Virtual Platform to Analyze the Security of a System on Chip at Microarchitectural Level

Quentin Forcioli, Jean-Luc Danger, Clémentine Maurice, Lilian Bossuet, Florent Bruguier, Maria Mushtaq, David Novo, Loïc France, Pascal Benoit, Sylvain Guilley, et al.

Virtual Platform to Analyze the Security of a System on Chip at Microarchitectural Level

Quentin Forcioli*, Jean-Luc Danger*, Clémentine Maurice†, Lilian Bossuet‡,
Florent Bruguier§, Maria Mushtaq§, David Novo§, Loïc France§,
Pascal Benoit§, Sylvain Guilley¶ Thomas Perianin¶
*LTCI, Télécom Paris, Institut Polytechnique de Paris, France
†Univ. Lille, CNRS, Inria, France
‡Laboratoire Hubert Curien, St Etienne, France
§LIRMM, Univ. Montpellier, CNRS, Montpellier, France
¶Secure-IC, France

***Abstract*—The processors (CPUs) embedded in System on Chip (SoC) have to face recent attacks taking advantage of vulnerabilities/features in their microarchitectures to retrieve secret information. Indeed, the increase in complexity of modern CPU and SoC is mainly driven by the seek of performance rather than security. Even if efforts like isolation techniques have been taken to thwart cyberattacks, most microarchitectural features can open the door to security holes. One typical example is the exploitation of cache memory which keeps track of the program execution and paves the way to side-channel (SCA) analysis and transient execution attacks like Meltdown and Spectre, which take advantage of speculative execution. This paper introduces an ongoing study aiming at analyzing the attacks relying on the hardware vulnerabilities of the microarchitectures of CPUs and SoCs. The main objective is to create a virtual and open platform that simulates the behavior of microarchitectural features and their interactions with the peripherals, like accelerators and memories in emerging technologies. The gem5 simulator, whose configuration can be customized to a specific CPU or SoC architecture, is the basis of our chosen platform for security analysis.**

***Index Terms*—Security, Microarchitecture, Simulation, gem5, SoC**

I. Introduction

Cyber-security has an ever-increasing effect on our day-to-day lives. Although in the beginning, the attacks were mainly concentrated on the Operating System (OS), software, network stacks, smartcards, or discrete components on the PCBs, in recent years a new type of threats, known as microarchitectural attacks, has infiltrated every components in Systems-on-Chip (SoCs).

We observe a growing trend in the recent attacks to exploit vulnerabilities/features in on-chip components to finally gain control of protected information. Many attacks on caches have been well known for a few years: namely Prime+Probe [33] and Flush+Reload [49], which have been used to guess cryptographic keys or break Address Space Layout Randomization (ASLR) [18]. Attacks on CPU cores include Meltdown [31], which exploits out-of-order execution in modern Intel processors, and Spectre [28], which exploits speculative execution. Both of them use a covert channel to retrieve the data. Malware and keyloggers from integrated GPUs have been reported [11], [14], [29], [41], [45], and cache-timing attacks can be mounted from integrated on-chip FPGAs [4], [6], [11], [12]. Even the main memory has not been spared from these attacks, one of the prime examples being Rowhammer [26]. In face of this growing number of attacks and diversity, it is urgent to address this issue comprehensively, encompassing all the components, namely core, cache hierarchy, accelerators or I/O devices, and main memory.

To tackle the security analysis of processors and SoC against microarchitectural attacks, we propose to investigate a novel method that allows the designer to detect vulnerabilities and validate their exploitability through simulating attacks exploiting them. We use a virtual platform relying on the gem5 simulator [7]. The latter is a cycle-accurate simulator that presents many interests. Among them is the support of most CPU manufacturers, like ARM, and the ability to provide accurate information to observe the microarchitecture behavior and perpetrate attacks. Another interest is the possibility to simulate some components of the heterogeneous SoC architecture. This allows to reproduce the realistic behavior of accelerators or memories and execute fault attacks like the Rowhammer [39] on DRAM or Dynamic Voltage and Frequency Scaling (DVFS) in a cryptographic accelerator.

The paper is organized as follows. After Section I, which presents the context of the study, Section II gives an overview of the state-of-the-art of attacks against CPU and SoC. Then Section III describes the principle of the analysis method relying on a virtual platform built around the gem5 simulator. Section IV gives concrete examples of attack simulations with gem5. The perspectives of the study are presented in Section V before the conclusion in Section VI.

