Latest Trends in Hardware Security and Privacy

Giorgio Di Natale, F. Regazzoni, V. Albanese, F. Lhermet, Y. Loisel, A. Sensaoui, S. Pagliarini

Latest Trends in Hardware Security and Privacy

Giorgio Di Natale[1], Francesco Regazzoni[2], Vincent Albanese[3], Frank Lhermet[3], Yann Loisel[3], Abderrahmane Sensaoui[3], Samuel Pagliarini[4]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, Grenoble, France,
[2]University of Amsterdam and ALaRI - USI, Switzerland,
[3]Platform Security Team, SiFive, La Ciotat, France,
[4]Tallinn University of Technology (TalTech), Estonia

***Abstract*—In the last two decades we have witnessed a massive development of technologies (both hardware and software) which have enabled the creation of billions of connected devices. These devices are nowadays used in a very wide range of applications, and they all contain different types of valuable assets, which have been the target of an increasing number of cyber attacks. Both scientific and industrial communities have focused their attention to implement new design processes to reduce the risk of cybersecurity breaches. This paper includes two different contributions in the field of hardware security and privacy.**

I. Introduction

In the last two decades we have witnessed a massive development of technologies (both hardware and software) which have enabled the creation of billions of connected devices. These devices are nowadays used in a very wide range of applications, including computing and cloud computing, communication, gaming and entertainment, IoT, automotive, aerospace, defense. All these applications contain different types of valuable assets (e.g., safety, fortune, personal data, privacy, copyrights, industrial secrets). As a consequence, together with the development of versatile technologies and applications, we have also experienced an increasing number of cyber attacks, which use more and more sophisticated means (both hardware and software) to, for instance, improperly access sensitive information, extorting money, or interrupting services. Both scientific and industrial communities have focused their attention to implement new design processes to reduce the risk of cybersecurity breaches. This paper includes two different contributions in the field of hardware security and privacy.

The first contribution, provided by SiFive, describes how software isolation can be guaranteed by implementing the so-called WorldGuard architecture. Software isolation is an important feature mandated by distinct concerns (safety, security). Considering the current trend of software code increasing size and its potential various origins, important risks exist that one software part can affect another one on the same platform, intentionally or not. A secure software isolation solution could be easily developed on a RISC-V core, thanks to the privileged modes and the Physical Memory Protection (PMP) blocks. Nevertheless, this approach has a major limitation for complex platforms like SoCs, since the privileged modes and the PMPs are attached to a single core, and their configurations are neither shared nor synchronized with other cores. A hardware system-level resource isolation solution is therefore necessary. SiFive proposes the *WorldGuard* architecture, which provides a secure and scalable solution to address the modern security requirements on SoCs equipped with multiple masters, hosting complex software architecture. On a SoC, WorldGuard allows defining several worlds, as subsets of the SoC masters, memories, and peripherals. Each world behaves independently from the others, with a high level of trust.

The second contribution, provided by Samuel Pagliarini from Tallinn University of Technology, brings a discussion on how software threats and defences can be analyzed from a hardware-point-of-view. Indeed, for decades, software-based plagues like viruses and malwares have been infecting our computers and often spreading through our local networks. Yet, we have developed many successful defense strategies, from the obvious anti-virus software to firewalls and clever honeypots. This contribution will address whether these concepts have equivalents in hardware and what they would look like if so – the most obvious example is the comparison between hardware trojan horses and viruses. Moreover, we will analyze who the attackers are in both domains and the different degrees of anonymity an attacker can benefit from. The paper also presents, as a case study, a silicon-validated ransomware attack, a prime example of how a software threat can indeed make the transition to hardware.

II. WorldGuard: a SoC-level isolation solution

Software isolation is an important feature mandated by distinct concerns, safety, and security. Considering the trend of increasing code size, the different origins of the code and the ways they are combined, important risks appear where a piece of software can affect another one, intentionally or not. A health-oriented device (a medical device, a car brake, a nuclear plant sensor) shall keep its ability for its main mission, even if another piece of software running on the device has bugs and goes into infinite loop or erratic behavior (see [1]). A credential-oriented device (payment terminal, badge control system, DRM-equipped device) shall resist or raise a flag, and not compromise any asset it protects (see [2]).

