An abstraction model and a comparative analysis of Intel and ARM hardware isolation mechanisms

Guillaume Averlant, Benoît Morgan, Eric Alata, Vincent Nicomette, Mohamed Kaâniche

An abstraction model and a comparative analysis of Intel and ARM hardware isolation mechanisms

Guillaume Averlant, Benoît Morgan, Éric Alata, Vincent Nicomette and Mohamed Kaâniche
LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France
Email: firstname.lastname@laas.fr

***Abstract*—Computer systems software and hardware architectures have become increasingly complex today. Meanwhile, cyber-attacks are becoming more and more sophisticated and target any software or hardware components of these systems. Several isolation mechanisms, at the software and the hardware layers, are now available to provide the best protection against these widespread attacks. This paper is aimed at reviewing especially hardware segregation mechanisms available in today's CPU in order to provide better insights about the intended scope of the protection and the different threats that could be addressed by such mechanisms. An abstraction model presenting the main components of current architectures and their interactions through different communication channels is proposed to support such analysis. The study focuses on Intel and ARM architectures, and outlines various hardware isolation resources that provide a security layer to the software running on these architectures. A comparative analysis of these architectures is also presented together with a discussion of open issues and future challenges.**

INTRODUCTION

Nowadays, the security of recent computer systems is challenged by their architectural complexity. This complexity, resulting from the increasing number of features supported, makes it unlikely to design and implement a system without security vulnerabilities. Attacks against these systems are more and more diversified. They target both the software layers at all privilege levels as well as the hardware layers.

On this basis, each component can potentially be corrupted by an attacker, and can affect the overall system integrity. To limit the impact and propagation of such attacks, it is necessary to isolate the different components from each other. Moreover, considering that more and more machines are deployed in Cloud environments, hardware resources are increasingly shared among different users who may belong to different companies. In this context, it is essential to ensure data confidentiality by properly isolating their environment. Finally, the service provided by the system must be available. For this reason, the operating system should ensure the fair distribution of hardware resources between several tasks in order to prevent abuse of resources or potential denial of service. Isolation mechanisms also contribute to this fair distribution.

Currently, many manufacturers have already deployed isolation mechanisms in their systems. Due to the lack of standardization effort, they have therefore set up different implementations that meet the same segregation requirements. An open question that can be raised is related to the level of protection offered by these implementations to provide an adequate isolation and ensure integrity, confidentiality and availability requirements of the entire system. To answer this question, it is necessary to precisely understand these mechanisms, to analyze their scope and the associated threats that they are able to address, and also to determine to what extent they are different from each other.

A few analyses of Intel and ARM architectures and of the associated isolation mechanisms have been published in the literature (e.g., [19], [5], [17]). The research reported in [19] focuses on software architecture isolation techniques and briefly describes related hardware supported mechanisms. A detailed presentation of hardware security mechanisms is provided in [5] for Intel architectures and, in a similar way, in [17] for ARM architectures. However, to our knowledge, none of the existing studies presents a comparative analysis of security related hardware isolation techniques provided in these platforms, considering in particular technological advances proposed recently. This corresponds to the objective and to the main contribution of this paper. Additionally, this paper is solely dedicated to hardware isolation mechanisms that involve segregation by partitioning, whereas other technologies, like Intel SGX [5], provide isolation by encryption.

More precisely, we present a generic model of a computer hardware architecture, able to represent the different components of this architecture. This model enables us to highlight the existing communication channels between the different components, to determine the various threats that may target the modeled system, and also, to perform a comparative analysis of the isolation mechanisms provided by the Intel and ARM architectures. Finally, we identify the existing hardware isolation mechanisms that can be used in order to mitigate these threats. The proposed model is instantiated both for Intel 64 and ARM ARMv8-A platforms, which allows us to present and compare the hardware isolation mechanisms of these two platforms. The paper is organized as follows. Section I is dedicated to the presentation of the model while Section II reviews the associated threats. Then, Section III presents the isolation mechanisms able to mitigate the various attacks and finally, Section IV concludes and presents future works.

I. MODELING

In order to identify key locations suitable for setting up security-related isolation mechanisms, this section presents a generic model of current computer architectures that is designed to highlight the role of each hardware component and their interconnections. This model is aimed at providing

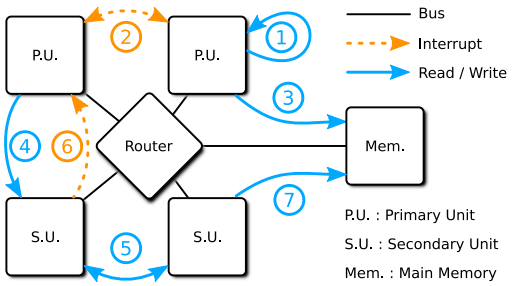

Fig. 1. Communications between the system units

a simple abstraction of these architectures to discuss potential threats and existing isolation mechanisms. Accordingly, an instance of this model is generated for each studied architecture (Intel, ARM), to check its ability to consistently represent the elements of these architectures and their communications.

A. The model

The different components of current hardware architectures can be classified into four distinct categories (Figure 1)).

Each machine has one or multiple **primary execution units** dedicated to general purpose and arithmetic operations, to communicate with the other components and to manage the hardware. These units have a privileged and central role in the system architecture.

Secondary execution units are dedicated to a specific type of service in order to provide this service more efficiently than primary execution units. Sometimes they also perform operations which cannot be supplied by primary execution units (e.g., data storage, input/output, user interface...).

A third important element, **the memory**, shared between primary and secondary units under certain conditions, offers a wide and rather fast storage area, accessible in read and write access, to host the code and data used by the various units.

Finally, the last type of component, **the router**, is a special secondary unit used to interconnect several elements of the system by implementing the needed communication protocols. For the sake of simplicity, segregation issues associated with the router are addressed the same way as other secondary units.

Let us note that there are two fundamental differences between the two types of execution units. Each primary unit always has a public interrupt controller that can be used by other primary and secondary units. Moreover, only a primary unit is able to configure the router interconnection channels.

B. Communications within the model

The previously identified components are interconnected by buses. Communications can be described as accesses to resources. They are classified into two subsets:

- **Reads/writes of data** in the memory, in secondary units external registers or in primary units internal registers.
- **Signals transmission**, for interrupting a primary unit to notify it of an event.