II. State of the Art

A. Attacks and defenses on CPUs

a) Microarchitectural side-channel attacks: Side-channel attacks aim to recover secrets by uncovering traces left in the microarchitecture by some victim program. These have typically targeted the cache [18], [30], [33], [49] and other elements such as the branch predictor [1], [15]. Cache timing attacks [33], [49] have been the most studied and

rely on differences in latencies between the CPU cache and the main memory. In particular, Flush+Reload [49] allows observing loads from other processes at the cache line granularity and across cores by using shared read-only memory between the attacker and the victim (e.g., a shared library). The attacker continuously flushes a cache line from the whole hierarchy and reloads it while timing the operation. If the victim used the cache line between the flush and the reload operations, the attacker observes a fast access; otherwise, a slow access. Other cache attack techniques include Prime+Probe [33] or Flush+Flush [20]. Cache attacks have been showed to be able to recover cryptographic secrets [33], [49], user inputs [21], and addressing information [18]. While they were mainly carried on x86, they have also been demonstrated on ARM [30].

Covert channels are a special case of side-channel attacks where the attacker controls both ends, the sender and the receiver. All techniques used for side-channel attacks can be used as a covert channel, e.g., Flush+Reload & Prime+Probe.

b) Transient execution attacks: Spectre [28] and Meltdown [31] have been the first of many *transient execution attacks* discovered since 2018 [8]. Modern CPUs rely on optimization mechanisms such as out-of-order execution and prediction to keep the pipeline full at all times. Mispredictions and faults are handled by flushing the pipeline to ensure correctness at the architectural level. However, *transient instructions*, i.e., instructions that started to be executed but whose results were never committed to the architectural state, leave traces in the microarchitecture, e.g., caches or branch predictors. Transient execution attacks therefore exploit these transient instructions to encode a secret – which is not normally accessible to an attacker – into the microarchitectural state. More formally, the transient instructions act as the sender of a microarchitectural covert channel. While Flush+Reload is typically used as a covert channel for its easy use and high speed, any microarchitectural covert channel can be used.

c) Defenses: Several past work have tried to provide a framework to evaluate attacks on real hardware. Some approaches use dynamic analysis [47], static analysis [9], [13], [42], or a combination of static and dynamic analysis [46], while other approaches directly measure cache hits and misses [21], [48] to find paths of exploitation for vulnerable implementations.

B. Attacks and defenses on System on Chips

Systems-on-Chip (SoC) are becoming increasingly complex as they integrate more and more functionalities including many processor cores, memory, third party hardware IPs, and reconfigurable hardware (i.e., FPGA) for hardware acceleration. Consequently, these systems are highly vulnerable to many inside-SoC physical attacks including SoC security corruption by using malicious IP/hardware, side-channel analysis, fault injection and covert channel. SoC security corruption attacks have been proposed first by Jacob et al. [23] which showed how a malicious hardware IP can access processor core features and memory to bypass software or system security such as the secure boot. Then Benhani and Bossuet [6] evaluated the security of the ARM TrustZone technology in an FPGA-based SoC. They presented how to insert automatically at the CAD tool level a malicious hardware to take advantage of security failures in the FPGA fabric and to bypass ARM core security including TrustZone technology.

Many recent works suggest embedding an information leakage sensor inside the SoC to be able to perform physical SCA without the need for external measurements. These works use the sensitivity of the power distribution network to power supply fluctuations due to intrinsic noise [34], [35], [38], [51]. At the SoC level, other side channels can be considered such as cache timing analysis as proposed by Chaudhuri [12] which presented three possible types of attack (direct memory access attacks, cache-timing attacks, and Rowhammer attacks) that can exploit the optional cache coherency between the programmable logic part and the processing system of an FPGA-based SoC.

Malicious use of cache coherency can also be used to perform inside SoC fault injection. In another work, Kim et al. [25] used cache coherency between the programmable logic part and the processing system of the SoC to slow down the execution of a CPU program. They used a hardware Trojan that continuously injects memory transactions, which increases the miss rate in the L1 data cache.