A. The Rationale Behind WorldGuard

Therefore, the goal is then to guarantee that an aggressive (security) or uncontrolled (safety) piece of software cannot affect a resource belonging to another piece of software running on the same platform (see [3]). Changing any resource can have a broad meaning, from simply reading a part of memory to overwriting it or preventing the use of a peripheral. Developing a satisfying solution of isolation shall also consider the major trend of integration in the semiconductor industry, where increasingly complex SoCs are designed, including increased

*Institute of Engineering Univ. Grenoble Alpes

978-1-7281-9457-8/20/$31.00

functions on the same die, for a lower cost. Today, an offer for a security software isolation can be easily developed on RISC-V cores, thanks to the clever privileged modes management and the PMP (Physical Memory Protection) (see [4]). This approach is based on a hypervisor, aka security monitor, being able to manage the different independent pieces of software, the boxes. The boxes' resources (memories, peripherals) and the inter-boxes communications are dynamically controlled and allocated by the hypervisor, upon request from the boxes. A great benefit of this approach is the almost unlimited number of boxes.

On open and flexible platforms, such as RISC-V-based products, it is ideal to assign any important function or task to only one single box, separated from the others, for example: one box for the cryptographic computations, another one for the communication library, one for the data sensors, and one for the main application: this helps to limit the risk of a bug spread among the whole software. Therefore, a common software architecture is to have the security monitor running in machine mode (M-mode), controlling boxes. Each of the boxes could be made of two parts, the operating system, running in supervisor mode (S-mode) and the application tasks, running in user mode (U-mode).

However, there is a major limitation for this approach on the more complex multi-core, multi-master SoCs: the privileged modes and the PMPs scopes are restricted to their core and their software, so they can only supply control on the memory-mapped areas for this single core. The privilege management registers belong to one single core, so their values and the resulting controls are neither shared nor synchronized with other cores. Moreover, other masters such as DMAs do not have any PMP and are independent from the cores and their applications; they are then able to circumvent resources access restrictions set by the cores, as these restrictions are not applicable to the other masters (see [5]). Solutions based on software control of the DMAs undermine the benefits of such IP blocks in terms of performance and efficiency.

So, for overcoming those limitations, SiFive proposes a very smart and secure solution named the WorldGuard architecture (see [6]). There are two different flavors in WorldGuard, the core-driven strategy and the process-driven strategy. The process-driven strategy proposes an even finer granularity than the core-driven, but the same general principles apply, and both can be combined anyway if needed: this architecture proposes to set rights at masters' level and check the rights at slaves' side.

B. Core-Driven Strategy

The principle is to gather masters, memories areas and peripherals within sets of resources, named worlds. In this strategy, the software has no specific property and automatically belongs to the world the master it runs on belongs to. The worlds are distinct from each other, which means the software running on cores belonging to a world 1 cannot access resources (memory portions, peripherals) belonging to a world 2 and more generally cannot interfere with the other worlds. Of course, for inter-world communications, some memory portions can belong to more than one world, behaving as shared memory.

So, one unique world ID value is assigned to a world, and within this world, each of the masters is assigned the same world ID value. This allocation is usually for each master, like a core, but for enhanced IP blocks, it can be with a finer grain. For example, it can be for each channel of a multi-channel DMA. Therefore, each time a data request leaves the master, this request is marked with the world ID. On the slave side, each memory or peripheral has an access control list (ACL) having the access rights per world ID. Through the communication bus, each marked request coming from the master is sent to the slave and analyzed versus its ACL. These ACL check IP blocks have their functions similar to the PMPs functioning with the world ID field acting as an additional field to be checked. This mark-and-check strategy allows complex and rich architectures to be easily secured as a wrong or missing world ID implies the immediate rejection of the request. There is no practical limitation in the number of supported worlds for a specific platform, being then consistent with the software-only proposal above: the number of worlds a SoC can support is application-dependent and shall be determined by the application architect.

C. Process Driven Strategy

Everything above still applies for the process-driven strategy except for the cores' management: the world distinction can now also depend on the privileged mode, having distinct world IDs based on the privilege mode (M-mode, S-mode, U-mode) on the same core. The obvious scenario for this process-driven strategy is the application of the hypervisor/security monitor scenario described above on a multi-core, multi-master SoC. The simplest example is to have a core "hosting" two worlds, the world 1 accessing the most sensitive assets (e.g., keys) and to some specific peripherals (e.g., the fingerprint reader on a smartphone), and the world 2 running generic application software.

