Jamais Vu: Thwarting Microarchitectural Replay Attacks
Extended Abstract

Dimitrios Skarlatos[†], Zirui Neil Zhao[†], Riccardo Paccagnella, Christopher Fletcher, Josep Torrellas
University of Illinois at Urbana-Champaign
{skarlat2, ziruiz6, rp8, cwfletch, torrella}@illinois.edu
[†]Authors contributed equally to this work.

1. Motivation

The microarchitecture of modern computer systems creates many side channels that allow an attacker running on a different process to exfiltrate execution information from a victim. Indeed, hardware resources such as caches [16, 24, 15, 11, 22, 23], TLBs [10], branch predictors [7, 2, 8], load-store units [12], execution ports [5, 3, 9], functional units [4, 3], and DRAM [17] have been shown to leak information.

Luckily, a limitation of these microarchitecural side channels is that they are often very noisy. To extract information, the execution of attacker and victim threads has to be carefully orchestrated [23, 15, 16], and often does not work as planned. Hence, an attacker needs to rely on many executions of the victim code section to obtain valuable information. Further, secrets in code sections that are executed only once or only a few times are hard to exfiltrate.

Unfortunately, a recently-introduced type of attack called Microarchitectural Replay Attack (MRA) [19] is able to eliminate the measurement variation in (i.e., to denoise) most microarchitecural side channels. This is the case even if the victim code section is executed only once. Such capability makes the plethora of existing side-channel attacks look formidable and suggests the potential for a new wave of powerful side channel attacks.

MRAs use the fact that, in out-of-order cores, pipeline squashes due to events such as exceptions, branch mispredictions, or memory consistency model violations can force a dynamic instruction to re-execute. Hence, in an MRA, the attacker actuates on one or multiple instructions to force the squash and re-execution of a victim instruction *V* multiple times. This capability enables the attacker to cleanly observe the side-effects of *V*.

MRAs are powerful because they exploit a central mechanism in modern processors: out-of-order execution with in-order retirement. Moreover, MRAs are not limited to speculative execution attacks: the instruction *V* that is replayed can be a correct instruction that will eventually retire. Finally, MRAs come in many forms. While the first MRA [19] exposed the side effects of *V* by repeatedly causing a page fault on an older instruction, the same result can be attained with other events that trigger pipeline flushes. In the Optional Appendix, we have uploaded a Proof-of-Concept of a memory consistency model violation MRA that we have developed.

To thwart MRAs, one has to eliminate instruction replay or at least bound the number of replays that a victim instruction *V* may suffer. The goal is to deny the attacker the opportunity to see many re-executions of *V*.

2. Limitations of the State of the Art

As MRAs were introduced last year, there is currently no defense against MRAs.

There are several mechanisms in the literature that, although not designed as defenses against MRAs, can mitigate specific instances of MRAs. For example, page fault protection schemes [18, 14, 13, 6] can be used to mitigate MRAs that rely on page faults to cause pipeline squashes. The goal of these countermeasures is to block controlled-channel attacks [21, 20] by terminating victim execution when an OS-induced page fault is detected. The most recent of these defenses, Autarky [14], achieves this through a hardware/software co-design that delegates paging decisions to the enclave. However, MRA attacks that rely on events other than page faults to trigger pipeline squashes (e.g., memory consistency model violations, other exceptions, branch mispredictions, and interrupts) would overcome these point mitigation strategies.

There is no defense that addresses the root cause of MRAs, namely that instructions can be forced to re-execute to denoise a side channel.

3. Key Insights

This paper presents *Jamais Vu*, the first mechanism designed to thwart MRAs.

The simple idea of *Jamais Vu* is to first record the victim instructions *V* that are squashed. Then, as each V instruction is re-inserted into the Reorder Buffer (ROB), *Jamais Vu* automatically places a fence before it to prevent the attacker from squashing it again. In reality, pipeline refill after a squash may not bring in the same instructions that were squashed, or not in the same order. Consequently, *Jamais Vu* has to provide a more elaborate design.

To understand the variety of MRAs, this paper first analyzes some of the characteristics of MRAs. These characteristics and why they matter are explained in Table 1 in the paper.

Then, the paper considers how to design the defense. A highly secure defense against MRAs would keep a fine-grain record of all the Victim dynamic instructions that were squashed. When one of these instructions would later attempt to execute, the hardware would fence it and, on reaching retirement, remove it from the record. In reality, such a scheme is not practical due to storage requirements and the difficulty of identifying the same dynamic instruction.

Hence, *Jamais Vu* proposes three classes of schemes that discard this state early: *Clear-on-Retire*, *Epoch*, and *Counter*. The schemes differ on when and how they discard the state. They effectively provide different trade-offs between execution overhead, security, and implementation complexity.

The *Clear-on-Retire* scheme discards any Victim information as soon as the program makes forward progress — i.e., when the instruction that caused the squash retires. The *Epoch* scheme discards the state when the current "execution locality" or *epoch* terminates, and execution moves to another locality. We developed an analysis pass to mark epochs in an x86 executable. Finally, the *Counter* scheme keeps the state forever, but it compresses it into a single counter per static instruction. The counter keeps the difference between the number of squashes and the number of retirements of any dynamic instance of that static instruction. *Counter* strives to keep such counter low (but can never be less than 0). Each of these policies to discard or compress state creates a different attack surface.