We can then list the interactions between the components of our model as illustrated in Figure 1[1]:

∗-1) A **primary unit** is able to read and write to its own internal registers in order to store data or to configure itself.

∗-2) Since the primary units are running independently of the system, they must be able to be alerted upon the occurrence of an important event. In particular, a **primary unit** can interrupt the other **primary units** through signals. They are also able to send signals to themselves (e.g., timers...).

∗-3) A **primary unit** can read and write multiple data from/to **memory**. For example, these data can be used by programs running on this primary unit. The code of these programs is also read from memory.

∗-4) A **primary unit** can also communicate with a **secondary unit** in order to send the parameters of an operation to be performed or to retrieve the result of an operation. This communication is done through reading/writing accessible registers of the secondary unit.

∗-5) In some cases, a **secondary unit** may be allowed to communicate directly with other **secondary units** (peer-to-peer communication). Thus, the resulting read/write accesses are not checked by any primary unit.

∗-6) Similarly to the communication channel n°2, a **secondary unit** must be able to notify any **primary unit** upon occurrence of an event related to its specific function. For example, a secondary unit can send a signal at the completion of an operation.

∗-7) Most secondary units have their own internal memory. However the latter has often a low capacity and is only accessible from the unit itself. Therefore, in order to benefit of more memory or to share data with other units (primary or secondary), a **secondary unit** should be able to read and write to system **memory**.

The communication channels between components being identified, the next step is to generate instances of the studied architectures (Intel and ARM) in order to understand their specific characteristics. For the sake of consistency, the enumerated lists describing the communications for each implementation refer to the numbering of Figure 1.

C. Intel

In this section, we apply our model to the Intel 64 architecture and more precisely to the Intel core 4th generation Haswell machines, associated to the Intel C220 chipset. We have explicitly chosen this platform because we needed to actually test some features which are not well documented. However, as all Intel 64 platforms are very similar, our work is applicable to the whole Intel 64 family.

For Haswell processors, a ”processor” is actually composed of four primary execution units, some secondary units like the graphics processor, but also the first router *Router 1* named *System Agent*. This router, formerly named *north bridge*

[1]For the sake of clarity, all communications of each PU and each SU are not represented. They are spread over both instances of PU and SU.

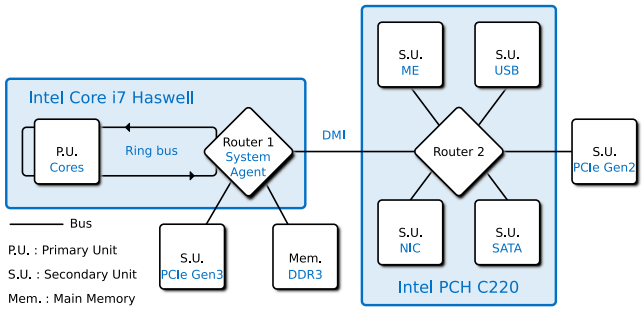

Fig. 2. Model instantiation on Intel Haswell CPU and Intel C220 PCH

and then *Memory Controller Hub* (MCH), was physically integrated into the processor since Intel Nehalem architecture. It enables the communication between primary units through the *ring bus*, and also with the memory, the integrated video processor, the high performance PCI Express devices, and any other secondary units. These communications are supported by the *Direct Media Interface* (DMI) bus, connecting the processor to the second router, the *Platform Controller Hub* (PCH) (Figure 2).

The Intel C220 PCH router enables the communication with the Router 1 by converting DMI messages to and from other buses, registers, or protocols such as PCI Express. It also includes other secondary units such as a gigabit Ethernet controller, a Serial ATA controller, an USB controller, and a specific co-processor, the *Management Engine* (ME) with its firmware. The *Management Engine* enables the remote management of machines, without interacting with the operating system hosted by the processor, and potentially within degraded power management modes.

Finally, it is possible to extend the number of secondary units via the PCI Express protocol. The C220 chipset offers eight PCI Express 2 root ports enabling the connection of additional secondary units, usually through expansion slots.

Intel PCH chipsets have also a specific characteristic. They are shipped with an analog display controller (VGA) that performs the signal modulation for the display from the video memory (*framebuffer*). However, this buffer is provided by the graphics processor itself, through a specific bus, the *Flexible Display Interface* (FDI) [8, Vol. 2 / 1.2.1, 5.28.3, Vol. 1 / 2.7]. To simplify the model and because of lack of information about this interface, the FDI bus is not included in our analysis.

Due to the backward compatibility requirement enforced by Intel, the studied architecture is highly configurable, supporting deprecated operating modes which are no longer used at runtime. Moreover, some parts of the architecture that are not meant to be configured by software are poorly, or not, documented. This requires to deduce the characteristics of some components and communication buses. This concerns in particular the interrupts management between execution units, especially in the PCH. As shown in Figure 3, a secondary unit like the SATA controller has multiple ways to interrupt the execution of a core [12, 3.6.2]:

- In the standard operating mode, it is possible to directly transmit a PCI Express interrupt named *Message Signaled Interrupt* (MSI). These messages are first sent to the local interrupt controller, the *local Advanced Programmable Interrupt Controller* (APIC) of one or more cores through memory writes. They are then transmitted to the target cores via the DMI bus.
- It can also perform an interrupt request to the historical interrupt controller, the *Programmable Interrupt Controller 8259* (8259 PIC), or use the newer *I/O Advanced Programmable Interrupt Controller* (I/O APIC) which can handle more interrupts. Once the interrupt request is accepted by the 8259 PIC or the I/O APIC, a bridge across these components emits a PCI Express write on the DMI bus targeting the core to be stopped [8, 5.10.1].

The management of an interrupt generated from a secondary unit depends on the configuration of the PCH and of the unit itself. Note that the I/O APIC is also able to operate in cascade with the 8259 PIC. Once the interrupt is sent through the DMI bus, it is first processed by the *System Agent*, and then sent on the ring bus to interrupt the target core. Inter-core interrupts are similarly handled by the system agent.

To sum up, the interrupts management outside the processor is fairly complex but each case leads to the same type of PCI Express writing messages. For this reason, we will only consider MSI interrupts and, if necessary, interrupts generated from the I/O APIC .

