A novel cache based on dynamic mapping against speculative execution attacks

Dehua Wu[1,2], Wan'ang Xiao[3,4], Shan Gao[1,2], and Wanlin Gao[1,2,*]

[1]Key Laboratory of Agricultural Informatization Standardization, Ministry of Agriculture and Rural Affairs, China Agricultural University, Beijing 100083, P. R. China
[2]College of Information and Electrical Engineering, China Agricultural University, Beijing 100083, P. R. China
[3]Institute of Semiconductors, Chinese Academy of Sciences, Beijing 100083, P. R. China
[4]Center of Mater. Sci. and Optoelec. Engineer. & School of Microelec., Univ. of Chinese Academy of Sciences, Beijing 100049, P. R. China

Abstract. The Spectre attacks exploit the speculative execution vulnerabilities to exfiltrate private information by building a leakage channel. Creation of a leakage channel is the basic element for spectre attacks, among which the cache-tag side channel is considered to be the most serious one. To block the leakage channels, a novel cache applies Dynamic Mapping technology, named DmCache, is presented in this paper. DmCache applies a dynamic mapping mechanism to temporarily store all the cache lines polluted by speculative execution and keep invisible when accessing. Then it monitors the head of the reorder buffer to determine which polluted cache line can become visible. In this paper, we demonstrated that Spectre attacks exerted no impact on a processor system equipped with DmCache based on the analysis of the processor's circuit behaviour, which equipped with the DmCache and under the Spectre attack.

Keywords: Microprocessors, Microcomputers, Spectre attacks.

1 Introduction

The design strategy of the speculative execution is widely applied in modern processors. This is an important way to improve the performance of processors. Using branch instruction predictor, the path of the processor is going to execute along can be predicted, which enable the processor to speculatively execute subsequent instructions without any security checks. If the prediction is correct, the result of speculative execution can be committed. Otherwise, the result is squashed. Compared to stalling the pipeline and waiting for the branch instructions to be counted out, speculative execution can improve the efficiency of execution.

However, it is due to speculative execution that processors are placed under great security risks. Normally, memory isolation is an important strategy to secure computer systems. By endowing protection domains to different processes, the system can protect the private information from malicious accessing. Nevertheless, recent research has shown that the

* Corresponding author: wanlin_cau@163.com

attackers glean information in an arbitrary memory location by initiating spectre attacks [1]. These exploit the central processing unit (CPU) speculative execution vulnerability to execute erroneous instructions (called transient instructions), which bypass the security checks.

The branch predictor can be mis-trained to force victim programs to speculatively execute down the paths specified by the attacker. Although the incorrect speculation can be squashed, the attackers can use *load* instructions to modulate the execution results into the data cache, resulting in measurable side effects in the cache. In the later phase of attacks, the attackers learn about leaked private information by probing the modification of the cache.

Many mitigations have been proposed against spectre attacks. Broadly speaking, there are two classes of these mitigations, namely hardware protection, such as Dynamic Allocated Way Guard (DAWG) [2], Safespec [3], InvisiSpec [4], and software protection, such as OO7 [5] and KASLR [6]. However, these efforts are mainly focused on high-performance processors, such as desktop or server class, which are multi-core and multi-level caches architecture. But due to limited computing resources, complex protection mechanism may cause the performance loss of processors for the resource-constrained processors.

Given these issues, a cache based on dynamic mapping mechanism (named DmCache) is proposed to protect processors from Spectre attacks. This paper focuses on the research about how to optimize the cache design to minimize the performance overhead in the resource-constrained processors chip. The protection strategy of DmCache is temporarily storing the new data cached by transient instructions (named updated data) and back up the original data (named replaced data). Obviously, setting up backup storage units for each cache line is a simple implementation of this strategy. However, this method can increase the size and power consumption of the processor die, which is not considered in practice. Therefore, this paper addresses this issue by remapping the data and address line, which is the minimum storage unit of data/address of cached data in DmCache. The remapping relationship can be restored in case of the mis-prediction.

Once miss occurs, the DmCache uses idle data line and address line to store the address and data parts of the updated data. If the instruction is squashed, the reassign cache line corresponding to this instruction stays invisible, as a result it cannot be accessed by subsequent instructions, and the original cached data is retained. If the instruction is retired, the cache line becomes visible. In this way, DmCache keeps the state of the cache unchanged to block information leakage channels.

To summarize, this paper makes the following contributions:

A novel cache against spectre attacks is proposed. By applying this cache, the channel of information leakage created by attackers can be blocked, which provides protection against spectre attacks.

We gave a detailed explanation about how DmCache protects data cache from suffering spectre attacks. A processor simulation platform with speculative execution vulnerabilities is constructed, where the DmCache is tested under spectre attacks with two cache tag state-based channel building technology.

