GuardianOS - *A SoC architecture for RISC-V Trusted Execution Environments*

1st AZOU Raphaël
Master ASSEL
ENSTA Bretagne
raphael.azou@ensta.fr

2nd LAGADEC Loïc
[Project Supervisor]
Lab-STICC ENSTA Bretagne
loic.lagadec@ensta.fr

***Abstract*—Over the past few years, System on Chips (*SoC*) have become increasingly popular for a wide range of use cases. As they contain more and more software and hardware components that comes from untrusted sources, their attack surface has increased tenfold. This document presents a SoC architecutre, which aims to create trusted execution environments (*TEE*) that can be identified with some id's on the bus to authorize certain memory or peripheral accesses, enabling more granular control over access permissions at the physical level as well as the operating system level.**

***Index Terms*—SoC, TEE, RISC-V, QEMU**

I. Introduction

A. Context

A System on Chip (SoC) is an integrated circuit that consolidates all the primary components of a an electronic system into a single chip. Unlike discrete components like microprocessors or GPUs, a SoC integrates a central processing unit (CPU), memory, input/output ports, or intellectual property blocks that comes from differents parties, making it highly efficient and compact. This integration enables SoCs to power a wide range of devices, from smartphones to embedded systems as well as a wide range of IoT devices. The SoC market is experiencing significant growth, driven primarily by advancements in machine learning and the proliferation of IoT applications [11]. This widespread adoption has also made SoCs a prime target for sophisticated attacks, including privilege escalation and arbitrary code execution. To address these threats, solutions such as Trusted Execution Environments (TEE) have been proposed to enhance the security of SoC-based systems [8, 10].

The aim of the GuardianOS project is therefore to develop a solution to create different trusted execution environments that can be identified with id's that can be then propagate on the bus to authorize or deny IP's accesses. GuardianOS is a complementary solution to the TrustSoC project [7], which focuses on verifying the behavior of intellectual property (IP) components in SoCs.

B. Summary of the solution

In this work, we propose a solution that is based on the seL4 microkernel to create multiple trusted execution environments as well as rich execution environments (REE) on a SoC, and the security of these environments are guaranteed by the capabilities security model of seL4 which has been formally verified for its functional correctness. A capability can be described as a tamper-proof token, used to identify a rights to a given kernel object (i.e memory frame, I/O ports or even address space [3].) GuardianOS contains a task called the "security monitor" which is the first application that boot on seL4 and it gets all of the available capabilities. This security monitor can then spawn the different applications and give them the necessary capabilities based on a defined security policy. The seL4 microkernel has been patched to deliver an id to the smallest unit of execution in seL4, the thread. These id's are then used to check every accesses to the intellectual property blocks (IP's) of the SoC. As explained in the TrustSoC work, each IP's are wrapped with a component that intercepts the ingoing requests, check the id of the originating component (i.e a seL4 thread or another IP) and send a request to a specifif component that is called the "security oracle" to check if there is a matching rule for the association `source:destination`. If a valid rule is found, the request is forwarded to the IP. If not, the request is dropped. The security oracle is a component that is mapped in the security monitor memory space and can only be accessed only by the security monitor.

In order to simulate the SoC architecture, and more specifically the TrustSoC IP's verification, the QEMU emulator has been used.

C. Key Concepts and acronyms

TEE: Trusted Execution Environment. A secure area of a main processor that guarantees code and data loaded inside to be protected with respect to confidentiality and integrity. **REE**: Rich Execution Environment. An execution environment to run

untrusted applications. **RISC-V**: An open-source instruction set architecture (ISA) based on established reduced instruction set computing (RISC) principles. **SoC**: System on Chip. An integrated circuit that integrates all components of a computer or other electronic system into a single chip. **CSR**: Control and Status Registers. Registers that control the operation of the processor and provide status information (RISC-V). **IP**: Intellectual Property. Pre-designed and reusable block of logic or a sub-system that performs a specific function within the SoC. **MMU**: Memory Management Unit. Hardware unit that translates virtual addresses to physical addresses.

