A Fine-Grained Dynamic Partitioning Against Cache-Based Timing Attacks via Cache Locking

Nicolas Gaudin[1], Jeremy Guillaume[1], Pascal Cotret[2], Gogniat Guy[1] and Vianney Lapotre[1]

[1]UMR 6285, Lab-STICC, Univ. Bretagne-Sud, Lorient, France
[2]UMR 6285, Lab-STICC, ENSTA Bretagne, Brest, France

Abstract

Cache-based timing side-channel attacks represent a security threat for both high-end and embedded processors. As countermeasure to these attacks, previous works intoduced the lock and unlock instructions allowing a program to ensure constant-time accesses to cache. However, their implementation was still subject to cache-based attacks. In this paper, we propose a new implementation of these instructions, and experimentally demonstrate that our proposed solution defeats contention-based cache side-channel attacks such as Prime+Probe, while leading to a low impact on area overhead and performance efficiency of processes.

Introduction

Cache-based timing side-channel attacks such as Prime+Probe make it possible to retrieve confidential data, e.g., a secret cryptographic key. Numerous countermeasures have been proposed in the literature to thwart these attacks. From the hardware side, there are two main strategies. The first one, called *cache randomization*, consists in randomizing the address indexing in the cache to prevent the attacker from performing the Probe attack phase. However, data recovery is still possible with a suitable attack like Prime+Prune+Probe. The solution would be to periodically update the randomization, but this operation costs performance since it invalidates the whole cache.

The second strategy is *cache partitioning*, which consists in isolating the cache resources used by the victim from the ones used by other (potentially attacking) processes. Proposed solutions for partitioning, such as NOMO-CACHE, significantly reduce performance of other running processes, since they allocate a set of ways to sensitive applications. To remove this limitation, PLcache [1] has been proposed as a lightweight partitioning mechanism. It relies on a hardware-software collaboration by integrating two new instructions *lock* and *unlock*, enabling a victim to lock its data present in cache, and prevent it from being evicted by an attacker, or any other process.

PLcache reduces the impact on performance, however, the LRU state of locked data is still updated during memory access from any process. Additionally, in case a locked data is also the oldest data in a cache line, it can still be evicted during a memory access of the victim process itself. It has been demonstrated that these two characteristics keep this solution vulnerable to cache attacks [2]. Therefore, this paper 1) proposes new definitions of the lock and unlock instructions that remove these vulnerabilities, and 2) evaluates the impact of this countermeasure on performance. This study is done on the CV32E40P, which is a 4-stage in-order 32-bit RISC-V mono-core processor, including a single level of cache.

Proposed countermeasure

Our countermeasure against cache-based timing attacks is based on the concept of lock/unlock instructions proposed by PLcache [1], with the addition of a new LRU update mechanism whose diagram is shown in Fig. 1.

As illustrated in Fig. 2, calling the lock instruction has the same effect as calling the load instruction, with the addition of the LRU state of locked data being set to 0 (*update LRU locking* in Fig. 1). At each memory access, the cache ways with LRU state equal to 0 remain unchanged (*no update LRU*). Therefore, they are not considered to be evicted by the LRU way selection. At least not until they are unlocked and their LRU state is set back to a value different from 0 (*update LRU unlocking*).

To ensure minimum performance, at least one way always remains unlocked. If a request to lock the last way is received, an exception is raised.

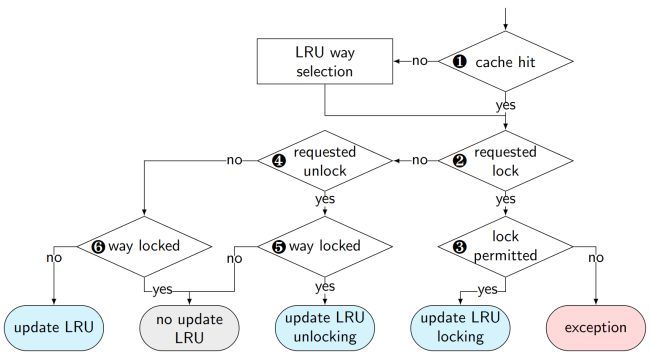


Figure 1: *Cache access procedure with locking mechanism.*

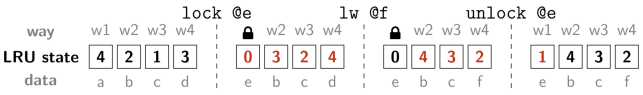


Figure 2: *Use-case: Behavior of LRU states considering our lock mechanism with N =4 ways.*

Impact on security

In this section, the impact of our countermeasure against the Prime+Probe attack targeting AES-128 is evaluated. It is assumed the attacker knows the binary of the program executed by the victim, the plaintext value and targets the value of the secret key. The attack consists in identifying which indexes of the Sbox table the victim has accessed, since these ones depend on the value of the secret key bytes.