At the SoC level, other ways to perform fault injections are also presented in the literature. Tang et al. proposed the CLKSCREW fault injection attack [43]. This attack used Dynamic Voltage and Frequency Scaling (DVFS) to induce a fault in the seventh round of an AES encryption executed in the secure world. DVFS can also be used as a covert channel to secretly send information from one thread to another [2], [5]. In addition, the security of main memories should not be overlooked. To increase the performance, the memory cells are becoming smaller which makes them more vulnerable to disturbance errors [26]. Rapidly, Rowhammer, an attack exploiting this error, was able to precisely flip bits and gain kernel privileges [39]. This attack has been declined in different versions and across different architectures and environments [19]. Van der Veen et al. showed that the Rowhammer effect can be used to attack an architecture based on ARM processors [44]. This was confirmed by Carru [10] who described a successful attack to recover a private key stored in the secure memory of an RSA signature implementation, by making a specific read of a row in the accessible DRAM. Rowhammer can also be performed using only network requests [32].

III. Analysis Method

A. Principle and challenges

In this section, we present the targeted methodology to detect, characterize and validate new vulnerabilities. Namely, we introduce a *virtual-to-real* approach, enabling the in-depth study of vulnerabilities, attacks, and countermeasures in a virtual sandbox-like environment before validation on a real platform. The method consists of 3 steps:

a) Vulnerability identification: detecting lines of code that are potential targets of cache timing attacks, for example from the source code of a security-sensitive application using static or symbolic analysis methods which have the advantage of being platform-agnostic.

b) Virtual implementation: implementing the attack in a virtual environment (namely gem5) to study its feasibility. Here, the attack configuration can be studied in detail. Indeed, attack parameters or target configuration (at the microarchitecture level or at the system level) impact the outcome of an attack. Additionally, the virtual platform can be used to characterize a vulnerability severity (the amount of leakage), by accounting for several security metrics such as the probability of success of an attack, the number of traces required, etc.

c) Validation in the real world: the necessary final step of the analysis method is to verify the vulnerability acuteness in the real world, by bootstrapping an attack on a real platform.

This *virtual-to-real* method enables several improvements to microarchitectural security research.

Attack analysis: this approach enables the exhaustive study of attack parameters and of targets microarchitectural configurations, which is often not possible with a real-only approach. By using a virtual platform, it is possible to extensively customize target properties, e.g., to modify the size of cache memories, the cache replacement policies, etc. Therefore, it is possible to analyze attacks and their parameters to design better countermeasures.

Vulnerability characterization: cache timing vulnerabilities are more or less exploitable, depending on several factors such as the number of times they are encountered within an execution or the number of secret bits leaking, but also depending on the underlying architecture and the environmental context. For example, third-party processes running concurrently to the victim and the attacker may impact the readability of cache timing traces. Leveraging the virtual platform, it is possible to characterize these factors in detail and to create metrics in order to quantify a vulnerability severity, depending on a given context and on a given microarchitecture.

Countermeasures: our methodology offers the possibility to design and test countermeasures, and to evaluate and compare their effectiveness and performance overhead thoroughly thanks to the debug features of the virtual platform. These countermeasures may be at the hardware or software level, and prevent various attacks, e.g., transient execution attacks, cache timing attacks, fault attacks.

Reproducibility: microarchitectural attacks are notoriously difficult to reproduce. To do so, one needs specific CPU models and it is sometimes unclear whether an attack can be replicated on a different or newer version of a given physical platform. The virtual platform allows researchers and security testers to replicate easily an attack without having access to the proper hardware.

This approach also offers several challenges to tackle:

- Imitating the security vulnerabilities of real architectures. Indeed, simulators, such as gem5, are primarily designed for performance evaluation. As we will detail next, extending such simulators for security analysis requires extensive modeling of additional behavior that is not relevant or has a negligible effect on performance (e.g., the Rowhammer effect in DRAM).
- Facilitating the study of security vulnerabilities by exposing relevant changes in the virtual platform of microarchitectural state to the user. This will require the design of a monitoring infrastructure specially tailored for security analysis.
- Building a list of example existing attacks (either on gem5 or on both gem5 and a real platform) that can be easily tested and demonstrate the different means that are used by the attack thus representing the threat, a countermeasure is mitigating.
- Anticipating new attacks by designing efficient vulnerability detection mechanisms. In addition to studying and validating attacks on the virtual platform, it is important for the tool to be able to uncover new attacks, from static or symbolic source code analysis or from dynamic trace analysis, while also being able to detect abnormal behavior on a real SoC which could hide undocumented security features [17].