II. Related work

SoCs consolidate multiple functionalities, making them powerful yet vulnerable to security threats, including privilege escalation and code execution attacks. TEEs provide isolated execution environments to enhance security, but they have limitations, such as the verification of Intellectual Property (IP) components. GuardianOS aims to create TEEs with improved access control based on execution identifiers. SoCs integrate CPUs, memory, and peripherals, leading to efficiency gains but also increased security risks. They are vulnerable to attacks such as privilege escalation or arbitrary code execution.

Name	ISA	Features			
TrustZone [9]	ARM	●	○	○	○
Keystone [5]	RISC-V	●	○	●	●
Hector-V [8]	RISC-V	○	●	○	●
Intel SGX [6]	x86	●	○	●	○
SANCTUARY [1]	RISC-V	●	○	◐	●
Sanctum [2]	RISC-V	●	○	●	●
sel4-based TEE [8]	RISC-V	●	○	○	●
TrustSoc [7]	RISC-V	●	●	○	●

TABLE I
FEATURES OF TEE SOLUTIONS.
FROM LEFT TO RIGHT:
SINGLE PROCESSOR SUPPORT, IPS VERIFICATION, MULTIPLE SECURE WORLDS SUPPORT, OPEN SOURCE
(●: YES, ○: NO, ◐: PARTIAL).

Comparisons between the existing solutions indicate that no single solution fully addresses the need for robust IP verification and multiple TEEs creation. Therefore, GuardianOS aims to address these limitations by providing a solution that supports multiple secure worlds and enables granular control over access permissions for each world. GuardianOS is based on the TrustSoC project [7], which provides a mechanism for verifying IP behavior.

III. Contribution

GuardianOS provide a solution to create multiple trusted execution environments on a SoC with a granular control over the permissions given to every world (i.e a TEE or a REE). It support software components isolation as well as IP's behaviour verification. It is based on the previous work done in the *TrustSoC* project [7] for the IP's verification mecanism. GuardianOS is based on a patched version of the seL4 microkernel and is a RISC-V solution as it involves the use of a specific type of registers provided by the RISC-V ISA.

A. Patched seL4 microkernel

The first part of GuardianOS is the different applications configuration based on a trust level (i.e trusted or untrusted) for each of these applications. The seL4 microkernel use the capability security model to control every accesses made on the differents kernel objects.

Kernel object	Description
CNodes	Store capabilities.
Thread Control Blocks	Represent a thread of execution.
Scheduling Contexts	Represent CPU time.
Endpoints	Message-passing communication between threads.
Reply Objects	Track scheduling context donation.
Notification Objects	Simple signalling mechanism.
Virtual Address Space Objects	Represent a virtual address space.
Interrupt Objects	Give applications the ability to receive and acknowledge interrupts from hardware devices.
Untyped Memory	Allows the creation of new kernel objects.

TABLE II
KERNEL OBJECTS IN SEL4

Each kernel object is then referenced by a capability that represents the rights on this object. Each thread has a *VSpace* that represents the virtual memory allocated for this thread, and a *CSpace* that represents the list of capabilities that the thread has. Theses capabilities are stored in a CNode that is a list of capabilities. Each element in a *CNode* is called a *CSlot*.

The seL4 microkernel is not an operating system [3], and the system ressources management is fully delegated to the user space, excepting for system calls handling, scheduling and capabilities verification. These ressources should be managed by a crucial component called the *root task*, which is the first application that boots on the seL4 microkernel. The root task is responsible for setting up the system by allocating memory, creating other tasks, and distributing capabilities according to the system's security policies. It can spawn threads with specific permissions by setting capabilities in his CSpace, and establish inter-process communication (IPC) channels using endpoints and notifications. At the boot time, this task receives a structure from the kernel that contains the list of all capabilities and the initial memory regions that can be used.