With this in mind, we detail in the following all types of communication related to this specific architecture:

Intel-1) In the Intel 64 architecture, each primary unit has several general purpose registers for data manipulation and special control registers to configure its behavior.

Intel-2) The communication between the multiple execution cores of the studied architecture is made possible by interrupts. These interrupts are handled by the local APICs of each core, and are sent or received through the ring bus. They can be initiated by an execution core by writing in the memory or in an internal register.

Intel-3) Data in main memory are accessed from a CPU core by initiating DDR3 read and write commands. These

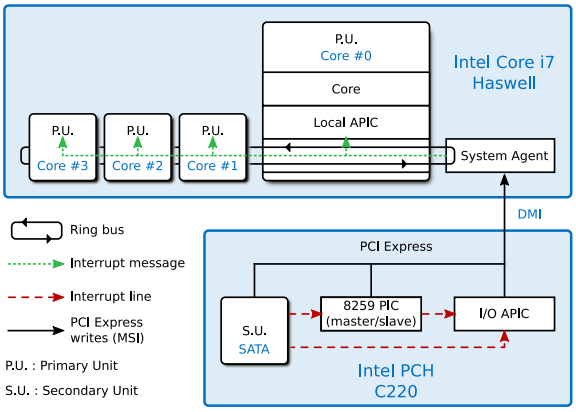

Fig. 3. Internal mechanisms for interrupts management

commands are created by the *Integrated Memory Controller* (IMC) included in each primary unit, and are then transmitted by the system agent to the main memory.

Intel-4) Primary units memory accesses to secondary units are processed in various ways depending on the destination: i) Accesses to the integrated graphics processor are directly transmitted by the ring bus; ii) Accesses to local PCI Express secondary units are converted to PCI Express messages by the system agent; iii) accesses to secondary units interconnected by the PCH are converted to DMI messages, and then to PCI Express messages if required.

Intel-5) Direct communications between Intel secondary units integrated into the PCH are poorly documented. Nevertheless, it is possible to study the case of PCI Express devices. Due to the star topology of networks supported by the processor and the PCH, PCI Express messages must pass through one of the routers. Generally, direct peer-to-peer memory accesses are not allowed. If they are necessary, they must be performed by a primary unit.

Intel-6) Secondary units must be able to notify a primary unit upon the completion of a requested operation, or at the occurrence of an event. To do this, they can generate an interrupt request. Depending on the source secondary unit, this request is received from the DMI bus or converted from the local PCI Express end point of the CPU. The requests are finally transmitted to the local APIC of one primary unit via the ring bus.

Intel-7) There are two types of memory accesses from secondary units towards the main memory: those coming from local secondary units linked directly to the system agent, and those coming from the PCH through the DMI bus. In both cases, they are converted into DDR3 memory accesses by the local IMC included in the system agent.

D. ARM

ARM is primarily a seller of *Intellectual Property* (IP), which is then integrated by manufacturers in their systems. This technology is included in various devices, e.g., smartphones, tablets, and some low-power servers. We have instantiated our model on a general implementation of an ARM architecture in order to cover different use cases of this architecture (Figure 4). In particular, we focus on an ARMv8-A[2] compatible architecture.

Unlike the Intel implementation, all primary units, routers and secondary units are on a single *System On Chip* (SOC). Communication between SOC components is performed via *Advanced Microcontroller Bus Architecture* (AMBA) buses.

The CPU cores (primary units) are grouped into clusters, with a maximum of four cores per cluster. The latter includes a L2 cache shared between these cores. Moreover, a single interrupt controller, named *Generic Interrupt Controller* (GIC), is shared between all primary units. Each primary unit has a direct access to the GIC's internal registers.

[2]The profile "A" is optimized for application use, whereas profiles "M" and "R" are intended for micro-controller and real-time requirements respectively.

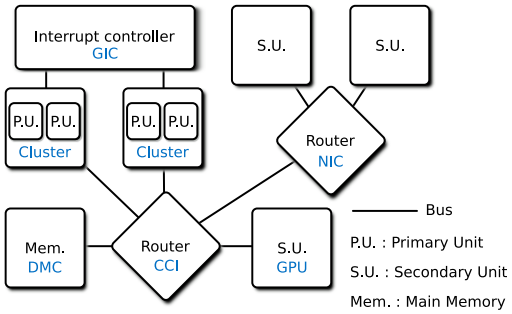

Fig. 4. Model instantiation on a generic ARMv8-A architecture

The primary units are connected to the rest of the system by a first router. This router, named *Cache Coherent Interconnect* (CCI), physically maintains the cache consistency between each interconnected element and the main memory. However, the CCI router has a limited number of interconnections, and it is not able to interface all types of AMBA buses. So, the other secondary units must be connected through a new router, named *Network InterConnect* (NIC), that supports all buses of the AMBA standard. The number of interconnections can be extended depending on the system requirements.

Regarding the secondary units, we typically have a GPU directly connected to the CCI router. Furthermore, when a PCI Express root port is provided on the SOC, it is also connected to this same router. The other secondary units mainly consist of USB, UART, I2C, SATA and SD controllers used by external devices to interact with the SOC.

Let us now analyze the communications between the various components of the ARM architecture:

ARM-1) CPU cores have several types of internal registers. A set of general purpose registers is dedicated to store data being processed. But there are also system registers that aim at configuring the behavior of these cores.

ARM-2) As the primary units are connected to the same interrupt controller, an execution core can generate a *Software Generated Interrupt* (SGI) to one or more cores within or outside the same cluster. An SGI can thus be triggered by writing into a specific GIC register.

ARM-3) The read/write requests issued by a CPU core are relayed by the cache coherency router to the *Dynamic Memory Controller* (DMC). If a memory access to the same address was previously carried out by another primary unit, the CCI has a specific behavior to optimize memory fetch performances. For a read access, the previous cached data retrieved from the last access to this address is returned. For memory writes, the related cache lines are invalidated on each unit connected with the CCI.

ARM-4) On ARM architectures all secondary units accessible registers, are mapped into memory. Read/write requests are thus routed to the target secondary unit by the routers through the dedicated AMBA communication buses.