It is worth mentioning that the virtual platform can be used in combination with existing security evaluation tools: for example, a vulnerability identification step can be performed by static and symbolic analysis of source code [9], [42], which are functionalities embedded in the Catalyzr™ tool developed by Secure-IC. The found vulnerabilities may then be analyzed in depth with the virtual platform. Ultimately, the usage of the platform within global security assessment tools would enable a better coverage of threats and better understanding of vulnerabilities when using such tools in the industry.

B. gem5

We build our virtual environment on gem5, which is a state-of-the-art cycle-accurate SoCs/system simulator that is developed and supported by CPU founders and computer engineering researchers [7]. Developed in C++ and using a Python interface to easily configure architectures (as shown in Figure 1), gem5 provides a platform to develop microarchitecture and an API implementation in a controlled and highly instrumented environment. Our goal is to use this environment to analyze and monitor data leakage through hardware weaknesses (e.g., Spectre) using visualization to help tuning the attack to the target simulated library on a specific microarchitecture. The simulator accuracy regarding ARMv8 security-sensitive features and behavior is one of our focuses as we are checking whether an attack working on the simulator will also work on real hardware and vice versa.

1) gem5 features: A gem5 simulation is divided between the Python configuration files and the C++ files, which are compiled inside the gem5 binary, behaving like a Python interpreter. The configuration file connects Python objects which are either other Python objects or gem5 primitives representing hardware modules, through port generally using

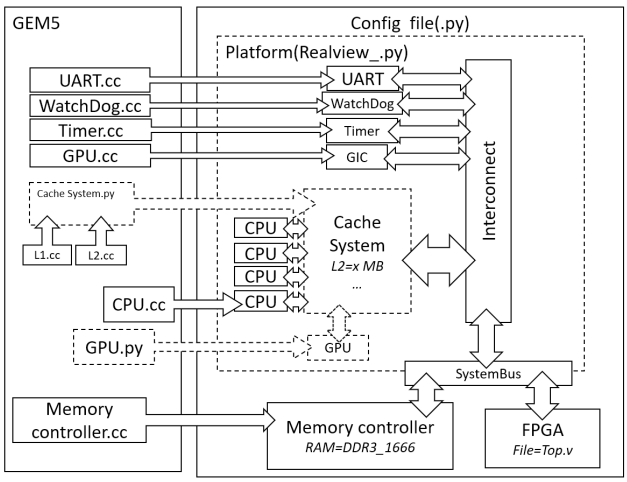

Fig. 1. How a gem5 simulation model is laid out between C++ files and Python files.

a gem5 memory packet protocol which is similar to AXI. This hardware primitive can be configured using parameters directly in the Python configuration file. gem5 also provides a configurable Python file that includes primitive and interconnects them in a specific way (e.g., a cache system made of caches and controllers).

2) Advantages of using gem5: gem5 enables unlimited non-intrusive instrumentation. For instance, the ARM CoreSight allows the designer to have a full trace of execution, information about cache state and MMU/TLB state. gem5 simulates CoreSight but in addition, it allows to know information from every module inside the architecture even if they are not connected to the CoreSight module, e.g., providing information about the state and building statistics for branch prediction (Konata [40]). This allows the debugging of attacks and their customization against a specific library or program, shortening the development time for adapting an attack to a new target.

Because gem5 can also integrate FPGA (using a SystemVerilog simulator) or GPU (with a precise emulation) architectures, it is also possible to realistically experiment attacks on such accelerators.

Additionally, we can use the same disk image to compare with real hardware, while keeping the same binaries and libraries between the real world and the simulated world. This makes testing attacks and ensuring simulation of security features and side-channel sources (e.g., caches, DVFS) more reproducible and easier than with a real platform.