GuardianOS propose a personalized root task called the "security monitor" that is able to configure and launch different applications that are labeled as trusted or untrusted from a fixed configuration. In this simulator, this configuration is represented as a list of pairs `<application name, trust>`. The security monitor can then pass the necessary capabilities to the applications based on their trust. It provides

a granular way to manage the permissions given to a application as every trusted applications can be manage differently by passing differents capabilities to it. An untrusted application can't be completely isolated from the rest of the system, but it receives only the necessary capabilities to run and nothing more. When a thread want to access a kernel object, it will provides an index representing the CSlot in one of his CNode where the capability for this object is stored. The kernel will then perform a *capability lookup* to check if the thread has the rights to access this object by checking the rights in the capability. Any attempt to access a kernel object without the right capability will result in a *cap fault*.

To manage the different applications interactions with the different IP's of the SoC, an identification mecanism is needed. For this purpose, some modifications has been added to the seL4 microkernel to reflect this feature. In seL4, when a thread is created, the kernel that handle this thread creation invoke a specific function called `seL4_TCB_Configure` that will setup the thread (set the priority for the scheduler, the thread CSpace root, the VSpace etc.) This function has been modified to add a new parameter that is the id of the thread. This id is then stored in the thread metadata structure that is called `tcb_t`. An important point to note is that there is no conventional process object in seL4 like a classic operating system. seL4 provides a process abstraction that is just a configured thread with a dedicated VSpace and CSpace. Therefore, every execution unit creation pass through the mentionned functions that configure a thread. As shown in the table II, A thread can create another thread by retyping an untyped capability that represent the right to create a new kernel object. A thread without an untyped capability in his CSpace can't create a new kernel object. Therefore, the security monitor can be configured to pass such capability to a trusted application to allow it to create new threads but it is not recommanded to do so as the security monitor should be the only one to create new threads and a thread who needs to create a new thread should ask the security monitor to do it by sending a message through an IPC endpoint. Considering that the thread is properly configured with a specific id and capabilities, and started by the security monitor, it can't modify his own id in any way as a modification of these elements would require a capability for the TCB object that is not given to the thread. In comparison, a thread can't set his own scheduling priority by invoking the `seL4_TCB_SetPriority` function unless the security monitor has given the right to do so by passing the scheduling context capability to the thread.

B. RISC-V CSR

Given that seL4 can now pair an id with a thread, a thread activity should be always identified with his id for an IP access. To do that, the seL4 scheduler has been slightly updated to guarantee this requirement. When a thread is scheduled, the scheduler read the id present is the thread `tcb_t` structure, and the id is then stored in a specific RISC-V CSR register. The RISC-V ISA provides a set of CSR's to configure the CPU and provide status information. For instance, the `mstatus` CSR is used to control the CPU's operating mode. Some of these CSR's are reserved for custom use. GuardianOS use the CSR n° `0x5C0` to store the id of a thread that is scheduled. This CSR has been chosen because it is only accessible in supervisor mode, which means that the user space can't read or write in this CSR. This is a security measure to prevent a thread to modify his own id.

C. Patched QEMU

In order to track and propagate the id of a thread or an IP id that is trying to access an IP, some modifications have been made to the QEMU sources to reflect the TrustSoC work. The first modification is the addition of the CSR required to store the id's. By default, QEMU include the primary CSR's listed in the ISA specification [4] but does not implements the custom CSR's. The patch adds the CSR `0x5C0` to the list of supported CSR's in QEMU making it available for GuardianOS. The second modification is the addition of the id propagation in the MMU. When an application wants to access an IP, the MMU will translate the virtual address that represents the base address of the IP to the physical address of the IP in order to dispatch the read or write request. Before dispatching the request, the MMU will read the value present in the CSR `0x5C0` to get the id of the thread that is trying to access the IP. The request is then dispatched with a memory transaction attribute that is the id of the originating component (i.e an IP or a thread). These memory transaction attributes are also used by TrustZone to identify if a request comes from the secure or the non-secure world.

D. IP's simulation and identification

In the GuardianOS simulator, the different IP's are represented as PCI devices. The reason is that QEMU does not currently support the simulation of an AXI bus, which is the most common bus used in SoC. GuardianOS simulator use a PCI bus to connect the differents IP's, and each of them are represented as a PCI device.

