Secured Bus as a Countermeasure against Covert Channels: NEORV32 Case Study

Régis Leveugle[1], Nathan Hocquette[2], Charles Labarre[2], Romain Plumaugat[2], Loic Tcharoukian[2],Valentin Martinoli[3], Yannick Teglia[3]
[1] Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, 38000 Grenoble, France, firstname.name@univ-grenoble-alpes.fr
[2] Univ. Grenoble Alpes, Grenoble INP*, Phelma, 38000 Grenoble, France, firstname.name@grenoble-inp.org
[3] Cybersecurity Hardware lab, Thales DIS, La Ciotat, France, firstname.name@thalesgroup.com

***Abstract*—Cache-based Covert Channels are a significant threat for the security of systems built around a CPU. We evaluated a countermeasure taking advantage of a secured bus added in the CPU core with very little intrusiveness at the system level. This bus is used to bypass the data cache and other leaking elements when memory accesses are related to sensitive data or functions. We show on the NEORV32 CPU core that hardware overheads are negligible and a comparison with the use of the fence.t instruction illustrates noticeable gains in terms of secured execution speed.**

Keywords—hardware security, countermeasure, covert channels, secured bus

I. Introduction

In the last decades, and especially since the hype created by the publication of Spectre and Meltdown in 2018, a lot of work has been dedicated to demonstrating and countering so-called micro-architectural attacks in CPU-based systems [1-2]. Such attacks are implemented by software but take advantage of hardware mechanisms to recover sensitive data manipulated by a target victim. They mostly leverage the resources implemented for fast memory access, speculation or Out-of-Order execution. In this short paper, we will focus on attacks exploiting the timing differences between a hit and a miss when accessing data in L1-D cache. This is the root of several Covert Channels [3], used as a final step in many micro-architectural attacks.

We evaluate here the efficiency of a countermeasure based on a secured bus added in the CPU core to bypass shared micro-architectural resources when critical data memory accesses occur [4]. We consider as an example a widely used Covert Channel technique called Prime+Probe (P+P), with the same threat model as in [5], but the studied protection may be used against other similar exploits. We have chosen the NEORV32 CPU included in [6] as the experimental vehicle, especially for its small footprint. We expect larger CPU cores to exhibit smaller hardware overhead percentages.

Major related points in the state-of-the-art are recalled in Section II. Our countermeasure principles and the main test vehicle characteristics are summarized in Sections III and IV. The implementation is briefly described in section V and results are discussed in section VI.

II. Related State-of-the-Art

The studied countermeasure is designed to bypass the resources that can be exploited by variants of micro-architectural attacks such as the MDS attacks [7] and Covert Channels including Prime+Probe, but also e.g., Flush+Reload or Flush+Flush that provide more precise outcomes but require satisfying stronger hypotheses. The P+P Covert Channel used in this study consists in three distinct phases. First, the Spy process initializes the whole L1-D with its own data (Prime phase). Then, the target application runs time-sharing the cache with the Spy and we assume as in [6] the worst case of a victim that is infected by a Trojan, actively trying to transfer some secret by evicting lines in the data cache. These evictions can e.g., encode a piece of data as being the number of evicted lines. After the application resumes, the Spy process tries to recover the transmitted secret by accessing the L1-D as during the Prime phase, identifying the number of lines evicted by the victim thanks to the longer access time required in case of miss.

Protections can be based either on hardware modifications or on software-controlled operations. Pure hardware protections can quickly become expensive in terms of resources but pure software approaches have limited possibilities and most often a large impact on the performance-oriented optimizations. In [5], the fence.t instruction was proposed to reset all leaking resources to a state that is independent of previous execution history, including invalidating the L1-D lines. This implies some hardware modifications to implement the new instruction, requires executing the instruction after each sensitive computation and before the context switch, but also reduces the time penalty of the reset compared to a pure software approach, while allowing access to all leaking resources. With this approach, the Spy can only record cache misses and is not able to recover any secret. However, all data are erased (sensitive or not), even in the case there was no ongoing attack. Processes running in the same time frame than the protected application can therefore suffer a noticeable slow-down due to the systematic invalidations.

III. Countermeasure: Principles and Main Advantage

The proposed approach also takes advantage of custom instructions, but does not reset any resource. Instead, a secured bus is added in the CPU core to bypass all leaking resources, including L1-D, only when sensitive information is concerned. This bus is activated through custom Load/Store instructions LOAD_SEC and STORE_SEC so that the L1-D contents is not modified during the execution of a sensitive process and thus in case of attack the Spy cannot exploit any timing difference to recover a secret. The other processes, when activated again, can benefit from the data previously in the cache if not modified in the meantime by a Prime phase.

Several possibilities of architectural modifications are identified in [4], with different memory organizations. In particular, the secured bus can be (1) directly connected between the CPU pipeline and a second port of the main memory, or (2) connected to a dedicated memory, avoiding the dual port requirement for the main memory. In this work, we focus on modifications having no impact on the system architecture and connections outside the CPU+Cache area.

