Speculative Interference Attacks: Breaking Invisible Speculation Schemes Extended Abstract

1. Motivation

Speculative execution attacks such as Spectre [29] and follow-on work [8, 11, 21, 28, 30, 34, 41, 51] have opened a new chapter in processor security. In these attacks, adversary-controlled transient instructions—i.e., speculative instructions bound to squash—access and then transmit potentially sensitive program data over *microarchitectural covert channels* (e.g., the cache [53], port contention [8]). For example in Spectre variant 1—`if (i < N) { j = A[i]; B[j]; }`—speculative execution bypasses a bounds check due to a branch misprediction, accesses an out-of-bounds value (`j = A[i]`) and transmits that value through a *cache-based covert channel* (`B[j]`), i.e., by forcing a cache fill to occur in a set that depends on `j`. In this paper, we consider the illegally accessed value `j` to be the *secret*. Here, the attacker controls the value of `i`, thus `j` can be any value in program memory and the covert channel can reveal arbitrary program data.

While a variety of covert channels can be used to leak secret values under mis-speculation, cache-based covert channels [29, 35, 52, 53, 54] make the fewest assumptions on the attacker and have therefore received the most attention. This is for two reasons. First, secret-dependent cache fills leave a persistent footprint in the cache which is observable long after speculation squashes. Second, certain levels of modern cache hierarchies are globally shared by all cores in the system, enabling attackers to observe said persistent state changes from other physical cores [33, 52]. By contrast, many other covert channels (e.g., arithmetic port contention [6, 8]) leave only intermittent side effects that must be monitored before the squash, and/or require that the attacker share hardware resources on the same physical core (e.g., branch predictor channels [4, 14])—both of which can be easily blocked (e.g., disabling SMT).

State-of-the-art transient memory side channel defenses. The above view of the covert channel landscape has led to a surge of architecture-level "*invisible speculation*" proposals to block cache-based covert channels due to mis-speculations (e.g., InvisiSpec [51], SafeSpec [26], Delay-on-Miss [37], Conditional Speculation [31], MuonTrap [5]). Invisible speculation schemes add hardware to prevent mis-speculated loads from making persistent state changes to the memory subsystem. To maintain the performance benefits of caching, only non-speculative loads that are bound to retire are allowed to modify the cache state. To maintain the performance benefits of out-of-order execution, loads are allowed to "invisibly" execute (i.e., bring data directly to the core without filling the cache) and forward their results to dependent instructions.

2. Key Insights

In this paper we introduce *speculative interference attacks*, which show that invisible speculation schemes do not fully block speculation-based attacks that use the cache state. Our attacks are based on two key observations. First, that mis-speculated instructions can influence the timing of older, bound-to-retire operations. Second, if changing the timing of a memory operation changes the *order* of that memory operation *with respect to other memory operations*, the resulting re-ordering can cause persistent cache-state changes. Putting these together, we show (among other attack variants) how secret information accessed in a mis-speculated window influences the order of bound-to-retire loads, leaving secret-dependent state changes in the cache—even if invisible speculation is enabled.

To explain these ideas in more detail, consider a simple but representative invisible speculation scheme *Delay-on-miss* (DoM) [37]. DoM issues a speculative load and (a) on an L1 cache hit, forwards the load result to dependent instructions, or (b) on an L1 cache miss, delays servicing the miss and re-issues the load when it becomes non-speculative. In case (a), DoM does not update any replacement state (e.g., replacement bits) in the L1 cache until the load becomes non-speculative. For simplicity, we explain ideas assuming only branch instructions cast speculative shadows [37], i.e., a load is considered non-speculative/safe iff it is older than the oldest unresolved branch. We discuss attacks on more conservative DoM variants in the main paper.

DoM's (and other invisible speculation schemes') stated security goal is to only focus on blocking cache state changes due to mis-speculations, while leaving other covert channels unblocked. This is reflected in DoM's design. On the one hand, DoM prevents mis-speculated loads from directly changing the cache state. On the other hand, DoM allows mis-speculated loads to forward their results to dependent instructions, which can clearly form covert channels through intermittent state changes. For example, both whether a mis-speculated load hits or misses in the L1 cache, and the mis-speculated load's return value, determine whether and how dependent mis-speculated instructions execute. This is exactly the basis for forming, e.g., arithmetic unit port contention covert channels [6, 8].