ARM-5) In the same way that secondary units have the ability to access the main memory, they can reach any

register exposed by other secondary units. Peer-to-peer communication is thus allowed.

ARM-6) There are two ways for secondary units to send an interrupt signal to one or more primary units: i) if an interrupt line is directly connected to the interrupt controller, the targeted core is determined by the GIC configuration; ii) if the secondary unit is DMA capable, it can use the GIC memory-mapped registers to generate the interrupt, and the targeted core is determined by the secondary unit or the GIC depending on the interrupt type.

ARM-7) Secondary units have two ways to perform accesses to the main memory. If they are DMA capable, they can perform read/write requests by themselves, just like primary units. Otherwise they have to interrupt a primary unit, so that this unit could perform the memory access for them, or configure a DMA controller to carry out this access. The cache consistency is maintained by the CCI upon DMA accesses.

Both Intel and ARM architectures, as abstracted with our model, are similar in many aspects, with some small differences due to their different aim, philosophy and evolution. First the Intel architecture is split into two parts (the CPU and the PCH), whereas on ARM the whole system is implemented on the same SoC. The communication buses are also fairly different. ARM uses only AMBA buses, primarily designed for low consumption purpose but also extended to meet high performance requirements, while Intel favors PCIe and proprietary buses. Lastly, as Intel enforces backward compatibility, its interrupt scheme is more complex compared to ARM.

Through these two implementations, we have shown that our model is relevant to properly represent the target systems of our study. In the next section, we provide a detailed analysis of the communications between each component of our model. This analysis is a necessary step to identify possible threats targeting the studied systems.

II. Access control & Threats

To identify the security-related threats based on the analysis of the communication channels described in our generic architecture model, we need to represent in the model the access control mechanisms aiming at authorizing or denying the corresponding communications. We define the notion of context that allows us to specify the necessary and sufficient rules for an execution unit to access a given resource (Figure 5). The implementation of access controls related to these contexts is performed in two locations:

- In primary units for accesses carried out by software running on primary units.
- In routers for accesses carried out by primary and secondary units.

A context is associated with every unit of the system. It consists of the combination: i) the set of all accessible resources of the unit; and ii) its current privilege level. This context may change over time. Moreover, a privilege level is assigned to every resource. Therefore, each resource is only accessible by units having a higher privilege level.

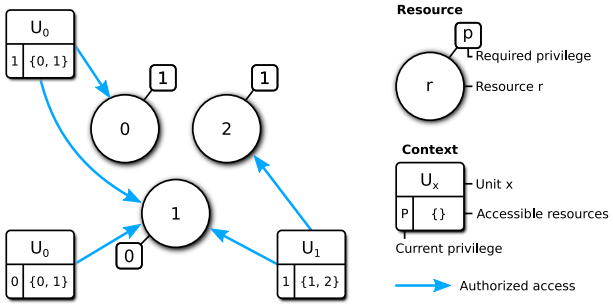

Fig. 5. Context restricting resource access of several units

These rules can be summarized as follows: Let $C = (p_u, \mathcal{R})$ be the current context of a unit U, with $p_u \in \mathbb{N}$ its current privilege and $\mathcal{R}$ the set of its accessible resources. Let r be a resource with $p_r \in \mathbb{N}$ its privilege. We have:

$$U \xrightarrow[\text{allowed}]{\text{access}} r \Leftrightarrow r \in \mathcal{R} \text{ and } p_u \geq p_r \quad (1)$$

A. Attack assumptions

We focus on the atomic accesses of system units to resources (read/write accesses and interrupts). Indeed, hardware isolation mechanisms have been designed to provide the basic building blocks to system designers to create atomic access control mechanisms. Each of these mechanisms taken alone is probably not sufficient to deal with complex attack scenarios, which involve the collaboration of several entities and the execution of several atomic operations. Addressing such attack scenarios requires dealing with an additional complexity, partly due to the "semantic gap". However, these scenarios necessarily consist of several elementary accesses. Therefore, appropriate countermeasures to mitigate such attacks will necessarily rely on one or several isolation mechanisms outlined later in this paper.

We initially assume that the studied platforms are not malicious, i.e. the primary units, buses and the main memory are not malevolent. However, we consider that an attacker can gain control of three different system components:

- A piece of software running on a CPU core at any privilege level.
- A piece of software part of a virtual machine.
- A secondary unit.

B. Threats

Malicious hardware and software try to circumvent the resource access controls. These violations correspond to the non-existing arcs on the resource access graph (Figure 5) and can be classified into two types:

1) Access to resources that should not be accessible ($r \notin \mathcal{R}$)
2) The privilege level required is not respected ($p_u \ngeq p_r$)

For every communication in the system (summarized in Table I), there are two threats matching these two types of access violations. These threats are numbered in the form $\boldsymbol{C.V}$, $\boldsymbol{C}$ identifies the communication, $\boldsymbol{V}$ the violation type.

TABLE I - TABLE SUMMARIZING THE COMMUNICATIONS

N°	Unit	Resource
1	P.U.	reading or writing an internal P.U. register
2	P.U.	transmitting a signal to a P.U.
3	P.U.	reading or writing a memory area
4	P.U.	reading or writing a S.U. register
5	S.U.	reading or writing a S.U. register
6	S.U.	transmitting a signal to a P.U.
7	S.U.	reading or writing a memory area

For example, **3.2** corresponds to an unauthorized access to a memory access by a primary unit which does not possess the required privilege level. A user process accessing a memory area dedicated to the kernel is an example of such a threat.

Similarly, the threat **7.1** identifies a secondary unit which is able to access a memory area that is not supposed to be accessible by this unit. Writing data from a secondary unit to the memory area containing the kernel code is an attack related to this kind of threat.

Finally, the threat **3.1** refers to a prohibited memory access by a primary unit to a memory area that does not belong to its context resource set. An example, is an attack by a virtual machine targeting a memory area owned by another virtual machine running on the same system.

In order to cope with these threats, many manufacturers have introduced hardware isolation mechanisms within their systems. This is the case for Intel and ARM whose solutions are presented and compared in the following section.