3) Limitations of gem5: Microarchitecture characteristics of the CPU, GPU, FPGA, interconnect, and cache emulation are key for ensuring that we are building actual attacks that work on real hardware. However, these are often black-boxes and we cannot be reasonably sure that they behave exactly like the gem5's version. Some CPU likes the Apple-M1 are even providing new undocumented ISA extensions [24] which would require modification inside gem5 core to be correctly implemented. Moreover, some key features, for instance, TrustZone or Qualcomm Secure Processor Unit, rely on dedicated hardware (like the TZC-400) which is not yet implemented in gem5 while being ubiquitous in off-the-shelf devices. This hinders our possibility to test for weaknesses in current security frameworks since it is not completely implemented, and therefore the simulated behavior may not exhibit the same key vulnerabilities as the real hardware. Finally, the gem5 memory subsystem is not always precise with the information that is passed to devices. For instance, gem5 is not an emulator of the AXI protocol which would allow differentiating between privileged and unprivileged accesses. This is especially the case with an FPGA that would host Hardware Trojans Horses and send forged unauthorized requests.

IV. Practice and Examples of attacks with gem5

A. Bare-metal and OS platform

To have low-level access to all the CPU and OS features to test countermeasures and demonstrate attacks at a theoretical level, we developed a bare-metal system. This allowed us to study how SoC configuration, hardware-wise and software-wise, influences the success and effectiveness of an attack. We designed a tool for developers to have the same environment as a traditional platform (e.g., ARM DS-5 or Xilinx xSDK). This tool provides:

- Basic libc functions, e.g., `malloc`, `printf`.
- Maximum platform compatibility (resilient to SoC modification in gem5).
- Cleaner execution traces (instructions, cache events, ...): bare-metal provides a tighter integration.
- Simple implementation of security features, e.g., System Management Controller (SMC), secure page tables.

This tool ensures that code working on gem5 also works on the real platform.

This platform could also be used to detect hidden features by comparing behavior between the real platform and the virtual. The real hardware could provide some undocumented assistance to the OS to avoid side-channel leakage, such feature would only lead to a difference in behavior in a bare-metal execution.

For example, we have re-implemented a Flush+Reload cache side-channel attack to demonstrate the compatibility between our gem5 platform (gem5 + micro-kernel) and a hardware platform, a Raspberry Pi 3 B+ (BCM2837B0: 4×Cortex-A53) which supports aarch64 and launches code at the highest exception level. At the same time, we are also testing attacks on Linux (incorporating our previous work [3]) using all of this to verify emulation and attack model accuracy.

B. Side-channel attack from a secure environment

ARMv8a, along with other ISAs such as RISC-V or x86, provides a secure environment with controlled access to memory resources and seclusion from classical user code, allowing user applications to run without leaking information through unsanitized memory, cache, or other shared resources, even from code that shares the same address space (e.g., inside

a web browser). This feature that allows building a Trusted Environment is called TrustZone on ARM. It is built using dedicated hardware device and dedicated instructions. There are already several attacks on TrustZone [36], [50] using both intrinsic weaknesses and oversights from the secure environment.

We want to reproduce such attacks on gem5 to serve both as a demonstration that gem5 SoC emulation correctly reproduces TrustZone features, and as a threat model to propose new countermeasures to mitigate this kind of threat.

We set up OP-TEE on gem5 using the recent "TF-A on gem5" ARM tutorial [22] and we plan to demonstrate side-channel attacks from the secure world to the unsecured world using both Spectre and Prime+Probe attacks like in [36]. We also plan to provide an accurate model for the TZC-400 or TZC-380 to have a representative environment when we will have chosen which real platform to compare with gem5 for this attack. Indeed, the Raspberry Pi 3 B+ does not have a TrustZone Space Controller, even though we can set up a TEE on it.

C. Covert channel communication using DVFS

With gem5, one challenge is to reproduce an attack like [5]. It is possible by using the following components: SimObject, EnergyCtrl, DVFSHandler and ClockDomain. The software component SimObject allows to define a clock domain with a specific number of frequencies and to connect one of several CPUs of the SoC model to this domain. In simulation time, the CPU selects a performance level by using the CPUfreq driver which conduces to a frequency selection by the ClockDomain through the EnergyCtrl, DVFSHandler components as presented in Figure 2.

We have implemented these components to perform CPU to CPU covert channel communication exploiting the DVFS. Moreover, we have managed the DVFS from a hardware accelerator if this component has a DMA access and then perform hardware IP to CPU covert channel communication. To conclude, with gem5 we are able to perform all the attacks proposed in [5].

