secure and non-secure Xvisor, have been developed to realize the dual-hypervisor scheme. A third component, denoted as X Monitor, has also been developed to orchestrate the two versions of Xvisor and handle bootstraps, interrupt dispatching, and world switches. Experimental results showed that the entire implementation has a limited footprint (about 3.3 MB) and introduces affordable latencies at run-time. Overall, ARM TrustZone proved to be an excellent candidate to realize virtualized systems with hardware-based secure capabilities, with the only limitation residing in the absence of the virtualization extensions in secure world.

This work lefts open a number of interesting and challenging issues, including the support for multi-core platforms, the support for ARMv8 architectures, the integration of spatial and temporal isolation mechanisms in both the worlds [13], the inclusion of attack mitigation techniques, and the support for popular TEE kernels (e.g., OP-TEE) upon the secure Xvisor. Furthermore, it is also planned to apply the proposed design to different hypervisors.

ACKNOWLEDGEMENTS

The authors like to thank Jason Andrews and Eric Sondhi from ARM for providing licenses of ARM tools and their valuable support in setting up the ARM Cycle Models environment. This work has been partially supported by the RETINA Eurostars Project E10171.

REFERENCES

[1] S. Pinto, J. Pereira, T. Gomes, A. Tavares, and J. Cabral, "LTZVisor: Trustzone is the key," in *In Proc. of the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, 2017.

[2] G. Cicero, "A dual-hypervisor for platforms supporting hardware-assisted security and virtualization," in *Master Thesis in Embedded Computing System, University of Pisa and Scuola Superiore Sant'Anna.*

[3] Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, and H. Guan, "vTZ: Virtualizing ARM trustzone," in *In Proc. of the 26th USENIX Security Symposium*, 2017.

[4] M. Paolino, A. Rigo, A. Spyridakis, J. Fangude, P. Lalov, and D. Raho, "T-KVM: A trusted architecture for KVM ARM v7 and v8 virtual machines securing virtual machines by means of KVM, trustzone, TEE and SELinux," in *In Proc. of the Sixth International Conference on Cloud Computing, GRIDs, and Virtualization*, 2015.

[5] P. Lucas, K. Chappuis, M. Paolino, N. Dagieu, and D. Raho, "VOSYS-monitor, a monitor layer for mixed-criticality automotive systems on armv8 platforms," in *In Proc. of the 29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, 2017.

[6] A. Patel, M. Daftedar, M. Shalan, and M. W. El-Kharashi, "Embedded hypervisor Xvisor: A comparative analysis," in *In Proc. of the 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2015)*, 2015.

[7] Security on ARM trustzone. [Online]. Available: https://www.arm.com/products/security-on-arm/trustzone/

[8] ARM fast models. [Online]. Available: https://developer.arm.com/products/system-design/fast-models

[9] ARM cycle models. [Online]. Available: https://developer.arm.com/products/system-design/cycle-models

[10] Trustonic. Kinibi. [Online]. Available: https://developer.trustonic.com/discover/technology

[11] ARM. ARM generic interrupt controller. [Online]. Available: https://static.docs.arm.com/ihi0048/b/IHI0048B_b_gic_architecture_specification.pdf

[12] ——, "SMC calling convention. system software on ARM platforms," 2016.

[13] P. Modica, A. Biondi, G. Buttazzo, and A. Patel, "Supporting temporal and spatial isolation in a hypervisor for arm multicore platforms," in *Proceedings of the 18th IEEE International Conference on Industrial Technology (ICIT 2018)*, Feb. 2018.

Authorized licensed use limited to: University of Massachusetts Amherst. Downloaded on December 02,2024 at 11:32:22 UTC from IEEE Xplore. Restrictions apply.