III. EXISTING ISOLATION MECHANISMS

The isolation technologies existing in both studied architectures can be described as access control mecanisms that cope the threats identified in the preceding section. In the following, we highlight some isolation mechanisms addressing the identified threats. The numbers in bold and in brackets ($\boldsymbol{C.V}$) refer to the threats defined in Section II which are addressed by the described techniques. Then a comparison analysis is performed at the end of this section.

A. Intel

The Intel Core Haswell architecture associated with the Intel C220 PCH implements a significant number of access controls to ensure the system security. We focus on the essential ones, and especially on those visible from the perspective of software developers.

1) The primary units: The primary units, or cores, have various operating modes which widely make use of the concept of contexts by restricting the available memory areas and registers. The Haswell processors cores implement the Intel 64 architecture which offers the 64-bit execution mode currently used at runtime. However, for backward compatibility reasons, they support a multitude of older and less efficient modes in which the cores start before being reconfigured into 64-bit mode. They are presented in the following according to their historical chronological order, which also corresponds to their activation order at startup.

The primary units start in the historical mode called *real mode*. It provides the runtime environment used by the first software developed for the IBM PC, which contained an Intel 8088 CPU. This mode is today no longer used and does not have any access control mechanism. That is why the core is quickly reconfigured into the protected mode.

The *protected mode*, brought by Intel 80286 and extended by the 80386 [7, Vol. 1 / 3.1.1], is much more suited to isolation purposes. It provides the segmentation and pagination mechanisms that enable virtualization of the CPU core address space, with two translation phases performed by the *Memory Management Unit* (MMU). Software handles logical addresses, translated first into linear addresses by segmentation and then into physical addresses with the tree structures of pagination (**1.1, 2.1, 3.1, 4.1**). Segmentation mostly provides the management of privilege levels called *rings* which affect, among others, instructions and registers accessible by software (**1.2, 2.2**). The address translation process also allows to associate a required privilege level to a memory segment or a page (**1.2, 2.2, 3.2, 4.2**). With segmentation and pagination, it is thus possible to run more or less privileged software on the primary unit. Each of them embodies the notion of contexts that we have defined earlier. A compatibility mode, called *virtual-8086 mode*, that allows to run 16-bit code in a protected environment, is also available.

The *IA-32e* mode is the latest mode added by Intel. It consists of two modes of operation [7, Vol. 1 / 3.1.1], the 32 or 16-bit compatibility mode and the 64-bit mode. The 64-bit mode removes the address translation provided by segmentation, considered obsolete, in favor of the pagination which is much more accurate. However, this new mode doesn't provide new isolation mechanisms.

With the massive introduction of virtualization, Intel proposed an extension of Intel 64 which enables to enhance the performance of CPU cores virtualization. This new extension, named *Intel VT-x* [7, Vol. 3 / 23.1-3], brings a new execution mode and a dedicated instruction set, to serve this purpose. Indeed, a CPU core can enter in *VMX operation* mode in which a guest software is executed under the supervision of a privileged software, the hypervisor (or *Virtual Machine Monitor* (VMM)). The latter is thus executed under the *VMX root operation* mode and benefits from the new VMX instruction set. Therefore, the guest software, running in *VMX non-root operation* mode, has a restricted access to privileged actions, such as using some specific registers (**1.*, 2.2, 4.2**). Furthermore, an extra virtualization level of the memory, controlled by the hypervisor, is added. This new translation mechanism, named *Extended Page Tables* (EPT), is very similar to the pagination of the IA-32e mode and simplifies the address space management of virtual machines (**2.*, 3.*, 4.***). The incoming and outgoing signals of CPU cores are controlled as well by the hypervisor through the local APIC virtualization (**2.2, 6.1**).

A transversal execution mode (*management mode*), has been also integrated since the Intel386 SL processor [7, Vol. 1 / 3.1]. It offers a privileged execution environment to perform specific management operations on a given platform (power manage-

TABLE II - SUMMARY OF INTEL CONTROL MECHANISMS

Threat origin	How	Control mechanism	**1.1**	**1.2**	**2.1**	**2.2**	**3.1**	**3.2**	**4.1**	**4.2**	**5.1**	**5.2**	**6.1**	**6.2**	**7.1**	**7.2**
Primary Unit without virtualization	R/W Reg.	Rings		√		√										
		SMM	√													
		TXT	√		√											
	R/W Mem.	MMU	√	√	√	√	√	√	√	√						
		SMM	√		√		√									
		Router 1					√									
		TXT	√		√				√							
Primary Unit with virtualization	R/W Reg.	Virtual core	√	√												
		Virtual APIC				√										
	R/W Mem.	Virtual core								√						
		Virtual APIC				√										
		Virtual MMU	√	√	√	√	√	√	√	√						
Secondary Unit	R/W Mem.	Router 1									√				√	
		Router 2									√					
		I/O MMU									√	√	√	√	√	√
		TXT													√	
		Virtual Apic & SMM											√			

ment, hardware configuration, etc.). This mode, designed to host some of the system firmware code, uses another address space separated from other modes, named SMRAM (**3.1**), as well as full access to machine resources. The only event that could toggle a primary unit in this mode is the reception of a *System Management Interrupt* (SMI) signal (**2.1, 6.1**).

2) The first router: Some above-mentioned access control mechanisms require the collaboration of the CPU cores with the Router 1. Indeed, physical addresses handled by primary units may not represent an address of a DRAM memory block (*bank, row, column*). Memory accesses are translated by the Router 1, or *System Agent*, and then routed to the appropriate memory block or secondary unit. For example, accesses to the segment `[0xa0000; 0xbffff]` are dedicated to the *Legacy Video Area*. However, if the CPU core is in SMM mode, the access is routed to the SMRAM. This verification, performed as near as possible to the cores helps to ensure the protection against SMM mode attacks by cache poisoning [6]. A DRAM area is furthermore reserved for the *management engine* co-processor for security reasons. Therefore, accesses from the primary units are restricted by the *System Agent* as well [11, Vol. 2 / 2.3, 2.5.2] (**3.1**).

Router 1 is located at the center of the architecture in terms of memory access and signal transfer. It is also responsible for controlling secondary units accesses to the physical memory space. As we mentioned in Section II, some of the memory read or write accesses are routed towards local or remote secondary units (via the DMI bus). This is the case in the opposite direction as well. Indeed, secondary units can access a certain part of the DRAM but are unable to access the SMRAM, the *management engine*, and the DMA protected segments, etc. [11, Vol. 2 / 2.5, 2.5.2, 2.15, 1.16] (**7.1**).