D. Rowhammer on gem5 and Ramulator

We plan to reproduce Rowhammer effects which require an accurate simulation of the main memory. On gem5, main memory management is not native. So, we added a fast and extensible DRAM simulator called Ramulator [27]. This open source simulator provides out-of-the-box timing-accurate models for DRAM standards (DDR3/4, LPDDR3/4...) and allows to easily add new ones. By design, gem5 and Ramulator do not include the management of Rowhammer effects, so we developed a dedicated memory corruption module and updated some existing components.

The modifications allow to:

- Store an associative map to maintain the link between virtual and physical addresses. It determines the neighbors of each row and establish the affected ones.

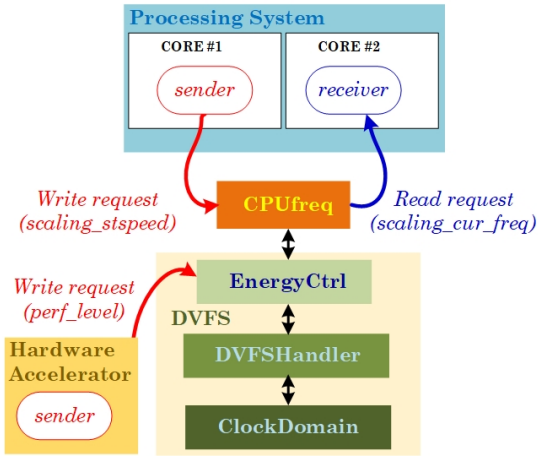

Fig. 2. Possible DVFS covert channel on gem5.

- Count the number of activations in each row since the last refresh.
- Affect one bit-flip threshold to each row.
- Establish if one bit-flip occurs and modify the affected cells in consequence.

This modification will make it possible to successfully run existing Rowhammer attacks on x86 platform associated with classical DRAM. However, our approach can only model the currently known attack surface of the Rowhammer effect. As researchers keep on challenging the newest DRAM technology, this attack surface will expand (e.g., the recent half-double effect reported recently by Google [37]). Accordingly, our gem5 model will need to be upgraded every time a new vulnerability that expands the known Rowhammer attack surface is discovered.

Machine learning was used to develop a detection mechanism (at it was already shown in [16]). The introduction of memory corruption in gem5 will allow us to train and test the detection mechanism with datasets created with attacks that need to witness corruption to work properly.

V. On-going work and Perspectives

Figure 3 describes the structure of the current work which addresses all the parts of a SoC. The main task is to develop the virtual platform relying on gem5 and real hardware platforms. A study will cover all types of cache timing attacks for different configurations of protections and microarchitectures. The capability to simulate heterogeneous SoC allows to tackle the vulnerabilities coming from the accelerators and emerging technologies as Non Volatile RAM (NVM). The main expected outcomes of this project are:

- An open environment to speed up the security study of microarchitectural attacks.
- An anticipation of attacks and validation on both the gem5 virtual platform and a real platform.
- A security analysis in numerous architecture configurations as: multicores, protection levels, accelerators, new memory technologies...

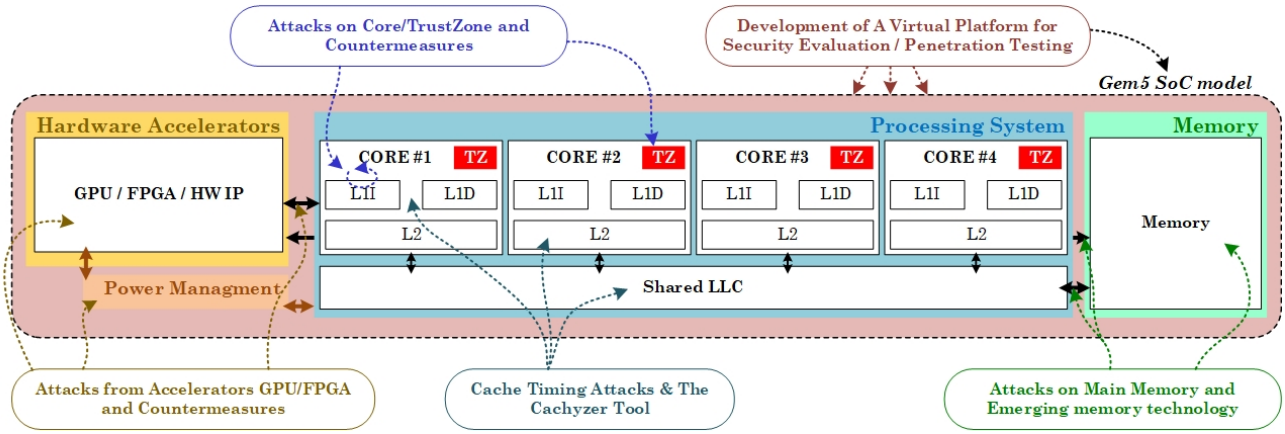

Fig. 3. Target architecture of the project.