To adapt the work that has been done in *TrustSoC*, each IP contains a wrapper that is in charge of intercepting ingoing requests to the IP. The wrapper extract the initiator id from the memory transaction attributes (i.e a thread or another IP) and send a request to the component called "security oracle" that is in charge of storing the access rules for the differents IP's. This component contains a table of rules in the form `<source, destination, right>`. The source can be a thread id or an IP id, and the destination is an IP id. The security oracle needs to be configured by the security monitor. When the security monitor starts, it maps the base address of the security oracle in his VSpace and fill the table with rules. This process is done with the help of a minimal PCI driver created for GuardianOS that operates in the user space. It can discover PCI devices, map BAR's on a given VSpace and perform read and write operations at a givent offset. The overall security is still guaranteed with this approach as a BAR of an IP can't be mapped in the VSpace of an untrusted

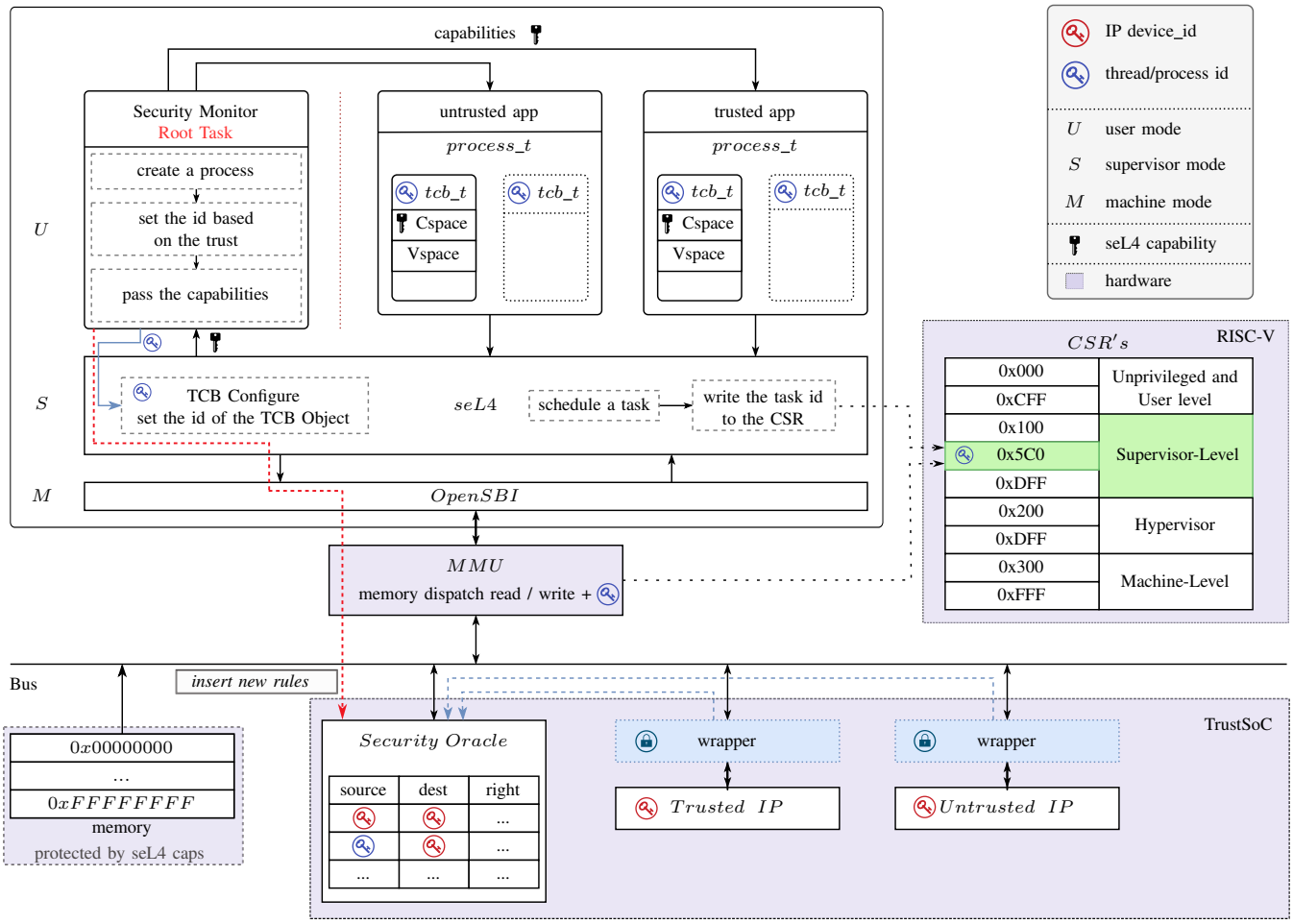

Fig. 1. GuardianOS architecture

application as every memory mapping is done by the security monitor. The security monitor can chose to give the access right to an IP to a thread by performing a `copy` or a `mint` (i.e copy a capability with modified right) operation on the capability for the address of the PCI device and set it in the thread CSpace. Then, each wrapper's IP will check the id of the originating component and send a request to the security oracle to check if there is a matching rule for the association `source:destination`. If the rule is found, the request is forwarded to the IP. If not, the request is dropped.

The physical address of the security oracle BAR0 is mapped in the security monitor VSpace. This means that the security monitor can insert, update or delete rules in the security oracle.

IV. Evaluation and discussion

The particularity of the seL4 microkernel is that it has been formally verified for its functional correctness, which means that the kernel has been mathematically proven to be free of unintended behavior. Especially, these proofs guarantee that any attempt to access a kernel object without a valid capability will result in a *cap fault*. This property is essential for the GuardianOS environnements creation.

As it is shown in the figure 1, a thread id is setted directly in the seL4 kernel. This step could be done directly in the security monitor, but it would decrease the overall security as the CSR whould need to be accessible in user mode. Therefore, the first test that has been done for this project was the re-verification of the existing seL4 proofs to see if the modifications breaks these proofs. No proof has been broken which means