Each secondary unit can be used in different contexts, which raises security issues. That is why Intel introduced a new kind of component, named I/O MMU, specified by the Intel VT-d extensions [10]. This component is able to perform more specific access controls on read/write accesses and on signals from the secondary units, while taking into account their associated contexts. These controls are respectively named *DMA remapping* (**5.*, 7.***) and *IRQ remapping* (**6.***). In our architecture, an I/O MMU is used for local secondary units and another one for remote secondary units of the PCH [11, Vol. 2 / 4.4, 4.5]. Therefore, several secondary units are controlled by the same I/O MMU. The address translation phase is divided into two stages [10, 3.2]. The first one is used to identify the related secondary unit in order to retrieve its vision of memory [10, 3.4]; this process is called *device to domain mapping*. The second step translates virtual addresses into physical addresses by using the memory vision of the device; this process is similar to the one performed by the paging mechanism of a CPU core. Additionally, address translation is divided into two phases, similarly to MMU virtualization, to enable *DMA remapping* at virtual machines level; the first translation phase structures are controlled by a virtual machine, and those of the second phase are configured by the hypervisor [10, 3.1, 3.5-8]. Therefore the I/O MMU was designed to block attacks originating from secondary units which perform read and write accesses in accessible segments of the main memory [13].

Besides I/O virtualization, I/O sharing is also an issue when running multiple VMs that want to access a same device. The sharing can be done at the hypervisor level using device emulation, but it leads to poor performances. On the other hand, the device can be assigned to only one VM (direct assignment) to improve the performance, but with a negative impact on scalability. However, another technology, named *Single Root I/O Virtualization and Sharing* (SR-IOV) [16], has recently been brought up by Intel for this purpose. A device compatible with this technology provides multiple virtual functions assignable by a VMM to its VMs. These virtual functions leverage Intel VT-d to directly communicate with the VMs without involving the VMM. This way, the sharing can be done without loosing performance, and while maintaining the isolation layer brought by Intel VT-d.

3) The second router: PCH is responsible for interconnecting remote secondary units with the processor through the DMI bus [8, 5.2, 9.4]. As explained earlier, access controls from the DMI bus are performed upstream via the I/O MMU. As seen in Section I, all the communication between internal

secondary units of the PCH, as well as between secondary PCI Express units, are transmitted through the PCH and processed by the *PCI-to-PCI bridge*. This component prohibits memory accesses from secondary units to *DMI decoded* areas, i.e., memory areas where the secondary units registers are mapped [8, 5.2, 9.4] (**5.1**). In our architecture, peer-to-peer memory requests are not allowed between secondary units.

4) Startup and chain of trust: As explained in previous sections, the processor and its primary units start in a degraded mode and then set up progressively a more efficient runtime environment, as well as suitable access controls to enhance system security. This configuration is carried out by software. The firmware, bootloaders, hypervisors and kernels take care of this task. It is therefore important to ensure their integrity. The principle of chain of trust, where each loaded software component gradually checks at startup the integrity of the next one, provides the required trust [6, 3.1]. These verifications are preferably assisted by dedicated hardware such as *Trusted Platform Modules* (TPM) [18] [8, 5.26].

These mechanisms ensure the integrity of the system at startup, but do not protect against the exploitation of software and hardware vulnerabilities at runtime. Malicious operations may for example aim at disabling some hardware access control mechanisms like the I/O MMU [14]. In order to address such threats, Intel proposed new hardware extensions, named *Safer Mode Extensions* (SMX) [7, Vol. 2C / 5], specified by *Trusted Execution Technology* (TXT). With these extensions, after a verified boot, it is possible to protect memory ranges from secondary units and to lock some features and hardware registers to prevent their reconfiguration, even as part of an attack, by entering in a *Measured Launched Environment* (MLE) [9, 1 and 2] (**1.1, 2.1, 4.1, 7.1**).

Table II summarizes the isolation mechanisms present in this Intel architecture and the associated threats. The first two columns identify the source of these threats (as assumed in II-A) and the corresponding initiating action. Grey cells refer to threats that are not applicable in this architecture.

B. ARM

The growing usage diversity of ARM processors has led ARM to incorporate several isolation mechanisms in its architecture as well. This is especially the case for the most recent one, the ARMv8-A architecture, which are included in our smartphones for example. ARM relied on the concepts developed for Intel x86 processors to provide its own isolation mechanisms. However, the approach chosen by ARM is much closer to the operating mode of current operating systems as shown in the following paragraphs. In the following, we present these mechanisms focusing first on those implemented in the primary units, and then on those hosted in the other components of the system. Finally, a specific discussion is dedicated to the *TrustZone* isolation technology.

1) Inside primary units: Four privilege levels are defined, named *Exception Levels* (EL) [1, D1.1], for the software running on the primary units:

- EL0, the lowest privilege level, is devoted to applications.
- EL1 is dedicated to kernel and interrupt management.
- EL2 is devoted to virtual machine managers (VMM).
- EL3 is the most privileged level. Its usage is detailed later on in the section presenting TrustZone technology.

These privilege levels restrict especially the access rights to the system registers of a primary unit (**1.2**); these system registers being in charge of controlling the software execution at various levels on the CPU core. For example, sending an interrupt (SGI) to other system cores (**2.2**) or setting up the MMU require the EL1 privilege level.

The MMU [1, D4.2.1] is also a central component of isolation mechanisms present in the primary unit. It allows to define translation tables converting virtual addresses used by programs running on the CPU core, into physical addresses of the system (**3.1, 4.1**). Moreover, it is possible, by specifying access rights, to restrict the access to these virtual addresses, according to the software privilege level (**3.2, 4.2**). Its behavior is similar to the Intel 64 pagination.