The paper is organized as follows. Section II gives an introduction of the related work. The hardware design implied by DmCache is presented in Section III. The design of the processor simulation platform is detailed in Section IV. Security analysis and evaluation are the subjects of Section V. Section VI provide the conclusions.

2 Related work

Tomasulo algorithm [7] is a hardware algorithm for improving the performance of the processor and have been widely used. It enables out-of-order execution and allows malicious instructions to be executed in the speculative execution window. Spectre attacks exploit the

fact that the Tomasulo-based processor allows the speculative load instructions to modify the cache state without any security checks.

2.1 Speculative execution window

At the beginning of an attack, the attackers need to build a window of speculative execution to allow malicious code to be executed. The attackers mis-train a branch predictor to force the processor to open a speculative execution time window, during which the malicious code can be speculatively executed without any security checks. To maximize the speculative execution window, the data hazard is set by the attacks to block instruction execution flow. This results in instruction to be executed one by one and a large number of instructions are blocked in the reservation station. However, malicious instructions that have no data hazard with other instructions can be executed in advance.

2.2 Cache tag state-based leakage channel

Due to the speculations that execute transient instructions, their execution results would be squashed. Spectre attacks and their variants exploit measurable side-effects in processor microarchitecture caused by the speculative execution as the information leakage channel. A lot of processer component can be exploited to build leak channels, such as TLB, data cache, instruction cache, etc. However, spectre attacks based on data cache are a typical and mature attack method, which was considered to be a serious threaten. Many existing technologies can be used to build the leak channels needed for Spectre attacks, such as Flush+Relaod [8], Prime+Probe [9], etc.

Flush+Reload and Prime+Probe are popular ways to build the cache tag state-based channel. Using the *clflush* instruction in the x86 ISA, the Flush+Reload attacker evicts the cache line from all the cache levels, including from the shared Last-Level-Cache (LLC). Then, the victim program is allowed to execute a small number of instructions, making the data cache to record the memory location accessed by the victim program. The attacker can learn the private information by probing whether the data in these cache locations is reloaded into the cache.

Unlike Flush+Reload, Prime+Probe learns private data by monitoring the data replaced by the victim program. In the prime phase, the attacker accesses cache lines so as to fill the cache set with known data. Then, the attacker forces the victim program to execute a small number of instructions. In the probe phase, the attacker learns the memory address accessed by the victim program through measuring the access time.

In fact, a lot of methods can be used for building the cache tag state-based channel, such as Evict+Reload [10], etc. However, essentially, there are only two technologies that are suitable to modify the state of the cache as a leakage channel, i.e. evict the replaced data out of the cache and loading the updated data into the cache.

Thus, this paper focuses on these two cache tag technology. We make conservative assumptions that all traces left by uncommitted load instructions in the cache, are likely to be the channels for information leakage, and all these traces should be hidden. Therefore, this paper focuses on the security issues about single-core processors based on the Tomasulo algorithm with malicious modification in the cache.

3 Dynamic mapping-based cache

The detailed design of DmCache is introduced in this section. First of all, the building blocks of DmCache are elaborated. Then, we introduce three design ideas, i.e. temporary storage, dynamic mapping and blacklist mechanism, which are applied in DmCache.

3.1 Building blocks of DmCache

As shown in the figure 1, the DmCache is composed of Address Table, Conversion Table, Valid BitTable, Replacement Strategy Module, Data Table and Write Back Buffer.

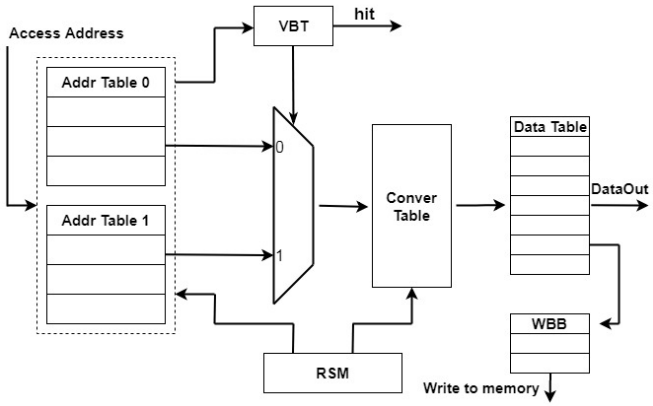

Fig. 1. The Architecture of DmCache.

Address Table: The Address Table stores the addresses parts of the cached data.

Data Table: The value part of the cache is stored in the data table.

Conversion Table: The Conversion Table stores the mapping relationship between the address line and the data line.

Valid BitTable (VBT): The current status of each address line is stored in the VBT. Each address line is represented by a bit in the VBT, where the '1' is valid and the '0' is idle.

Replacement Strategy Module (RSM): When DmCache is missing, the RSM can select a suitable replacement address/data line from the address/data table to store the address/ value.

