Reconfigurable Hardware for Microarchitectural Timing Attacks Detection

Yuxiao Mao, Vincent Migliore, Vincent Nicomette

Reconfigurable Hardware for Microarchitectural Timing Attacks Detection

Yuxiao Mao, Vincent Migliore, Vincent Nicomette
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
Email: {firstname}.{lastname}@laas.fr

***Abstract*—Software-based microarchitectural timing attacks evolve quickly, exploiting hardware design properties that widely affect both general-purpose processors and embedded processors. Among the means of attack detection, hardware monitoring benefits from less overhead and less power consumption compared to software monitoring. Nevertheless, as usual hardware cannot be upgraded, the efficiency of a hardware monitor device component cannot be guaranteed against future attacks.**

In this paper, we study the feasibility of using reconfigurable hardware alongside software attacks detection to cope with microarchitectural timing attacks. We propose to use the hardware's capability of parallel execution, to deal with the problem that reconfigurable technologies suffer from a lower frequency than hardwired technologies. This architecture is designed to adapt to new attacks, because the processor can decide to reconfigure the detection logic to take them into consideration. We briefly present an implementation of a proof of concept on FPGA to validate our design.

I. Introduction

Software-based attacks are evolving faster and faster. In recent years, an increasing number of low-level attacks, at the interface between hardware and software have been published, such as microarchitectural timing attacks (MTAs) [1].

During a process's execution, some information related to the execution may be stored in the hardware microarchitecture, such as cache status or execution delay. This information is not supposed to be visible to software. MTAs break this assumption and aim to infer microarchitectural information by measuring the duration of the execution of specific instructions. A noticeable difference in this duration may enable a malicious process to infer some sensitive information of a victim process that shares hardware ressources with the malicious process (such as memory cache lines in the processor). This difference in the execution duration, measured by MTAs, can come from performance optimization, or some design choice [2]. As sacrificing the performance gain or completely preventing leak point in all hardware is unrealistic, MTAs present a true threat to cryptography and other security critical algorithm [3].

Software-based protection is often the first choice for attack detection, because it's easy to deploy and to update. However, at runtime, software can only rely on microarchitectural information provided by underlying hardware such as Hardware Performance Counter (HPC) for instance. This limits the effectiveness of software solutions. In the other hand, hardware-based protection has low overhead and low power consumption, but it is a static and deterministic protection that cannot evolve to cope with new attacks. In this context, we believe that introducing flexibility to hardware for MTAs monitoring is an inevitable trend. For that purpose, we propose in the paper a solution that uses reconfigurable hardware, such as Field-Programmable Gate Array (FPGA).

Section II presents a state of the art of MTAs detection techniques and the use of reconfigurable hardware with processor. Section III provides some details about our reconfigurable architecture, while Section IV describes a proof of concept that is currently implemented in FPGA. Section V draws some conclusions.

II. State of the art

MTAs detection can be divided into two categories: static analysis and runtime monitoring.

Static analysis aims at analyzing the instructions of a given binary file in order to determine whether it contains an attack [4] or not. A file that is suspected to include a malicious code may not be allowed to run, or can only run in strict condition. This method is suitable for scans in the applications store for example and is performed before execution of the binary. However, when the code is encrypted, obfuscated or dynamically loaded, static analysis is not efficient any more.

Runtime monitoring consists in monitoring the process' execution at runtime. It is generally more precise than static analysis but has a significant impact on the runtime performance. Some monitoring techniques do not modify the hardware, simply read some information provided by the existing hardware and take a decision using software logic. The information often comes from HPC that are present in most modern processors and may be combined with thresholds, heuristics, or machine learning processing to identify abnormal behavior [5]. Our solution can provide complementary hardware-level information to these software methods.

Modifying the hardware for runtime monitoring can provide more effective detection and reduce runtime overhead. Chen et al. [6] propose to modify some shared hardware components such as caches, in order to include a specific monitoring unit providing specific counters, then collect the data from these custom counters. Their approach enables to collect information that is directly useful to detect microarchitectural attacks and is more precise than HPC. However, the performed modification is only suited for a specific kind of known attacks and may be inefficient to cover another category of attacks.

Ozsoy et al. [7] proposed a solution that extracts information directly from the processor, combined with a hardware

machine learning unit for analysis. This work has some similarities with our proposal, but has put aside the main problem of updating this hardware when new attacks appear.

Using reconfigurable hardware with processors is not a new research topic. High-performance reconfigurable computing has been applied in many fields such as image processing and communication. As regard to the security domain, reconfigurable hardware has already been proposed for cryptography, for power and communication monitoring against hardware attacks [8], etc. However, to our best knowledge, no one has proposed the use of reconfigurable hardware to dynamically monitor the running software on a processor.