The EL2 privilege level and associated virtualization extensions [1, D1.5] provide isolation within the primary unit. The latter enables to share resources of the CPU core and of the system between multiple virtual machines (VMs) running in EL1. To be more specific, these extensions enable the virtualization of physical CPU cores through the virtualization of their registers (**1.*, 2.1**), of the VMs memory space (**3.*, 4.***) and their incoming interrupts (**2.1, 6.1**). That way, the VMM is able to change its VMs access to some resources (memory access and system registers) by getting back the execution control flow when a VM tries to use them. Memory virtualization is implemented by allowing the hypervisor to define one more translation table that converts VMs physical addresses to real ones. An hypervisors is also able to mask or trigger virtual interrupts to their VMs.

2) Outside primary units: Isolation mechanisms are also implemented outside primary units. For example, the interrupt controller (GIC) is able to mask the interrupts received from primary and secondary units, or to distribute them to one or more physical or virtual CPU cores [3] (**2.1, 6.1**).

Just as primary units have a vision of the memory that can be modified through the MMU, secondary units have a specific entity dedicated to this task. The latter, so-called *System MMU* (SMMU), stands in-between one or more secondary units and their associated router. Each secondary unit connected to a SMMU is identified by a *StreamID*, which is then associated with one or more translation tables of virtual addresses into physical addresses [4, 2.1]. Indeed, a SMMU handles two address translation levels. Its behavior is analogous to the MMU of a primary unit running a VMM. This feature is especially useful when a VMM wants to virtualize the first level of address translation set up by its VM for a secondary unit. The resulting addresses may reference data in memory, registers of secondary units or even registers of the GIC used to emit interrupts to primary units (**5.*, 6.*, 7.***).

3) TrustZone: TrustZone is an isolation technology implemented in several parts of an ARMv8-A system. It enables the definition of two hermetic "worlds": the *normal world* and

TABLE III - SUMMARY OF ARM CONTROL MECHANISMS

Threat origin	How	Control mechanism	**1.1**	**1.2**	**2.1**	**2.2**	**3.1**	**3.2**	**4.1**	**4.2**	**5.1**	**5.2**	**6.1**	**6.2**	**7.1**	**7.2**
Primary Unit without virtualization	R/W Reg.	Privileges (EL)		√		√										
		GIC			√	√										
		TrustZone	√	√	√											
	R/W Mem.	MMU					√	√	√	√						
		TrustZone					√	√	√	√						
Primary Unit with virtualization	R/W Reg.	Virtual Core	√	√	√											
		Virtual Int. + GIC			√											
	R/W Mem.	Virtual MMU					√	√	√	√						
Secondary Unit	R/W Mem.	System MMU									√	√	√	√	√	√
		GIC											√			
		TrustZone									√		√		√	
	Signal	GIC											√			

the *secure world*. The resources belonging to the *secure world* become inaccessible from the *normal world* through a set of isolation mechanisms outlined in the following paragraphs.

First, let us focus on the EL3 privilege level of a primary unit. The most privileged level is actually dedicated to change the security level of a primary unit [1, D1.4]. Indeed, it is the only privilege level able to switch a CPU core from the *normal world* to the *secure world* and vice versa (**1.2**). When a primary unit belongs to the *secure world*, it can use EL1 and EL0[3] privilege levels the same way as in the *normal world*. Therefore, it is possible to create an operating system running in EL1[4], isolated from software belonging to the *normal world*, and able to communicate with resources reserved to the *secure world* (**1.1, 2.1, 3.*, 4.***).

Besides primary units, secondary units may also belong to the *secure world*. Thus, they have the ability to issue "secure" interrupts that will be processed by the GIC [3, 4.6.1] and forwarded to the *secure world* of targeted cores (**6.1**). Furthermore, these secondary units can access memory resources of the *secure world* (**5.1, 7.1**). However, an additional control can be performed by the SMMU [4, 2.2 and 7.3].

Finally a new entity called *TrustZone Address Space Controller* (TZC) [2], manages the segmentation of the memory areas belonging to the *secure world* and the *normal world*. It is also able to check the security level of each memory access to properly isolate the memory areas of the *secure world*.

Therefore, *TrustZone* enables a strict separation between the different hardware parts of the system, as well as software running on the primary units. Consequently, the threats related to the communications between the *normal world* and the *secure world* can be handled by the segregation mechanisms implemented within *TrustZone* technology.

The Table III provides a summary of all segregation mechanisms outlined above.

C. Comparative analysis

There are many similarities regarding the philosophy behind the Intel and ARM platforms isolation mechanisms. For example, the ELs in ARMv8-A platform are similar to the rings on Intel platform. Both architectures also contain a MMU to provide virtual address translation for software running on primary units. Furthermore, the emergence of architectures that provide more capabilities to secondary units has urged Intel and ARM to improve their isolation technologies targeting peripherals (VT-d for Intel and SMMU for ARM). In addition, virtualization extensions are implemented on these two platforms and provide similar isolation functionalities. In other words, Intel and ARM provide a set of isolation techniques that are relevant to cope with each attack scenario identified in section II-A. More precisely, none of the columns of the table II and the tableIII are empty, whether or not the environment is virtualized. All these aforementioned mechanisms are rather close in terms of philosophy and provide a common base to both architectures. However, they still have some specificities.

On one hand, the different Intel architectures have always been designed and developed with the aim of providing backward compatibility, i.e., interoperability with older legacy architectures. Consequently, there are often multiple ways to perform the same operation, and several isolation mechanisms stacked up over time (e.g., pagination over segmentation). As a consequence, more communication channels must be isolated in comparison with the ARM architecture (e.g., threats **2.1** and **2.2** for the MMU), and some mechanisms complete each other. Nevertheless, Intel still promotes innovative technologies with adapted isolation mechanisms. For example, Intel is the only one to provide a technology to improve the performances of peripheral sharing between multiple VMs (SR-IOV) with corresponding adjustments in the I/O MMU. Nonetheless, a similar technology could be developed by ARM if its processors were used more often in servers.

On the other hand, the fact that backward compatibility is not a requirement allows ARM to build new architectures by rethinking their isolation mechanisms. Therefore, these mechanisms seem less scattered inside the architecture and more thorough. Moreover, even if the Intel isolation mechanisms have influenced those implemented by ARM, the latter has still developed genuine and innovative solutions to address segregation issues. The most obvious example is TrustZone technology which provides a complete ecosystem to perform privileged actions on the system (like SMM mode), the ability to verify the "chain of trust" at startup (like SMX and TPM),

[3]EL2 and virtualization extensions are not available in the secure world.