D. WorldGuard Configuration Setup

The WorldGuard worlds configurations, including assignments of masters, definitions of ACLs, shall be set up at platform startup. This is an operation that is usually performed after the secure boot operation, the WorldGuard configuration firmware being the first software to be run by the secure boot. It is only once this configuration is completed that the application can start.

III. Re-Imagining Software Threats and Defences in Hardware

For decades, software-based plagues like viruses [7] and malwares [8] have been infecting our computers and often spreading through local networks and email communication [9]. Recently, incidents related to ransomwares have been reported to shut down airports, car manufacturing plants, and even entire local government systems. Threats keep evolving and new defensive approaches have to keep evolving as well.

On the hardware side, however, it could be argued that attacks have not (yet) led to such large-scale incidents, but the threat is looming. An episode that certainly was eye opening

for the entire community was the discovery of the Meltdown/Spectre [10] vulnerabilities. However, despite the reach and plausibility of exploitation by an attacker, the unearthed flaws seem not to be intentional. The obvious research question then becomes would an attacker be able to create such an intricate malicious logic? If so, what are the lessons that can be learned from threats and defences utilized in software?

Due to sheer necessity, security experts have developed many widely adopted defense strategies, from the obvious antivirus software to network firewalls. These concepts, however, do not have strict equivalents in hardware. Perhaps, the closest comparison to be made is that a software virus resembles a hardware trojan [11] to some extent, especially if their intent is simply to corrupt data or execution. In terms of defenses against software-based threats, one can deploy static analysis in order to expose (a signature of) a malicious code. On the hardware side, many works have attempted to use current/power measurements that would indirectly detect the presence of extraneous logic [12]. This process, albeit feasible at first sight and akin to code static analysis, gets more convoluted as circuits fabricated in present-day technologies display a high degree of process variation. It is not uncommon for chips from the very same batch to present static currents that differ by one order of magnitude. Trojans of small sizes can effectively "hide" in the margins of the fabrication process. The same way antivirus detection routines has evolved from simple static checks to signature-less approaches, trojan detection has also shifted to dynamic approaches like concurrent execution and (self-)checking [13]. Other attacks that target hardware take different forms. For instance, side channel attacks try to exploit circuit emanations (power signatures, timing, electromagnetic radiation, or even sound) to discover through correlation what is being computed. The classical example is the use of side channel traces to discover encryption keys that should, otherwise, remain a secret. Here, detection does not help because the attacker performs his/her analysis in a non-invasive manner. The attacker only needs physical access to the circuit. Efforts from the hardware security community shifted to avoidance instead of detection, i.e., making sure that the circuits being designed do not leak useful information [14].

However, there might be other approaches that promote deception instead of avoidance. Not surprisingly, this is exactly the role of honeypot servers in networks: usually, an isolated server is employed, which has unnecessarily obvious open ports or other vulnerabilities that attackers are probably on the lookout for. The research question then becomes what would be a hardware honeypot. The goal would be to devise a circuit (or portion thereof) that would attract attackers. Perhaps this can be achieved in the context of reverse engineering: maybe a circuit layout can be devised to look like it stores a set of keys for cryptography in very easy to identify fuses, while the real set of keys rests safely in a camouflaged layout. By the time the attacker understands the con (and let us assume he/she will, given enough time and determination), precious resources were already wasted looking in the wrong place. This is already a win for a defence strategy. In other contexts, such as hardware trojan insertion and fault injection, it is likely that effective hardware honeypots can also be devised.

Table I
ATTACKERS AND THEIR ROLES.

Attacker	**Attacker role**
3PIP provider	Contracted by the design house, provides specific IP that is part of a larger project. IP might be compromised.
Foundry	In charge of fabrication, provides the silicon for the design house. Might introduce small trojans or parametric trojans.
CAD vendor	Develops the CAD tools that the design house utilizes. Theoretically, could introduce malicious logic into a design.
SoC integrator	Design houses may procure other companies for finishing the top-level of their chip. The integrator is a 3rd party that enjoys full visibility of the project, potentially being capable of inserting localized and smart trojans.