- Tools to monitor side channel leaking or fault injection.

VI. Conclusion

We presented a work-in-progress aiming at analyzing the security at microarchitecture level of CPU and SoC with a virtual platform. It relies on the gem5 simulator whose versatility and accuracy allow designers to find vulnerabilities and attacks exploiting them. The practice of gem5, including its advantages and limitations, and some examples of attacks are given. Among them, a cache timing attack on unprotected and protected environment are presented, along with the development of a Bare Metal system and its validation on a real hardware platform. We also show that DVFS and Rowhammer attack can be accurately simulated with gem5. The work is currently on-going and aims at providing to the community an open platform for the security analysis of microarchitectures.

Acknowledgements

The work presented in this paper was realized in the framework of the ARCHI-SEC project number ANR-19-CE39-0008-03 supported by the French "Agence Nationale de la Recherche".

References

[1] O. Aciiçmez, Ç. K. Koç, and J. Seifert, "Predicting secret keys via branch prediction," in *CT-RSA*, 2007.

[2] M. Alagappan, J. Rajendran, M. Doroslovački, and G. Venkataramani, "Dfs covert channels on multi-core platforms," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017.

[3] P. Ayoub and C. Maurice, "Reproducing spectre attack with gem5: How to do it right?" in *14th European Workshop on Systems Security (EuroSec)*, 2021.

[4] E. M. Benhani, L. Bossuet, and A. Aubert, "The Security of ARM TrustZone in a FPGA-based SoC," *IEEE Transactions on Computers*, pp. 1–1, 2019.

[5] E. M. Benhani and L. Bossuet, "Dvfs as a security failure of trustzone-enabled heterogeneous soc," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2018, pp. 489–492.

[6] E. M. Benhani, C. Marchand, A. Aubert, and L. Bossuet, "On the security evaluation of the ARM trustzone extension in a heterogeneous soc," in *International System-on-Chip Conference (SOCC)*, 2017.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[8] C. Canella, J. V. Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in *USENIX Security Symposium*, 2019.

[9] S. Carré, A. Facon, S. Guilley, S. Takarabt, A. Schaub, and Y. Souissi, "Cache-timing Attack Detection and Prevention — Application to Crypto Libs and PQC," in *COSADE*, ser. LNCS, vol. 11421. Springer, April 4-5 2019, Darmstadt, Germany.

[10] P. Carru, "Attack arm trustzone using rowhammer," in *GreHack, 2017*, 2017.

[11] S. Chaudhuri, "Cache Timing Attacks from The SoCFPGA Coherency Port (Abstract Only)," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[12] ——, "A Security Vulnerability Analysis of SoCFPGA Architecture," in *DAC*, 2018, pp. 120–125.

[13] G. Doychev, B. Köpf, L. Mauborgne, and J. Reineke, "Cacheaudit: A tool for the static analysis of cache side channels," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 1, pp. 4:1–4:32, 2015.

[14] L. Duflot, Y.-A. Perez, and B. Morin, "What if you can't trust your network card?" in *RAID*, 2011.

[15] D. Evtyushkin, R. Riley, N. B. Abu-Ghazaleh, and D. Ponomarev, "Branchscope: A new side-channel attack on directional branch predictor," in *ASPLOS*, 2018.

[16] L. France, M. Mushtaq, F. Bruguier, D. Novo, and P. Benoit, "Vulnerability assessment of the rowhammer attack using machine learning and the gem5 simulator-work in progress," in *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, 2021.

[17] M. Green, L. Rodrigues-Lima, A. Zankl, G. Irazoqui, J. Heyszl, and T. Eisenbarth, "Autolock: Why cache attacks on ARM are harder than you think," in *USENIX Security Symposium*, 2017.