Write Back Buffer (WBB): The data, needed to be written back to the memory, are temporarily stored in WBB. When the cache receives a write signal, DmCache performs a check to see whether the requested address hits a memory location in the DmCache. If it hits, the written data are immediately written back to the WBB. Then, the WBB writes the data back to the main memory in sequence. If the cache is missed, the data written back can be written directly into main memory and backed up in DmCache at the same time.

3.2 Temporary storage

DmCache needs to temporarily store the address and value of the cache to prevent attackers from building a covert channel. In order to implement temporary storage, the redundant storage unit is designed in DmCache. Because the capacity of address and value is different, the structure of the redundant address table and data table are different too. Thus, a quantity relationship is elaborately designed for the number of address lines and data lines. In this study, the number of cache lines in DmCache is assumed to be n.

Two addresses of the updated data are placed in DmCache. Thus the number of address line is $2n$. These address tables are used alternately, one of which is a backup of the other.

That is, when one address line is in a valid state, another is idle state. VBT is used to record whichever address line is valid at that time.

The value of the updated data also needs to be temporarily stored. However, if the data table is designed with the same structure as the address table, it can greatly increase the sizes of the chip die. Therefore, the number of entries in the data table is increased by a specified number compared with the number of cache lines, that is, the number of data lines is $(n + x)$.

3.3 Dynamic mapping mechanism

Using the dynamic reorganization of the address line and the data line, DmCache establishes the dynamic mapping mechanism. There are two parts of a cached data (the value and the address) that need to be stored. For address part, DmCache endows each address line with a status register, stored in the VBT, to indicate which address line is valid. When the cache misses, the DmCache chooses a cache line to store the data. As each cache line is equipped with two address lines, the DmCache chooses the idle address line to store the address parts of updated data by querying VBT. Concerning the value parts, the RSM chooses a data line in the redundant data table to store parts of the updated data, instead of evicting the original data. Therefore, the dynamic mapping mechanism can back up replaced data and temporary storage of updated data.

As the address line and the data line are selected by their RSM separately, the storage units in each cache line are dynamic. Therefore, this way of storing data dynamically needs to maintain the mapping relationship between the address table and data table to ensure the accuracy of cached data. To ensure the correct composition of the data line and the address line, the conversion table is designed to record correspondence between them. Once the memory location that requested by the processor hits an address line, DmCache determines which data line is corresponding to this address line by querying the conversion table.

3.4 Blacklist mechanism

Any modification caused by the transient instruction that is not committed yet, cannot be visible immediately. The blacklist mechanism is applied to hide this cache line, which prevents subsequent instructions from accessing. Any cache line recorded in the blacklist can be disallowed to be hit.

In DmCache, the VBT works as the blacklists functionally. When the cache data is temporarily stored, the VBT cannot be updated immediately. Instead, the VBT monitors the head of the ROB to determine which instructions are submitted. Once the instructions reach the head of ROB, the VBT updates immediately to become valid. At this moment, modification of the micro-architecture state needs to be visible. Otherwise, the polluted cache line remains in idle, resulting in invisible to modification.

4 Experimental setup

In this section, a simple implementation of DmCache is introduced through a design example. A processor based on the Tomasulo algorithm was integrated with DmCache. First, an RISC-V instruction set-based processor with speculative execution vulnerability is implemented. Then, the integration design of DmCache and processor is introduced.

4.1 Speculative processor design

A processor with speculative execution vulnerability was established. This processor, which had a six-stage pipeline and five execution components, was an out-of-order execution processor based on RISC-V instruction set.

In order to simplify the implementation of the spectre attack, a static branch predictor was used, which always predicted not-taken. Therefore, we do not need to mis-train branch predictor in the subsequent simulation.

4.2 Processor system integrated with DmCache

The integration design of DmCache and processor is introduced in this section.

Allocation: A table (called instruction recorder) is used to record the address line that is polluted by transient instruction. Once an instruction is issued from the dispatch stage, an entry in the instruction recorder is allocated to this instruction. Renamed addresses are used as the identifiers of instructions, which are unique tags in that processor. The instruction recorder uses the register renamed address of each instruction as an address reference.

Record: When the instruction is in the execution phase, it records the address lines polluted by this instruction.

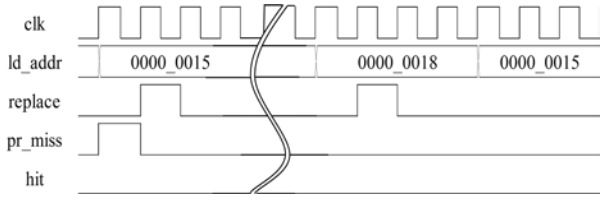

Fig. 2. DmCache under attacks with Flush+Reload.