This paper demonstrates how instructions that cause intermittent state changes can be leveraged to create persistent state changes in the cache. Consider the example in Figure 1,

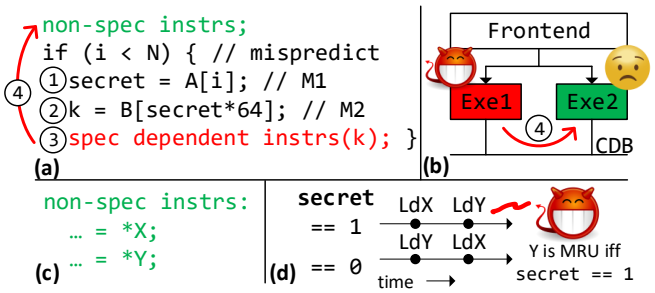

Figure 1: Speculative interference example. (a) Assume the code snippet is run on a processor protected by invisible speculation such as DoM and that `&B[0]` is cached while `&B[64]` is not cached. (b) This results in *speculative dependent instructions* conditionally contending for execution resources with *non-speculative instructions*, depending on the value of `secret`. (c), (d) If the non-speculative instructions are two loads, the contention can influence the order in which the loads are issued. Finally, the attacker can infer the secret based on the cache replacement state after the loads both issue.

modeled after Spectre variant 1. Suppose this code is run on a processor using DoM. In Figure 1 (a), a mis-speculated load M1 forwards secret data `secret` to a second load M2 (①). A normal Spectre attack would monitor the cache state change left by M2 to deduce `secret`. To prevent this leak, DoM would prevent M2 from changing the cache state, specifically by allowing it to access and return data from the L1 if there is an L1 hit and delaying its execution otherwise. While this blocks the cache state change due to M2, M2 is allowed to forward its result when it completes (②), meaning that dependent instructions execute at a time that depends on `secret` (③). This has the potential to create a traditional non-cache based covert channel, e.g., through execution unit port usage, which DoM ignores.

Our key observation is that secret-dependent timing effects caused by the dependent instructions can be monitored *indirectly* through how they interact with the execution of *older non-speculative instructions*. In this example, the instruction(s) before the mis-speculated branch (④). Although the non-speculative instructions come before the speculative dependent instructions in program order, out-of-order execution could have both of them executing concurrently and contending for resources as shown in Figure 1 (b). For example, if they use EXE1 and EXE2, respectively, and contend for the common data bus in the same cycle. We call this *speculative interference*.

Next, we show how speculative interference can be used to bootstrap a change in the cache state. Specifically, suppose the non-speculative instructions are made up of two independent loads to addresses X and Y in different cache lines mapped to the same set, shown in Figure 1 (c). Since these loads are older than the mispredicted branch in program order, they are not protected by DoM. We show how, depending on the timing changes caused by the speculative dependent instructions, the order in which load X is issued with respect to load Y can change. *That is, depending on a secret, the processor issues either loads to X followed by Y or Y followed by X.* To finish the attack (Figure 1 (d)), we show how changing the order of memory operations can be used to create persistent changes in the cache state, the intuition being that state in the cache (e.g., replacement bits) depend on not just what requests are made, but also their order.

This issue is not easy to fix. The crux of the problem is that timing changes can be converted to persistent state changes. These timing changes can arise due to interference through a large number of microarchitectural structures, through different instructions, etc. Further, while our example reorders two loads that originate from the same thread, there are many other memory address streams through which to interleave operations, e.g., interleaving instruction and data cache accesses, accesses made across threads and security domains, etc.—which further widens the attack surface.

3. Main Artifacts

To evaluate our attacks, we implement three proof-of-concept (PoC) attack variants—creating speculative interference through non-pipelined functional units, MSHR usage, and instruction fetch unit backpressure—and verify each variant on an Intel Core i7-7700 Kaby Lake CPU with 4 physical cores. (While commercial processors do not currently implement invisible speculation, we write each PoC to emulate the behavior of such schemes.) Of independent interest, we develop a novel cache attack on the Intel QLRU_H11_M1_R0_U0 replacement policy that detects load-load reorderings. Finally, all of our PoC variants work when the receiver (attacker) runs on a different physical core.