that the modifications does not affect the functional correctness of the kernel. The second step was to pass the tests base of seL4 that test every feature of the kernel, and then to add extra tests to check the correct behaviour of our security monitor. The tests has been done using the `seL4test` test suite that is provided with the seL4 sources. As every thread creation now require an extra parameter that is the id of the thread, the tests has been modified to configure the differents threads used in the test suite to include an id. All of the tests has passed. The seL4 proofs can be also re-verified. These proofs are tested with the `isabelle` which is an interactive theorem prover. The patched version of seL4 has been tested with the proofs and no proof has been broken. Therefore, a lot of tests cases are covered by the seL4 proofs themselves. For instance, there is no need to add extra tests to check that a thread can't modify

his own id as the proofs already guarantee this property with the capability security model.

To test the correctness of the accesses verification based on a thread or IP id, a set of different scenarios has been tested.

The next step of the evaluation process was to test the correctness of the IP's accesses restriction. To evaluate the result, the simulator has been populated with two PCI devices with ids `0xcafe` and `0xbeef`. These id's are assumed to be physically printed in the wrappers of the PCI devices. Then, the security monitor has been configured to fill the security oracle with rules to tests differents scenarios that represent the interactions between these two IP's. For instance, the security oracle has been filled with these two rules :

- `0xcafe`→ `0xbeef` : `PERMISSION_NONE`
- `0xbeef`→ `0xcafe` : `PERMISSION_FULL`

Scenario	Expected result	Observed result
`0xcafe`→ `0xbeef`	request blocked	request blocked
`0xbeef`→ `0xcafe`	request forwarded	request forwarded

TABLE III
TEST SCENARIO EXAMPLE

The configuration of the security oracle has been adapted to tests every possible scenarios :

- Empty security oracle : every request is blocked
- Full security oracle (limit of 256 rules) : every request is blocked

The objectives mentionned in the section I-B has been achieved. We can see that the GuardianOS architecture provide the ability to create differents trusted execution environements to run each application individually with personalized permissions thanks to the seL4 capabilities. However, the simulated built with QEMU does not reflect the real architecture for the IP's side. In this simulator, the IP's are represented as PCI devices and each wrappers for these IP's are placed directly in the PCI devices codes. For a real SoC, these wrappers should be placed between the IP's and the bus. An IP internal content should not be trusted at all.