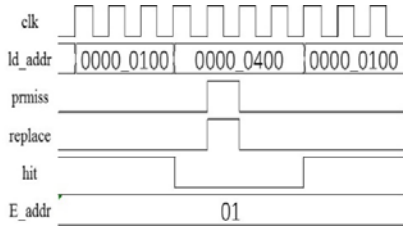

Fig. 3. DmCache under attacks with Prime+Probe.

Update: The DmCache monitors the head of the recorder buffer to determine which polluted cache line can become visible. When the instruction is committed, DmCache queries the instruction recorder to get the address line polluted by this instruction and updates the VBT with the address line. At that moment, the cache line loaded by the transient instruction becomes visible.

Table 1. The signal notations used in analysis

signal notations	Meaning and explanation
clk	Clock Signal
ld_addr	Address of load instructions
pr_miss/prmiss	Branch instruction prediction failure
replace	Cache replace happen
hit	Cache hit happen
E_addr	The address of address line to be evicted

5 Evaluation

In this section, DmCache is evaluated. Based on the work above, the security of DmCache was tested under spectre attacks with two cache tag state-based channel building technology. Through analyzing the behavior of DmCache, it was demonstrated that spectre attacks cannot work in a system equipped with DmCache.

For Spectre attacks using Flush+Reload, the VBT cannot be updated if uncertainty exists as to whether the instructions can be updated. As shown in figure 2, a malicious program tries to build a leak channel by loading the address of 0x0000_0015. After that, the attackers probe the address of 0x0000_0015, but fails because the 0x0000_0015 is hidden by the VBT. Thus the modification caused by transient instructions cannot be visible, resulting in the attackers not being able to build covert channel.

In the simulation under spectre attacks using Prime+Probe, the behavior of DmCache is shown in figure 3. The E_addr indicate the replaced cache line, which is calculated by the RSM. In order to simplify the simulation, RSM is modified to always choose the address of (0,1) to be evicted. When the 0x0000_0400 is loaded by transient instruction, the replaced data is not evicted from the cache. Instead, the updated data is loaded temporarily into the redundant storage units. So the attackers cannot build the covert channel.

6 Conclusion and future work

In this study, DmCache is proposed to protect the resource-constrained chips from spectre attacks based on the cache tag state-based channel. Applying the dynamic mapping mechanism, DmCache backs up the replaced data and stores the updated data temporarily, which blocks the cache tag state-based channel. In order to test the security of DmCache, an instance of DmCache is implemented to protect the data cache and tested under the spectre attacks with two different cache state-tag leak techniques. The test results show that the spectre attack cannot build a leak channel in a processor with DmCache, protecting the processor from spectre attack.

The authors would like to thank the Professor Wanlin Gao and Wan'ang Xiao for funding authorization. Additionally, we are fortunate and thankful for all the advice and guidance we have received during this work.

References

1. P.Kocher et al., "Spectre attacks: Exploiting speculative execution," in Proc. 40th IEEE Symp. Security Privacy, vol. 1801, May 2019, Art. No. 01203.
2. V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, J. Emer, "Dawg: A defense against cache timing attacks in peculative execution processors," in 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), vol. 2018, 2018, pp. 974–987.
3. K. N. Khasawneh, E. Mohammadian Koruyeh, C. Song, D. Evtyushkin, D. Ponomarev, and N. AbuGhazaleh, "SafeSpec: Banishing the Spectre of a Meltdown with Leakage-Free Speculation," ArXiv e-prints, 2018.
4. M. Yan, J. Choi, D. Skarlatos, A. Morrison, C. W. Fletcher, and J. Torrellas, "InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy," in Proceedings of the 51th International Symposium on Microarchitecture, ser. MICRO'18, 2018.

5. G. Wang, S. Chattopadhyay, I. Gotovchits, T. Mitra, A. Roychoudhury, “oo7: Low-overhead Defense against Spectre Attacks via Binary Analysis,” arXiv preprint arXiv:1807.05843, 2018.
6. D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, and S. Mangard, “Kaslr is dead: long live kaslr,” in International Symposium on Engineering Secure Software and Systems. Springer, 2017, pp. 161–176.
7. R. M. Tomasulo, “An Efficient Algorithm for Exploiting Multiple Arithmetic Units,” IBM Journal of Res. and Dev., 1967.
8. Y. Yarom and K. Falkner, “FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack,” in Usenix Conference on Security Symposium, 2014.
9. D. A. Osvik, A. Shamir, and E. Tromer, “Cache attacks and countermeasures: the case of AES,” in Cryptographers Track at the RSA Conference. Springer, 2006, pp. 1–20.
10. Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Cross-tenant side-channel attacks in paas clouds,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS ’14. New York, NY, USA: ACM, 2014, pp. 990–1003. [Online].Available: http://doi.acm.org/10.1145/2660267.2660356