4. Key Results and Contributions

This paper introduces and provides a framework to reason about *speculative interference attacks*, whereby subtle secret-dependent microarchitectural interference influences the behavior of older non-speculative instructions. We show how this can be used to create cache-based covert channels, even in the presence of invisible speculation schemes. As stated above, we implement three working proof-of-concept exploits creating different types of speculative interference that all lead to cache-based side channels. Finally, we conclude with a discussion on security definitions for soundly defeating the attacks, and preliminary defensive ideas based on those definitions.

5. Why ASPLOS

This paper sits at the intersection between Architecture and Security.

6. Citation for Most Influential Paper Award

This paper introduced speculative interference attacks, showed how those attacks can undermine security for multiple proposed invisible speculation schemes and set a research agenda for next-generation invisible speculation schemes.

References

[1] 8th and 9th generation intel® core™ processor families datasheet, volume 1 of 2. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/8th-gen-core-family-datasheet-vol-1.pdf.

[2] Kaby lake - microarchitectures - intel - wikichip. https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake.

[3] Andreas Abel and Jan Reineke. nanobench: A low-overhead tool for running microbenchmarks on x86 systems. *arXiv preprint arXiv:1911.03282*, 2019.

[4] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In *Cryptographers' Track at the RSA Conference*. Springer, 2007.

[5] Sam Ainsworth and Timothy M. Jones. Muontrap: Preventing cross-domain spectre-like attacks by capturing speculative state. In *Proc. of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2020.

[6] 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)*. IEEE, 2019.

[7] Kristin Barber, Anys Bacha, Li Zhou, Yinqian Zhang, and Radu Teodorescu. SpecShield: Shielding Speculative Data from Microarchitectural Covert Channels. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2019.

[8] 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.

[9] Samira Briongos, Pedro Malagón, José M Moya, and Thomas Eisenbarth. Reload+refresh: Abusing cache replacement policies to perform stealthy cache attacks. In *Proc. of the USENIX Security Symposium (USENIX)*, 2020.

[10] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. Fallout: Leaking data on meltdown-resistant cpus. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[11] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai. SgxPectre: Stealing intel secrets from sgx enclaves via speculative execution. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[12] 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.

[13] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. Understanding and mitigating covert channels through branch predictors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 13(1), 2016.

[14] 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.

[15] Agner Fog et al. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for intel, amd and via cpus. *Copenhagen University College of Engineering*, 93:110, 2011.

[16] Jacob Fustos, Michael Bechtel, and Heechul Yun. SpectreRewind: Leaking secrets to past instructions. *arXiv preprint arXiv:2003.12208*, 2020.

[17] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 1982.

[18] 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.

[19] Johann Großschädl, Elisabeth Oswald, Dan Page, and Michael Tunstall. Side-channel analysis of cryptographic software via early-terminating multiplications. In *Proc. of the International Conference on Information Security and Cryptology (ICISC)*, 2009.

[20] John L. Hennessy and David A. Patterson. *Computer Architecture, Sixth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 6th edition, 2017.

[21] Jann Horn. Speculative execution, variant 4: speculative store bypass. https://bugs.chromium.org/p/project-zero/issues/detail?id=1528, 2018.

[22] Intel. Refined Speculative Execution Terminology. https://software.intel.com/security-software-guidance/insights/refined-speculative-execution-terminology, 2020.

[23] Aamer Jaleel, Kevin B Theobald, Simon C Steely Jr, and Joel Emer. High performance cache replacement using re-reference interval prediction (rrip). *ACM SIGARCH Computer Architecture News*, 38(3):60–71, 2010.

[24] Mike Johnson. *Superscalar Microprocessor Design*. Prentice Hall Englewood Cliffs, New Jersey, 1991.

[25] Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. A high-resolution side-channel attack on the last level cache. In *Proc. of the Design Automation Conference (DAC)*, 2016.