Table 2 describes the salient points of each of the schemes and their pros/cons. Moreover, Table 3 assesses the relative security of the schemes by comparing their worst-case squash count for some representative code snippets.

The paper then describes an implementation of each of these schemes and some design variations. The *Clear-on-Retire* and *Epoch* schemes can be easily implemented with a Bloom filter and with counting Bloom filters, respectively (Figures 3 and 4). The *Counter* scheme is implemented by storing the per-instruction counters in memory and caching the counters in a special Counter Cache (Figure 6).

The paper describes a program analysis pass that we developed. The pass places "start-of-epoch" markers in an executable, for use in the *Epoch* scheme. The pass accepts as input a program in source code or binary.

4. Main Artifacts

The three key artifacts in the paper are three hardware designs of our *Jamais Vu* approach to thwart MRAs, and their hardware implementations. The designs are *Clear-on-Retire*, *Epoch*, and *Counter*, and are outlined in Table 2. They provide three different tradeoffs in execution overhead, security, and implementation complexity.

A fourth artifact is the program analysis pass for use in the *Epoch* scheme. The pass places "start-of-epoch" markers in loops and loop iterations in an x86 executable. The pass accepts as input a program in source code or binary. We did not stress it much because it does not include new ideas.

Our hardware designs are implemented in and evaluated with a cycle-level simulator (Gem5). They are evaluated on the SPEC17 benchmark suite, using the Simpoint methodology to select up to 10 representative intervals to estimate the end-to-end application performance.

The hardware designs are also evaluated on a simple MRA attack presented in [19]. To perform the analysis pass, we use the binary analysis tool Radare2 [1].

5. Key Results and Contributions

The key result is that variations of the three proposed designs of *Jamais Vu* (i) add modest execution time overhead, (ii) can effectively mitigate MRAs, and (iii) only need simple hardware implementations.

Specifically, among all the schemes, *Clear-on-Retire* has the lowest execution time overhead. It incurs only a geometric mean overhead of 6.0% over a processor with no MRA protection (Figure 7). It is also the simplest but least secure design (Table 3). *Epoch* with loop iteration epochs has the next lowest average execution overhead, namely 13.5%. This design is also very simple and is more secure. The next design, *Epoch* with whole-loop epochs, has higher average execution time overhead, namely 22.6%. However, it has simple hardware and is one of the two most secure designs (Table 3). Finally, *Counter* has the highest average execution overhead, namely 31.0%. It is one of the two most secure schemes, but the implementation proposed is not as simple as the other schemes. Other results are sensitivity analyses of the different designs.

The contributions of this paper are as follows:

• *Jamais Vu*, the first defense mechanism to thwart MRAs. It performs selective fencing of instructions to prevent replays.

• Several designs and implementations of *Jamais Vu* that provide different tradeoffs between execution overhead, security, and design complexity.

• An evaluation of these designs using simulations. Our preferred design, based on *Epoch* with whole-loop epochs, can effectively mitigate MRAs, has an average execution time overhead of 22.6% in benign executions, and only needs simple hardware counting Bloom filters associated with the ROB.

References

[1] UNIX-like reverse engineering framework and command-line toolset. https://github.com/radareorg/radare2.

[2] Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. On the power of simple branch prediction analysis. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[3] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. Port contention for fun and profit. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2019.

[4] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, May 2015.

[5] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandtner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. SMoTherSpectre: Exploiting speculative execution through port contention. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[6] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *Proc. of the ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2017.

[7] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Jump over ASLR: Attacking branch predictors to bypass ASLR. In *Proc. of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[8] Dmitry Evtyushkin, Ryan Riley, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Branchscope: A new side-channel attack on directional branch predictor. In *Proc. of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[9] Ben Gras, Cristiano Giuffrida, Michael Kurth, Herbert Bos, and Kaveh Razavi. ABSynthe: Automatic blackbox side-channel synthesis on commodity microarchitectures. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2020.

[10] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In *Proc. of the USENIX Security Symposium (USENIX)*, 2018.

[11] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2015.

[12] Ahmad Moghimi, Jan Wichelmann, Thomas Eisenbarth, and Berk Sunar. Memjam: A false dependency attack against constant-time crypto implementations. *International Journal of Parallel Programming*, 47(4):538–570, 2019.

[13] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. Varys: Protecting SGX enclaves from practical side-channel attacks. In *Proc. of the USENIX Annual Technical Conference (ATC)*, 2018.

[14] Meni Orenbach, Andrew Baumann, and Mark Silberstein. Autarky: closing controlled channels with self-paging enclaves. In *Proc. of the European Conference on Computer Systems (EuroSys)*, 2020.

[15] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. In *Proc. of the Cryptographers' Track at the RSA Conference (CT-RSA)*, 2006.

[16] Colin Percival. Cache missing for fun and profit. In *Proc. of the Technical BSD Conference (BSDCan)*, 2005.

[17] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for cross-CPU attacks. In *Proc. of the USENIX Security Symposium (USENIX)*, 2016.

[18] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2017.

[19] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher Fletcher. MicroScope: Enabling Microarchitectural Replay Attacks. In *Proc. of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2019.

[20] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[21] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2015.

[22] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2019.

[23] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proc. of the USENIX Security Symposium (USENIX)*, 2014.

[24] Yuval Yarom, Daniel Genkin, and Nadia Heninger. CacheBleed: A timing attack on OpenSSL constant time RSA. *Journal of Cryptographic Engineering*, 7(2), 2017.