Final Scientific Report for the SCRATCHS project

Frédéric Besson, Pascal Cotret, Nicolas Gaudin, Guy Gogniat,
Jean-Loup Hatchikian-Houdot, Guillaume Hiet, Vianney Lapôtre, Pierre Wilke

1. Objectives

SCRATCHS (Side-Channel Resistant Applications Through Co-designed Hardware/Software) is a collaboration between researchers in the fields of formal methods (Epicure, Inria Rennes), security (SUSHI, CentraleSupélec Rennes) and hardware design (Lab-STICC). Our goal is to co-design a RISC-V processor and a compiler toolchain to ensure by construction that a security-sensitive code is immune to timing side-channel attacks while running at maximal speed. Our claim is that a co-design is essential to get end-to-end security: cooperation between the compiler and hardware is necessary to avoid time leaks due to the micro-architecture with minimal overhead.

2. Problem

The execution time of a program may vary depending on the values it manipulates, and the state of the processor. For instance, on a processor equipped with a memory cache, load instructions may be fast if the data is already in the cache (cache hit), or slow if data needs to be fetched from memory into the cache (cache miss).

An attacker able to measure the execution time of the instructions of the defender can learn information about its memory access pattern. The attacker can also be active and tamper with the state of the cache.

Indeed, the memory and cache are shared between the processes running on the same hardware. Even if the memory is shared, one process cannot directly read the data of other processes because of memory protection. Indeed, each process has an assigned memory space disjoint from the memory space of other processes (thanks to virtual memory and or physical memory protection units, e.g. PMP on RISC-V).

If a process tries to perform an instruction targeting memory outside of its memory space, this is prevented by memory protection. However, since they also share the cache, the attacker can load several distinct lines in a cache-set to force the evictions of the cache-lines of the defender in this targeted set. Then the attacker can wait and monitor the cache state to learn whether the defender loads again its line previously evicted. This is a rough description of the Prime+Probe attack, one of the many cache attacks the attacker can use to learn what addresses the defender accessed. Several cache attacks target encryption algorithms like AES or RSA, because these attacks enable to recover (parts of) cryptographic keys.

3. State of the art

Several countermeasures against timing attacks and, more specifically, cache attacks have been proposed in the literature.

3.1. Hardware countermeasures

3.1.1. Randomization

The idea is to randomize the relationship between the address accessed and the cache set where the data is stored. Hence, an attacker will only observe a cache set identifier, but will be unable to recover the address. This is enough to prevent simple attacks like Flush+Reload, but the mapping from cache sets to addresses can be recovered over time [1].

3.1.2. Resource partitioning

Cache attacks are possible because the attacker and the defender share the cache. One approach consists of dividing the resources between the processes such that the usage of a resource by one process cannot impact the resource's behaviour for the other processes [2], [3]. This is effective but costly: if we have two running processes, each runs with a cache that is half the size of the whole cache, increasing cache miss rate.

3.2. Software countermeasure: Constant-time Programming

Constant-time Programming is a programming discipline where the developer assumes that attackers will be able to observe all branch choices and targeted addresses of memory accesses of the program. Depending on how strict the developer is when programming (i.e. how strong he assumes the potential attacker will be), he also assumes that the attacker will also be able to measure the execution time of instructions. Thus, in CT programming, secrets are never used in a branching condition, as the address for a memory access, or as an operand for an instruction whose execution time varies depending on its operand.

This discipline can make programs harder to write, lead to bugs, and may be broken by compiler optimizations. Specialized compilers exist that preserve the constant-time property [4], [5], [6].

3.3. Conclusion

All the countermeasures in the state of the art involve a high overhead on the execution time. We propose a hybrid (software/hardware) solution with a new operation on the cache that allows to lock some data in the cache. With that mechanism in place, memory accesses to secret addresses can be performed safely, provided the address is locked in memory. This way, the attacker will always observe a cache hit, and learn nothing about the memory address actually accessed.

4. Results

We specify an extension to the RISC-V ISA in order to protect against timing attacks.

We offer the possibility for a process to lock some data in the cache, and to unlock them. While locked, data cannot be evicted from the cache, and all accesses to this data are necessarily cache hits, and therefore do not leak the address of the data being accessed.

We propose :

- a precise specification of the cache locking mechanism;
- a software simulator of a RISC-V ISA augmented with this locking mechanism;

- a formal proof that the software model of leakage is adequate to rule out any leakage in presence of any attacker program executing on the same processor;
- a hardware implementation based on the CV32E40P processor, with a L1 data cache.