* Institute of Engineering Univ. Grenoble Alpes

We therefore limit the additional bus to the transfers within the processor chip. The leaky structures are bypassed without any hardware modification nor flush of these resources (thus avoiding performance penalty for non-secured operations). The standard bus and the secured bus are multiplexed at the periphery of the processor chip, allowing pin-level compatibility between versions with and without the security countermeasure and making the solution compatible with an implementation on a standard FPGA-based board. The main memory is not modified, but cache levels implemented outside the processor chip should be prohibited to avoid last level cache attacks.

Compared to the fence.t instruction, our solution can noticeably reduce the performance penalty if the number of critical data memory accesses to the same address remains small. There is no time lost in invalidating the cache so the penalty only comes from the systematic main memory accesses when the same address is read several times during the execution. For most sensitive data, it is possible to avoid writing intermediate results directly in memory so in that case the penalty is null, the first read and the last write being anyway from or to the main memory. When writes are unavoidable in memory, the penalty can remain small compared to the number of main memory reads induced for other processes by a systematic flush of the whole cache.

IV. Case Study: NEORV32 CPU

The NEORV32 CPU [6] is an open-source RISC-V core, part of a highly configurable microcontroller focusing on area optimization and safety rather than computing speed. In addition to its small footprint, this CPU has helpful characteristics from a security point of view: careful management of incorrect instructions, exceptions triggered in case of unexpected situations, in-order, without speculative executions, no shared internal buffer leaking critical information and Write-Through coherency policy.

V. Implementation of the Countermeasure

The NEORV32 CPU has been modified to add the LOAD_SEC and STORE_SEC instructions. The encoding was chosen according to the custom0 and custom1 opcodes defined in the RISC V ISA, leading to only one bit differing from a standard Load or Store and full compatibility with the different data formats (word, half, unsigned half, byte and unsigned byte). All the non-regression tests available in the NEORV32 repository have been run successfully.

The toolchain has been extended in two ways. First, a tool called Secure Convert has been developed. It takes as inputs an ELF file and a list of sensitive functions and automatically converts the standard Load and Store opcodes to the secured versions for the selected functions. Only compressed instructions are not supported at that time. The standard tool chain has also been updated to consider a "secure" attribute to be added in sensitive function prototypes. This attribute is recognized by gcc, gdb and objdump among others.

VI. Results

A. CPU Overheads

The modified CPU description has first been synthesized on an ASIC library to evaluate the overheads of the additional instructions and bus. Results shown in Table I clearly demonstrate the small impact of the proposed countermeasure. The CPU and its caches have then been implemented in an Artix-7 FPGA chip on a Basys 3 board. Table I shows the overheads reported after placement and routing. As previously mentioned, with the architectural choices made for the secured bus implementation, no modification was required on the board. As in the previous case, overheads are negligible and the power consumption was even evaluated slightly better with the secured bus due to some slight differences in routing.

TABLE I. Overheads

Synthesis on ASIC Library				
Clock frequency		**Area**	**Power**	
+0%		+0.09%	+0.2%	
P&R on Artix-7 FPGA				
Clock frequency	**Slices**	**Slice LUTs**	**Slice Registers**	**Power**
+0%	+2.9%	+1.3%	+0.4%	-1.4%

B. Prime+Probe

Two bare-metal versions of the P+P attack have been implemented with a pseudo-scheduling of the three phases. They have been run with and without activation of the secured bus. As expected, they recovered the encoded secret when the secured bus was not used, but were otherwise unsuccessful.

C. Experimental Comparison with fence.t

In the NEORV32 CPU, L1-D is the only source of leakage to bypass or reset. A Write-Through policy is used for Store operations. It is generally preferred to Write-Back when security is a concern and it is also the most favorable situation for the fence.t approach, since there is no dirty state to write back when invalidating the cache lines. The L1-D is small (8 lines of 64 bytes) and its full invalidation requires only 3 cycles. The main memory is very fast and a full line is read in 36 cycles. The difference in computation time has been evaluated using the second version of the P+P attack, with no other process involved and potentially impacted by the invalidation. Despite these very favorable conditions, the execution time overhead of fence.t was 2347 cycles while the secured bus only induced a 289-cycle penalty, so 8x less.

VII. Conclusion and Perspectives

This case study clearly shows the small overheads induced by the proposed countermeasure even in a very small core and the potential gains in performance compared to fence.t if sensitive functions are repeatedly called. The approach can be applied to any ISA allowing to define custom instructions and any modifiable core. Further work includes implementing the approach in more complex cores with more leaky resources.

References

[1] D. A. Osvik, A. Shamir, E. Tromer,. "Cache attacks and countermeasures: the case of AES,". In: Pointcheval, D. (eds) Topics in Cryptology – CT-RSA 2006. Lecture Notes in Computer Science, vol 3860. Springer, Berlin, Heidelberg.

[2] C. Canella et al., "A systematic evaluation of transient execution attacks and defenses," 28th USENIX Conference on Security Symposium (SEC'19),:August 2019, pp. 249–266

[3] Q. Ge, Y. Yarom, D. Cock, G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," J Cryptogr Eng 8, 1–27 (2018)

[4] European Patent Application EP4276633, published on Nov. 15, 2023

[5] N. Wistoff et al., "Prevention of microarchitectural covert channels on an open-source 64-bit RISC-V core," 4th Workshop on Computer Architecture Research with RISC-V (CARRV), May 29, 2020

[6] https://github.com/stnolting/neorv32

[7] https://mdsattacks.com/