V. CONCLUSION AND PERSPECTIVES

The GuardianOS SoC architecture contributes to the existing litterature by providing a solution that address some limitations that can be observed in other solutions mentionned in the section I-A. The main contribution of GuardianOS is the ability to create different and fully customized trusted execution environments. Some use cases would require to isolate each trusted application from each others for security reason and GuardianOS provide this feature. The seL4 microkernel is a promising solution for the TEE's creation as the formal proofs guarantee that an application can't behave in his own way without a explicit right (i.e capability) delivered by the security monitor.

We have seen that these proofs has been re-verified to test that the differents modifications added to the sources does not breaks these proofs. However, the complete evaluation of this patched version of the seL4 is still in progress as extra formal proof are needed to proof that the id's configuration and propagation by the kernel is correct. Currently, GuardianOS does not comes with a VMM (Virtual Machine Monitor) to run multiple trusted or untrusted applications inside guest OS. seL4 support this feature but it would require to write a specific VMM for GuardianOS.

References

[1] Ferdinand Brasser et al. "SANCTUARY: ARMing TrustZone with User-space Enclaves". en. In: *Proceedings 2019 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2019. ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23448. URL: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_01A-1_Brasser_paper.pdf (visited on 12/02/2024).

[2] Victor Costan, Ilia Lebedev, and Srinivas Devadas. *Sanctum: Minimal Hardware Extensions for Strong Software Isolation*. Publication info: Published elsewhere. Minor revision. USENIX Security Symposium 2016. 2015. URL: https://eprint.iacr.org/2015/564 (visited on 12/16/2024).

[3] Gernot Heiser. "The seL4 Microkernel – An Introduction". en. In: ().

[4] Karthik B. K. *RISC-V Memory Protection: Diving Deep into the Complexities*. en. Nov. 2023. URL: https://medium.com/@talktokarthikbk/risc-v-memory-protection-diving-deep-into-the-complexities-9d751212be6b (visited on 11/12/2024).

[5] Dayeol Lee et al. *Keystone: An Open Framework for Architecting TEEs*. arXiv:1907.10119 [cs]. Sept. 2019. DOI: 10.48550/arXiv.1907.10119. URL: http://arxiv.org/abs/1907.10119 (visited on 12/08/2024).

[6] Jämes Ménétrey et al. *Attestation Mechanisms for Trusted Execution Environments Demystified*. arXiv:2206.03780. Sept. 2022. URL: http://arxiv.org/abs/2206.03780 (visited on 11/14/2024).

[7] Raphaele Milan et al. "TrustSoC : Architecture SoC hétérogène légère et efficace sécurisée par conception". In: *Conférence francophone d'informatique en Parallélisme, Architecture et Système (COMPAS)*. Annecy, France: LISTIC - Laboratoire d'Informatique, Systèmes, Traitement de l'Information et de la Connaissance, July 2023. URL: https://hal.science/hal-04213598 (visited on 12/08/2024).

[8] Pascal Nasahl et al. "HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment". In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. ASIA CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 187–199. ISBN: 978-1-4503-8287-8. DOI: 10.1145/3433210.3453112. URL: https://doi.org/10.1145/3433210.3453112 (visited on 11/04/2024).

[9] Sandro Pinto and Nuno Santos. "Demystifying Arm TrustZone: A Comprehensive Survey". en. In: *ACM Computing Surveys* 51.6 (Nov. 2019), pp. 1–36. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3291047. URL: https://dl.acm.org/doi/10.1145/3291047 (visited on 11/19/2024).

[10] Moritz Schneider et al. *SoK: Hardware-supported Trusted Execution Environments*. arXiv:2205.12742. May 2022. DOI: 10.48550. URL: http://arxiv.org/abs/2205.12742 (visited on 11/24/2024).

[11] *System-On-Chip (Soc) Market Size, Analysis & Growth By 2032*. en. URL: https://www.databridgemarketresearch.com/reports/global-system-on-chip-soc-market (visited on 12/09/2024).