5. Publications

- [7] N. Gaudin *et al.*, “Work in Progress: Thwarting Timing Attacks in Microcontrollers using Fine-grained Hardware Protections,” in *SILM*, 2023.
- [8] M. Peters *et al.*, “On The Effect of Replacement Policies on The Security of Randomized Cache Architectures,” in *ASIACCS*, 2024.
- [9] N. Gaudin, P. Cotret, G. Gogniat, , and V. Lapotre, “A Fine-Grained Dynamic Partitioning Against Cache-Based Timing Attacks via Cache Locking,” in *ISVLSI*, 2024.
- [10] J.-L. Hatchikian-Houdot, P. Wilke, F. Besson, and G. Hiet, “Formal Hardware/Software Models for Cache Locking Enabling Fast and Secure Code,” in *ESORICS*, in Lecture Notes in Computer Science. Springer, 2024.

6. PhD Theses defended

This project involved two PhD students, who successfully defended.

- Jean-Loup Hatchikian-Houdot defended his Ph.D. thesis, *Mécanisme de sécurité contre les attaques temporelles via une coopération entre logiciel et matériel embarqué*, on December 16th, 2024. (see https://theses.fr/2024URENS101)
- Nicolas Gaudin defended his Ph.D. thesis, *Partitionnement dynamique à grain fin contre des attaques par canaux auxiliaires en cache*, on December 18th, 2024. (see https://theses.fr/2024LORIS714)

7. Software

Each part of the project produced working artefacts that ensure the reproducibility of our research results.

On the software part, the following pieces of software are grouped in this git repository, that was also used as an artefact for our ESORICS paper [10].

- a modified RISC-V `binutils` toolchain to support the lock and unlock instructions.
- a modified version of the CompCert C compiler to support these instructions.
- the leakage simulator, one of the main results of Jean-Loup Hatchikian-Houdot’s thesis, that enables to study the hardware events that are observable by an attacker for a given program.
- a formalization in Rocq of a model of our RISC-V processor equipped with a lockable cache, which ensures that a program that we prove to be safe (Observationally Non-Interferent) at the software level, is indeed safe at the hardware level.

On the hardware part, the hardware implementation of the locking mechanism has been done on top of the CV32E40P processor.

8. Transfer

At the end of the SCRATCHS project, we applied and obtained funding for a follow-up action from CominLabs. This project, named LockOS, aims at making the implementation

more mature, and in particular, allow an operating system to run on our system and take advantage of the locking mechanism.

9. International

Nicolas Gaudin had a 3-month mobility (Jun'23 to Aug'23) at the Ruhr-University Bochum, Germany, in Tim Gueneysu's group (https://informatik.rub.de/seceng/personen/gueneysu/), which resulted in a publication [8].

Bibliography

[1] D. Skarlatos *et al.*, "MicroScope: Enabling microarchitectural replay attacks," in *ISCA*, 2019.

[2] A. Shafiee, A. Gundu, M. Shevgoor, R. Balasubramonian, and M. Tiwari, "Avoiding information leakage in the memory controller with fixed service policies," in *TACO*, 2015.

[3] L. Domnitser, A. Jaleel, J. Loew, N. B. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," in *ACM TACO*, 2012.

[4] J. B. Almeida *et al.*, "Jasmin: High-Assurance and High-Speed Cryptography," in *CCS*, 2017.

[5] S. Cauligi *et al.*, "FaCT: a DSL for timing-sensitive computation," in *PLDI*, 2019.

[6] G. Barthe, B. Grégoire, and V. Laporte, "Secure Compilation of Side-Channel Countermeasures: The Case of Cryptographic "Constant-Time"," in *CSF*, 2018.

[7] N. Gaudin *et al.*, "Work in Progress: Thwarting Timing Attacks in Microcontrollers using Fine-grained Hardware Protections," in *SILM*, 2023.

[8] M. Peters *et al.*, "On The Effect of Replacement Policies on The Security of Randomized Cache Architectures," in *ASIACCS*, 2024.

[9] N. Gaudin, P. Cotret, G. Gogniat, , and V. Lapotre, "A Fine-Grained Dynamic Partitioning Against Cache-Based Timing Attacks via Cache Locking," in *ISVLSI*, 2024.

[10] J.-L. Hatchikian-Houdot, P. Wilke, F. Besson, and G. Hiet, "Formal Hardware/Software Models for Cache Locking Enabling Fast and Secure Code," in *ESORICS*, in Lecture Notes in Computer Science. Springer, 2024.