[18] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR," in *CCS*, 2016.

[19] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *DIMVA*, 2016.

[20] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *DIMVA*, 2016.

[21] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *USENIX Security Symposium*, 2015.

[22] A. Herrera, "Running trusted firmware-a on gem5," 2020, [Online; accessed 17-May-2021]. [Online]. Available: https://community.arm.com/developer/research/b/articles/posts/running-trusted-firmware-a-on-gem5

[23] N. Jacob, C. Rolfes, A. Zankl, J. Heyszl, and G. Sigl, "Compromising FPGA SoCs using malicious hardware blocks," in *DATE*, 2017.

[24] D. Johnson, "Undocumented arm64 isa extension present on the apple m1," 2021, [Online; accessed 17-May-2021]. [Online]. Available: https://gist.github.com/dougallj/7a75a3be1ec69ca550e7c36dc75e0d6f

[25] M. Kim, S. Kong, B. Hong, L. Xu, W. Shi, and T. Suh, "Evaluating coherence-exploiting hardware Trojan," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 157–162.

[26] Y. Kim, R. Daly, J. S. Kim, C. Fallin, J. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ISCA*, 2014.

[27] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, 2015.

[28] P. Kocher, 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 *S&P*, 2019.

[29] E. Ladakis, L. Koromilas, G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "You can type, but you can't hide: A stealthy gpu-based keylogger," in *6th European Workshop on System Security (EuroSec)*, 2013.

[30] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "Armageddon: Cache attacks on mobile devices," in *USENIX Security Symposium*, 2016.

[31] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *USENIX Security Symposium*, 2018.

[32] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice, and D. Gruss, "Nethammer: Inducing rowhammer faults through network requests," in *Proceedings of EuroS&PW*, 2020.

[33] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *S&P*, 2015.

[34] G. Provelengios, D. Holcomb, and R. Tessier, "Power Wasting Circuits for Cloud FPGA Attacks," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 231–235.

[35] C. Ramesh, S. B. Patil, S. N. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier, "Fpga side channel attacks without physical access," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 45–52.

[36] K. Ryan, "Hardware-backed heist: Extracting ECDSA keys from qualcomm's trustzone," in *CCS*, 2019.

[37] Q. Salman, K. Yoongu, B. Nicolas, S. Eric, and N. Mattias, "Half-double: Next-row-over assisted rowhammer," https://github.com/google/hammer-kit/blob/main/20210525_half_double.pdf, 2021.

[38] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *DATE*, 2018.

[39] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2015.

[40] R. Shioya, "Visualizing the out-of-order cpu model." 2018, [Online; accessed 17-May-2021]. [Online]. Available: http://learning.gem5.org/tutorial/presentations/vis-o3-gem5.pdf

[41] P. Stewin and I. Bystrov, "Understanding DMA malware," in *DIMVA*, 2012.

[42] S. Takarabt, A. Schaub, A. Facon, S. Guilley, L. Sauvage, Y. Souissi, and Y. Mathieu, *Cache-Timing Attacks Still Threaten IoT Devices*, 03 2019, pp. 13–30.

[43] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Clkscrew: exposing the perils of security-oblivious energy management," in *USENIX Security Symposium*, 2017.

[44] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic rowhammer attacks on mobile platforms," in *CCS*, 2016.

[45] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "Gpu-assisted malware," *Int. J. Inf. Sec.*, vol. 14, no. 3, pp. 289–297, 2015.

[46] S. Wang, P. Wang, X. Liu, D. Zhang, and D. Wu, "Cached: Identifying cache-based timing channels in production software," in *USENIX Security Symposium,*, 2017.

[47] S. Weiser, A. Zankl, R. Spreitzer, K. Miller, S. Mangard, and G. Sigl, "DATA - differential address trace analysis: Finding address-based side-channels in binaries," in *USENIX Security Symposium*, 2018.

[48] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," 2016. [Online]. Available: https://cs.adelaide.edu.au/ yval/Mastik/

[49] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *USENIX Security Symposium*, 2014.

[50] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "Truspy: Cache side-channel information leakage from the secure world on arm devices," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 980, 2016.

[51] M. Zhao and G. E. Suh, "FPGA-Based Remote Power Side-Channel Attacks," in *S&P*, 2018.