Another example of a successful strategy for vetting third party software is the use of virtual machines. The process consists of running the software under test in a virtualized environment while logging its activity: system calls, disk usage, access to operating system registry, etc. However, despite the widespread use of 3rd party intellectual property (3PIP) in integrated circuit design, no such virtual environment exists for vetting IPs. It could be argued that emulation platforms utilized in functional verification could serve this purpose, but the fact is that trojans/backdoors are not meant to be triggered by regular stimuli. Quite the opposite, trojans/backdoors are designed specifically to fly under the radar of verification vectors (functional) and test vectors (structural). The hypothesis is that a hardware within hardware solution could serve as a virtualization testbed while also allowing for logging and isolation if deemed necessary. Take a typical system-on-chip (SoC) as an example, containing a processor, a memory subsystem, and a few specialized accelerators, all connected by a shared bus. If one of the accelerators comes from a third party, it should be possible to bar it from accessing the main memory and other resources while providing it with its own memory space and an individualized bus. This approach, while feasible, is likely to incur overheads that are not suitable for all applications.

Furthermore, it should be noted that attackers in software and hardware domains are inherently very different from one another. For both software- and hardware-based exploits, the landscape comprises rogue individuals all the way to nation states. What is a distinguishing trait is where the attacker is located: often, an individual that has never worked for the company whose product he is attacking can create a computer virus for it. In hardware, however, we often assume the attacker is tightly involved in the development/fabrication process of the chips. This scenario is summarized in Table I, where it is assumed a chip is developed by a fabless design house.

As part of our ongoing research at TalTech's Centre for Hardware Security, we are investigating several avenues of research where (malicious) software finds a hardware counterpart. We have recently demonstrated in silicon what is believed to be the first hardware-only ransomware [15], which we describe in the text that follows. First and foremost, a ransomware attack only makes sense if the application/SoC has access to some form of storage that holds user data. Even further, this data

must be of value such that the user would be compelled to pay a ransom to acquire it back. Not all applications have this profile. For instance, processor-centric SoCs have a large amount of memory for cache, but the memory content is transient and matters not to the user. On the other hand, a system with embedded non-volatile memories makes for a much better target for mounting this type of attack. Assuming this characteristic is in place, the attacker has to reason about a trigger, the payload, and a communication channel back to the user (i.e., to notify the user he has been targeted by a ransomware).

For the trigger, approaches from hardware trojans can be borrowed: a rarely occurring combination of signals can be used. For the payload, a public key cryptography scheme based on elliptic curves was employed. This choice is not arbitrary, elliptic curves lend themselves well to hardware implementations with small footprints. For the communication, we make use of a UART to transmit data serially, in and out of the chip. The logic was implemented such that, once the ransomware is activated, all user data is encrypted and a shared key is provided to the user. This key serves as an identifier, for which the attacker can calculate the reciprocal shared key. If the user provide the chip with this reciprocal key, the user data is decrypted back to plaintext.

A concern we had while developing the ransomware is to make sure each chip generates a unique identifier. A physically unclonable function (PUF) was employed for this purpose. The PUF is implemented as an SRAM IP. Bench tests revealed that the generated bits present good entropy, even though a commercial memory compiler was utilized for generating the IP. A microphotograph of the fabricated chip is shown in Fig. 1. The technology node is 65nm and the fabrication was executed in a multi project wafer (MPW) in March 2020. Packaged parts were received in May 2020 and validated soon after. All parts passed our bench tests, successfully performing key generation, encryption and decryption on demand. The die size is 1mm x 1mm because that is the standard MPW offering. However, the actual size of the ransomware logic, when all debug structures are unaccounted for, is approximately 0.13mm2. More details about the fabricated design can be found in [15].

Finally, let us conclude by saying that our fabricated chip has no malicious purpose per se. Our intent was to show the technological feasibility of such attack. This specific design was executed by two first year PhD students, from RTL to sign-off. It would probably be naïve to assume a motivated adversary would not be able to mount a similar (or improved) attack in a real system.