III. Architecture Design

A. Overview

The overall architecture that we propose is depicted in Fig. 1. The architecture consists of a main processor core, a reconfigurable detection module, interconnected by three information communication channels, and a trusted software core executed on the main processor.

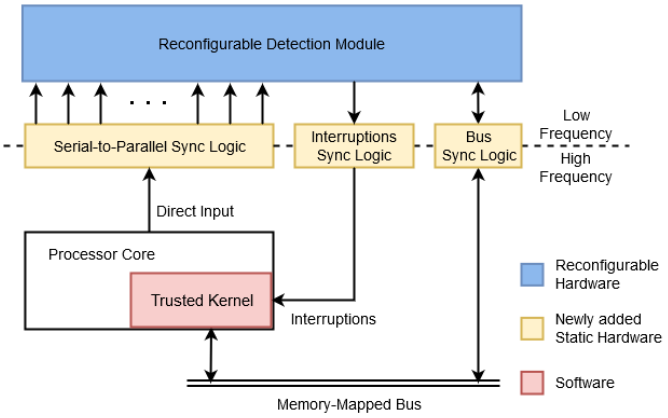

Fig. 1. Design of overall architecture.

The clock frequency of reconfigurable hardware such as FPGA is an order of magnitude slower than the clock frequency of the mainstream processors. Therefore, in order not to affect the frequency of the main processor, we let the processor runs at a normal high frequency while the *Reconfigurable Detection Module* runs at a lower frequency. All communication channels are equipped with synchronization logic for clock domain crossing. *Reconfigurable Detection Module* benefits from the parallel processing capabilities of hardware, to process simultaneously in one single clock cycle multiple information from multiple clock cycles of processor. Taking a modern processor with a clock frequency of 3.2 GHz, and a *Detection Module* able to process 16 clock cycles of information at the same time, we can use a FPGA running at a frequency of 200 MHz, which can be considered reasonable.

B. Communication channels

The communication between processor and *Reconfigurable Detection Module* goes through three channels. The first channel is a serial-to-parallel logic, into which high frequency serial data are buffered and converted into low frequency parallel data. It is used to transmit information extracted from the processor to *Detection Module* for analysis. The second communication channel is an interruption mechanism used to notify the processor of any abnormal situation. The last communication channel is a memory-mapped connection, providing peripheral-like access to the processor. It is used to read data, write some configurations, or even initiate a reconfiguration to the *Detection Module*.

Since processor microarchitecture cannot be modified during its entire lifetime, the type and number of processor internal state observations sent by the serial-to-parallel logic must be chosen wisely. For cache-based MTAs detection, the observations we chose are the instructions executed by the processor, and a valid bit to keep only one trace for each multi-cycle instruction. They are obtained directly from the execution stage of the processor pipeline. This choice is motivated by the fact the instructions actually executed can potentially help us to detect also transient execution, that is used by Spectre [9] for instance. In other words, the choice of using instructions as input increases the likelihood of detecting other software-based attacks that execute special instruction patterns. Other microarchitectural attacks that have physical access, such as electromagnetic side-channel attacks, are out of the scope of this paper.

C. Trusted software kernel

As hardware-level events alone are not enough to make complex decisions, we include a trusted software *Kernel* in our architecture. It may be embedded in the operating system kernel itself or an hypervisor, running on the main processor, and communicating with the *Detection Module*.

At runtime, the *Kernel* receives interruptions and other data from the hardware *Detection Module*. It synthesizes the situation with other known information of the running process, and determines whether the process may be a MTA. When a process is considered as suspicious, the kernel can react by forcing termination of the process or by enforcing isolation.

As the security requirements may vary from one process to another, the *Kernel* can adjust the threshold of the hardware *Detection Module* or reconfigure it according to the current needs. For example, when a cryptography process is running, it can improve detection sensitivity to prevent any possible leak of secret; when a benchmark process is running, it can set a non-interruption mode to prevent unnecessary alarms.

To take into account new attacks, the *kernel* is also able to perform a hardware reconfiguration of the *Detection Module*.

D. Detection Module

As discussed before, software monitoring is limited by its characteristics to only analyze serial instructions, and hardware monitoring is only designed to run at the same frequency as the processor. We want to take full advantage of the parallel nature of the hardware to maximize our detection capability in this parallel input context.

We target Flush+Reload [3] attack, the most common MTA, and propose a 3-stage detection logic as shown in Fig. 2.