[4]There is a "secure" version of some system registers, for example those dedicated to the MMU configuration.

and to host secure and isolated applications which can be used for DRM or other cryptographic purposes.

IV. Conclusion

In this paper, we analyzed hardware isolation mechanisms included in the latest Intel and ARM platforms, and more precisely in the *Intel 64* and *ARMv8-A* architectures. We can highlight the following contributions. We proposed a generic model to represent the components of each studied architecture and their communications. This abstraction enabled us to identify possible threats that the modeled systems may face. Then, we presented the isolation mechanisms implemented on each platform, and highlighted the threats addressed by them. A comparative analysis of these two platforms and their isolation mechanisms was also carried out.

We first noticed that many concepts about existing segregation technologies are shared between these two architectures. However, the resulting implementations have several differences. This is particularly due to the lack of standardization of these technologies, as well as of the different strategies adopted by Intel and ARM concerning their architectures.

These architectures are constantly evolving. In particular, Intel has designed the skylake architecture with a new extension named *Memory Protection Extensions* (MPX). This new technology enables the CPU to perform fine grained access controls on memory areas in order to prevent buffer overflows. To our knowledge, there is still no equivalent for ARM that employs a slower evolution scheme, with major technological leaps and less backward compatible solutions. However, we can strongly assume that if this extension becomes more mainstream, it will be integrated as well.

We believe that future architectures will increasingly use heterogeneous CPU cores. As a consequence, their design may be quite different, some of them may embed previously mentioned isolation technologies while others may not include some of them. We can for example mention Intel *Active Management Technology* AMT, ARM specialized co-processor, AMD *Project Skybridge*, and *manycore* architecture in general. Therefore, new issues will certainly arise from this special usage, and the combination of existing isolation mechanisms for each architecture may not be sufficient.

Furthermore, with *InfiniBand* technology, it is now possible to remotely access high performance storage area such as RAM. With the evolution of PCIe specifications, peripherals will most probably be moved away of physical computers. We do not consider such communication buses in this paper, but they have to be considered in future work as a part of the attack surface. Consequently, future isolation mechanisms will have to be designed to consider these buses and memories.

The mechanisms that were studied in this paper are very important from the security standpoint because, if properly employed, they are effective and difficult to circumvent or corrupt. However, these isolation techniques alone are not enough to counter any type of attack. It is first and foremost necessary to check their complementarity to cover all potential threats. In addition, these hardware segregation mechanisms only provide fundamental protections between components and can hardly be effective against complex attack scenarios (e.g., using the contribution of several components) because of the classic "semantic gap" problem. It is therefore necessary to use other complementary protection mechanisms, at a higher level of abstraction, which are semantically richer.

References

[1] ARM Limited. ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0487a.h/index.html.

[2] ARM Limited. ARM CoreLink TZC-400 TrustZone Address Space Controller Technical Reference Manual. http://infocenter.arm.com/help/topic/com.arm.doc.100325_0001_02_en/index.html.

[3] ARM Limited. ARM Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0. http://infocenter.arm.com/help/topic/com.arm.doc.ihi0069b/index.html.

[4] ARM Limited. ARM System Memory Management Unit Architecture Specification, SMMU architecture version 2.0. http://infocenter.arm.com/help/topic/com.arm.doc.ihi0062d.b/index.html.

[5] Victor Costan and Srinivas Devadas. Intel sgx explained. Technical report, Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086.

[6] Loïc Duflot and Olivier Levillain. ACPI et routine de traitement de la SMI. In *Actes du 7ème symposium sur la sécurité des technologies de l'information et des communications (SSTIC)*, pages 132–168, 2009.

[7] Intel Corporation. Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes:1, 2A, 2B, 2C, 3A, 3B, and 3C. www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf.

[8] Intel Corporation. Intel® 8 Series/C220 Series Chipset Family Platform Controller Hub (PCH). http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/8-series-chipset-pch-datasheet.pdf.

[9] Intel Corporation. Intel® Trusted Execution Technology (Intel® TXT) Software Development Guide, Measured Launched Environment Developer's Guide. http://www.intel.com/content/dam/www/public/us/en/documents/guides/intel-txt-software-development-guide.pdf.

[10] Intel Corporation. Intel® Virtualization Technology for Directed I/O. http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf.

[11] Intel Corporation. Mobile 4th Generation Intel ® Core TM Processor Family, Mobile Intel ® Pentium ® Processor Family, and Mobile Intel ® Celeron ® Processor Family. http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4th-gen-core-family-desktop-vol-1-datasheet.pdf http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4th-gen-core-family-desktop-vol-2-datasheet.pdf.

[12] Intel Corporation. MultiProcessor Specification. http://www.intel.com/design/pentium/datashts/24201606.pdf.

[13] Fernand Lone Sang, Vincent Nicomette, and Yves Deswarte. Ironhide : plate-forme d'attaques par entrées-sorties. In *Actes du 10ème symposium sur la sécurité des technologies de l'information et des communications (SSTIC)*, pages 237–265, 2012.

[14] Benoît Morgan, Éric Alata, Vincent Nicomette, and Mohamed Kaâniche. Bypassing IOMMU protection against I/O Attacks. In *7th Latin-American Symposium on Dependable Computing (LADC'16)*, 2016.

[15] PCI-SIG. PCI Express® Base Specification Revision 2.1. https://pcisig.com/specifications/pciexpress/base2/.

[16] PCI-SIG. Single Root I/O Virtualization 1.1 Specification. https://pcisig.com/specifications/iov/single_root/.

[17] Junaid Shuja, Abdullah Gani, Kashif Bilal, Atta Ur Rehman Khan, Sajjad A. Madani, Samee U. Khan, and Albert Y. Zomaya. A Survey of Mobile Device Virtualization: Taxonomy and State of the Art. *ACM Comput. Surv.*, 49(1):1:1–1:36, April 2016.

[18] Trusted Computing Group. Tpm specification version 1.2 : Design principles. http://www.trustedcomputinggroup.org/tpm-main-specification/.

[19] Arun Viswanathan and BC Neuman. A survey of isolation techniques. *University of Southern California, Information Sciences Institute*, 2009.