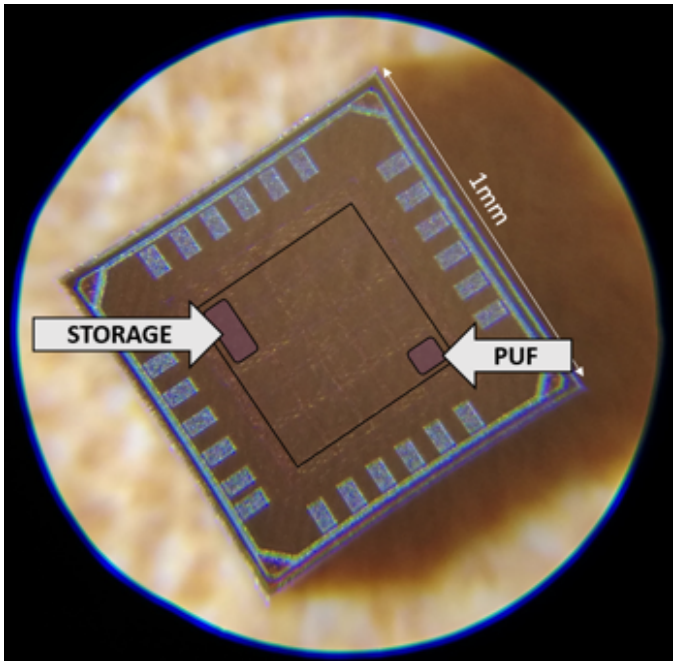

Figure 1. Microscope image of the fabricated chip. Die dimensions are 1mm x 1mm. Highlighted structures in the core area are the memories for user storage and PUF.

References

[1] "Fbi report about health care cybersecurity," FBI Cyber Division, Tech. Rep., 04 2014. [Online]. Available: https://publicintelligence.net/fbi-health-care-cyber-intrusions/

[2] "Payment card industry pin transaction security point of interaction modulare security requirements v6.0," PCI Security Standards Council, Tech. Rep., 06 2020. [Online]. Available: https://www.pcisecuritystandards.org/document_library?category=pts&document=pci_pts_poi_sr_2

[3] "Iot devices exposed to cyber risk," Tech. Rep., 03 2020. [Online]. Available: https://securitybrief.com.au/story/iot-devices-more-at-risk-of-cyber-attack-than-ever-report

[4] A. Waterman and K. Asanovic, *RISC-V PMP specification*, 5 2017. [Online]. Available: https://content.riscv.org/wp-content/uploads/2017/05/riscv-privileged-v1.10.pdf

[5] "Iot security guidelines for endpoints ecosystems," GSM Association, Tech. Rep., 02 2016. [Online]. Available: https://www.gsma.com/iot/wp-content/uploads/2016/02/CLP.13-v1.0.pdf

[6] "Sifive unveils worldguard," Tech. Rep., 10 2019. [Online]. Available: https://riscv-association.jp/en/2019/10/sifives-new-open-security-details-shield-and-worldguard-unveiled/

[7] C. C. Zou, D. Towsley, and W. Gong, "Modeling and simulation study of the propagation and defense of internet e-mail worms," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 2, pp. 105–118, 2007.

[8] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, M. Almgren, V. Gulisano, and F. Maggi, Eds. Cham: Springer International Publishing, 2015, pp. 3–24.

[9] P. Knight, "Iloveyou: Viruses, paranoia, and the environment of risk," *The Sociological Review*, vol. 48, no. 2_suppl, pp. 17–30, 2000. [Online]. Available: https://doi.org/10.1111/j.1467-954X.2000.tb03518.x

[10] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.

[11] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[12] X. Wang, H. Salmani, M. Tehranipoor, and J. Plusquellic, "Hardware trojan detection and isolation using current integration and localized current analysis," in *2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, 2008, pp. 87–95.

[13] R. S. Chakraborty, S. Pagliarini, J. Mathew, S. R. Rajendran, and M. N. Devi, "A flexible online checking technique to enhance hardware trojan horse detectability by reliability analysis," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 2, pp. 260–270, 2017.

[14] K. Tiri and I. Verbauwhede, "A digital design flow for secure integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 7, pp. 1197–1208, 2006.

[15] F. Almeida, M. Imran, J. Raik, and S. Pagliarini, "Design of a 5w 0.13mm2 hardware ransomware in 65nm cmos," in *26th Asia and South Pacific Design Automation Conference (ASP-DAC) (submitted to)*, 2021.