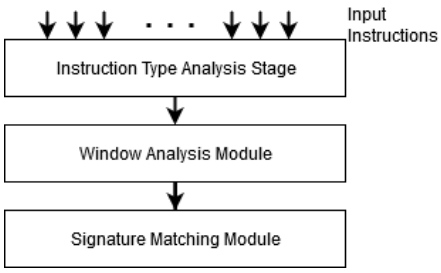

Fig. 2. Design of Detection Module.

In the instruction type analysis stage, the parallel inputs are analyzed with combined gate logic to deduce useful information. For example, this logic may evaluate whether the 16 input instructions include or not at least one instruction to get the value of the internal timer of the processor.

Then, we store the information obtained in a sliding window, and perform further analysis. For example, we may store values obtained from previous 32 clock cycles, and count how many timer instructions have been executed in this window.

At the end, we try to match information in the sliding window with attack patterns. For example, in Flush+Reload attack, the instruction that flushes the cache must be close to the instruction that gets the value of the timer, and this two-instructions pattern must be repeated multiple times during one attack. Thus, the *Signature Matching Module* counts the number of occurrences where both the cache flush and timer access instructions are present in a sliding window.

As this part of logic is fully reconfigurable, some other detection logic, such as machine learning algorithm purposely chosen, can be used instead.

IV. Implementation

Our design has been implemented on a Xilinx ML605 Evaluation Board (Virtex-6 FPGA) using the Orca RISC-V softcore processor [10]. The frequency difference between processor and reconfigurable logic has been modeled with a 1/16 clock divider. To enable dynamic hardware update, we put the *Detection Module* into a dedicated dynamically reconfigurable area. Up to now, the *Trusted Kernel* only manages interruptions, reads information from the *Detection Module*, and sets thresholds.

We adapted the Flush+Reload attack of Mastik toolkit [11] from x86 Instruction Set Architecture (ISA) to RISC-V ISA. In particular, the `rdtime` instruction was used instead of the `rdtscp` instruction. The cache flush instruction is not officially defined in RISC-V, nevertheless, we found that in Orca, when opcode is set to `MISC-MEM`, along with funct3 set to `REGION` and funct7 set to `CACHE-FLUSH`, a cache region is flushed. This special flush instruction was used instead of `clflush` of x86 ISA.

Our *Detection Module* focuses on instructions that access the timer Control and Status Register (CSR), including `rdtime`, and the cache flush instruction described below. By looking for timer/timer or timer/flush attack pattern, we successfully detected this Flush+Reload attack.

The synthesis of our *Detection Module* shows a maximum frequency of 271 MHz. In the fully implemented design, it occupies 235 registers and 400 LUTs. Static synchronization logic occupies additional 793 registers and 256 LUTs.

V. Conclusion and Future Work

In this paper, we discussed the feasibility of dynamic monitoring using reconfigurable hardware to detect cache timing attacks. We presented how we integrate reconfigurable hardware components into non-reconfigurable hardware and explained how can deal with different execution frequencies of the hardware.

For future work, we plan to improve our Detection Module, in order to be able to detect other cache-based microarchitectural attacks and their variants, and if possible, other software-based microarchitectural attacks as well. We continue our experimentation to evaluate the performance and area overhead of this solution. In the longer term, we want to take into account multithreading and context changes.

References

[1] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, no. 1, pp. 1–27, Apr. 2018.

[2] M. R. Fadiheh, D. Stoffel, C. Barrett, S. Mitra, and W. Kunz, "Processor Hardware Security Vulnerabilities and their Detection by Unique Program Execution Checking," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2019, pp. 994–999.

[3] Y. Yarom and K. Falkner, "FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *Proceedings of the 23rd USENIX Security Symposium*, San Diego, CA, Aug. 2014, pp. 719–732.

[4] G. Irazoqui, T. Eisenbarth, and B. Sunar, "MASCAT: Preventing Microarchitectural Attacks Before Distribution," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 377–388.

[5] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, Dec. 2016.

[6] J. Chen and G. Venkataramani, "CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, Dec. 2014, pp. 216–228.

[7] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2015, pp. 651–661.

[8] G. Gogniat, T. Wolf, W. Burleson, J.-P. Diguet, L. Bossuet, and R. Vaslin, "Reconfigurable Hardware for High-Security/ High-Performance Embedded Systems: The SAFES Perspective," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 144–155, Feb. 2008.

[9] 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 *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019, p. 19.

[10] VectorBlox, "Orca," Dec. 2019. [Online]. Available: https://github.com/VectorBlox/orca

[11] Y. Yarom, "Mastik: A Micro-Architectural Side-Channel Toolkit," 2016. [Online]. Available: https://cs.adelaide.edu.au/~yval/Mastik/