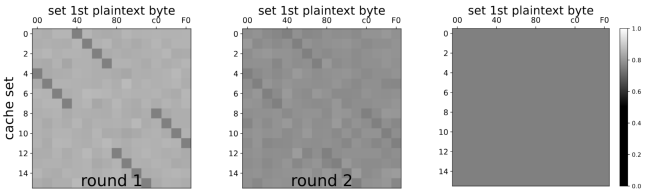
Fig. 3 shows the results of the Probe phase of the attack against the first key byte, (a) & (b) without and (c) with the locking mechanism. The darker the square is, the more time it takes for the attacker to access the data since the victim has evicted it from the cache (cache miss). Results highlight that the attacker cannot observe any difference in time to access the locked data, ensuring security against cache attacks.

Impact on performance

The proposed locking mechanism protects against cache attacks, but it is important to ensure it does not come with a significant reduction in performance. Thus, its impact is evaluated both on implementation overheads and on the loss of process performance due to cache contention caused by the locked ways.

Table 1 compares the resource needed, with and without the countermeasures, for both the Cache and CPU, on a Xilinx Kintex-7 chip with Vivado 2022.2. The overall overhead is lower than 3%.

Fig. 4 shows the performance of a list of processes (the Embench-IoT 1.0 suite) according to the number of caches lines that are entirely locked (3 ways out of 4 per line). The vertical blue and red dotted lines represent, respectively, the number of cache lines locked by a protected software implementation of the AES-128 and Camellia cryptographic processes. It can be



(a) Unprotected. (b) Unprotected. (c) Entirely locked.

Figure 3: *Prime+Probe attack against AES-128 S-Box. Key = 0x42.*

Table 1: *Post-implementation area on Kintex-7 FPGA.*

		Cache	CPU
Baseline	LUTs	980	5,661
	FFs	1,065	3,465
	BRAMs	8.5	8.5
Protected	LUTs	1,007 (+2.8%)	5,683 (+0.7%)
	FFs	1,077 (+1.1%)	3,481 (+0.3%)
	BRAMs	8.5	8.5

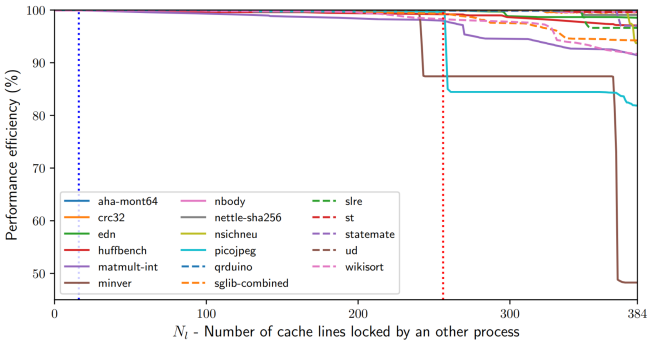


Figure 4: *Performance efficiency results.*

observed that the AES does not significantly impact processes performance, while Camellia, which requires to lock a higher number of cache lines, impacts a few processes. However, even in the worst-case scenario, performance remains close to 90% of initial values.

Conclusion and perspectives

In this paper, we proposed a fine-grained partitioning relying on a cache locking mechanism to thwart cache-based timing attacks, without significantly impacting processor performance.

As future works, we aim to study this countermeasure on a more complex processor such as the CVA6, with an OS running on it. Also, in the presence of an out-of-order processor, cases where unlock instructions are executed speculatively should be considered. Furthermore, this countermeasure does not protect against attacks such as Meltdown or Spectre. To overcome this limitation, we could investigate the combination of this countermeasure with others, like randomization.

References

[1] Zhenghong Wang and Ruby B Lee. "New cache designs for thwarting software cache-based side channel attacks". In: *Proceedings of the 34th annual international symposium on Computer architecture*. 2007.

[2] Nicolas Gaudin et al. "Work in Progress: Thwarting Timing Attacks in Microcontrollers using Fine-grained Hardware Protections". In: *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2023.