[26] Khaled N. Khasawneh, Esmaeil Mohammadian Koruyeh, Chengyu Song, Dmitry Evtyushkin, Dmitry Ponomarev, and Nael B. Abu-Ghazaleh. Safespec: Banishing the spectre of a meltdown with leakage-free speculation. In *Proc. of the Design Automation Conference (DAC)*, 2019.

[27] Vladimir Kiriansky, Ilia A. Lebedev, Saman P. Amarasinghe, Srinivas Devadas, and Joel Emer. Dawg: A defense against cache timing attacks in speculative execution processors. In *Proc. of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[28] Vladimir Kiriansky and Carl Waldspurger. Speculative buffer overflows: Attacks and defenses. *arXiv preprint arXiv:1807.03757*, 2018.

[29] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2019.

[30] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. Spectre returns! speculation attacks using the return stack buffer. In *Proc. of the USENIX Workshop on Offensive Technologies (WOOT)*, 2018.

[31] Peinan Li, Lutan Zhao, Rui Hou, Lixin Zhang, and Dan Meng. Conditional speculation: An effective approach to safeguard out-of-order execution against spectre attacks. In *Proc. of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[32] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *Proc. of the USENIX Security Symposium (USENIX)*, 2018.

[33] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2015.

[34] Giorgi Maisuradze and Christian Rossow. Ret2spec: Speculative execution using return stack buffers. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2018.

[35] 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.

[36] Gururaj Saileshwar and Moinuddin K. Qureshi. Cleanupspec: An "undo" approach to safe speculation. In *Proc. of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[37] Christos Sakalis, Stefanos Kaxiras, Alberto Ros, Alexandra Jimborean, and Magnus Själander. Efficient Invisible Speculative Execution Through Selective Delay and Value Prediction. In *Proc. of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2019.

[38] Jay Schulist, Daniel Borkmann, and Alexei Starovoitov. Linux Socket Filtering aka Berkeley Packet Filter (BPF). https://www.kernel.org/doc/Documentation/networking/filter.txt, 2018.

[39] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. ZombieLoad: Cross-privilege-boundary data sampling. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[40] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. Fantastic timers and where to find them: high-resolution microarchitectural attacks in javascript. In *Proc. of the International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2017.

[41] Michael Schwarz, Martin Schwarzl, Moritz Lipp, and Daniel Gruss. Netspectre: Read arbitrary memory over network. In *Proc. of the European Symposium on Research in Computer Security (ESORICS)*, 2019.

[42] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: Using sgx to conceal cache attacks. In *Proc. of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2017.

[43] Robert M Tomasulo. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of Research and Development*, 11(1):25–33, 1967.

[44] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proc. of the USENIX Security Symposium (USENIX)*, 2018.

[45] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue in-flight data load. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2019.

[46] Pepe Vila, Pierre Ganty, Marco Guarnieri, and Boris Köpf. CacheQuery: Learning Replacement Policies from Hardware Caches. In *Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2020.

[47] Jack Wampler, Ian Martiny, and Eric Wustrow. Exspectre: Hiding malware in speculative execution. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2019.

[48] Ofir Weisse, Ian Neal, Kevin Loughlin, Thomas Wenisch, and Baris Kasikci. NDA: Preventing Speculative Execution Attacks at Their Source. In *Proc. of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[49] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F. Wenisch, and Yuval Yarom. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution. *Technical report*, 2018.

[50] Wenjie Xiong and Jakub Szefer. Leaking Information Through Cache LRU States. In *Proc. of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[51] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher W. Fletcher, and Josep Torrellas. InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy. In *Proc. of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[52] 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.

[53] 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.

[54] Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: a timing attack on openssl constant-time rsa. *Journal of Cryptographic Engineering*, 7(2):99–112, 2017.

[55] Jiyong Yu, Namrata Mantri, Josep Torrellas, Adam Morrison, and Christopher W. Fletcher. Speculative Data-Oblivious Execution (SDO): Mobilizing Safe Prediction For Safe and Efficient Speculative Execution. In *Proc. of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2020.

[56] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W. Fletcher. Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data. In *Proc. of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.