Building General Abstractions for Comprehensive and Efficient Micro-architectural Side-Channel Protections

Jiyong Yu
Department of Computer Science
University of Illinois at Urbana-Champaign

December 6, 2022

1 Introduction

This thesis proposal focuses on designing micro-architectural side-channel-resistant processors with comprehensive security guarantees and high performance, and concurrently constructing novel attacks to defeat existing side-channel mitigations.

In the last 90s, when the rapid advancement in computer architecture design constantly delivers performance improvement, researchers started to realize that the data-dependent nature of those designs could leak the processed sensitive data [7, 4]. Since then, numerous processor structures have been exploited to leak different sensitive information, which turns micro-architectural side channels a substantial threat to today's computing industry [6, 9].

Given that side-channel resistance now becoming a priority for both software and hardware designers, both parties propose different approaches for addressing the side-channel threat. First, since the micro-architecture side-channel stems from the processor design, fixes from the hardware side provide direct and promising answers to this issue. Usually, the fixes will separately target individual vulnerabilities, with minimal impact on the overall performance. Since hardware is a synergy of countless structures, fixing side channels individually without a holistic view is always incomplete. Besides, given the amount and complexity of performance optimizations in modern processors, hardware designers usually fail to identify all the side channel vulnerabilities. *The missing holistic analysis and protection for modern processors is the primary challenge in side-channel research.*

Software mitigations, despite being more timely than hardware fixes, are also insufficient in addressing side-channel attacks. Since hardware details are abstracted from software, those defenses can only address specific side-channel attacks instead of providing comprehensive side-channel protection. Additionally, software mitigations cannot react to future side channels resulting from hardware changes. *This lack of security specification between hardware and software precludes any valid and accurate side-channel assessment for software applications.*

To overcome the challenges in existing micro-architectural side-channel mitigations, this proposal is devoted to **mitigating micro-architectural side-channel attacks comprehensively and efficiently, using hardware-based side-channel abstractions**, with the following two general directions:

Thrust 1: Building general hardware-based side-channel abstractions (existing works) For this direction, we present three existing works (section 2) to demonstrate that by constructing general abstractions covering broad micro-architectural leakages, we can develop simple yet holistic side-channel defenses, and at the same time, avoid unnecessary performance costs of the defense. "Data Oblivious ISA" [18] or OISA in short is the first proposal to add security specification at the instruction set architecture (ISA) level. The security specification precisely defines whether user-defined private data can be leaked through a given instruction's operand. With this new ISA, writing high-performance, side-channel-free applications becomes reality. In response to the more recent speculative execution attacks [6, 9], we introduced a taxonomy of all covert channels in modern processors that could lead to speculative leakage. The final product, named "Speculative Taint Tracking" [20] or STT, prevents any speculatively executed instruction from leaking speculatively accessed data through any data-dependent hardware activities. Due to the classification of covert channels and precise security policy, STT is the first speculative attack mitigation with provable security guarantees. The follow-up work "Speculative Data-Oblivious Execution" [19], or

SDO, is built based on STT to inherit STT's comprehensive security guarantees. Meanwhile, it improves several conservation protection approaches adopted by STT for reducing STT's performance overhead while keeping the security guarantees intact.

Thrust 2: Attacking existing software-based side-channel mitigations (ongoing works) For this direction, we aim to illustrate that pure software-based side-channel mitigations without any guarantees from the hardware are insufficient, with two of my ongoing works (section 3). To start, we focus on a series of side-channel attacks with the purpose of leaking secret-dependent control flow and their corresponding software-based mitigations. These software mitigations apply different code transformations to eliminate the code patterns that are exploited in the side channel, without completely eliminating the secret-dependent control flow. In this ongoing work, we develop a new side-channel attack capable of leaking the full control flow from the victim program's execution, in the form of dynamic PC sequences. This attack demonstrates that software-only approaches are insufficient to deliver provable security guarantees for today's complex and constantly-evolving processors. Another important class of software mitigations focuses on preventing side-channel attackers from observing secret-dependent hardware resource usage. Since most side-channel attackers rely on high-precision hardware performance counters, such as timers to measure how hardware resource states react to the secret data and further deduce the secrets, many defenses are proposed to disable or fuzz the performance counters. However, we claim that those software approaches still cannot achieve the ideal security guarantee since no evidence is available showing that performance counters are the only reliable way to measure hardware resource usage. In fact, in our ongoing project, we aim to show a fundamental hardware primitive, load-linked and store-conditional instructions, is a reliable channel for detecting cache evictions. Using these two instructions, the attacker can accurately observe the memory access pattern of a process running on the same processor.

2 Existing Works

2.1 Data-Oblivious ISA

A major obstacle that keeps software developers from building side-channel-free and high-performance applications, is the lack of knowledge about hardware implementation. As an example, *data-oblivious programming* has been proposed to block any side-channel leakage by not incurring observable secret-dependent hardware resource usage. However, running data-oblivious programs on modern processors is neither secure nor efficient. The software-hardware interface, ISA, is a functional specification rather than an implementation specification. As a result, hardware designers have the freedom to add arbitrary features to hardware, which may open up new side channels. Besides, writing programs to avoid data-dependent behaviors is inherently inefficient.

To address this obstacle, in *Data Oblivious ISA* [18] or OISA, we propose a series of ISA design principles for data oblivious programming. The ISA design principles we proposed expand existing ISA with security specifications, allowing software and hardware to communicate side-channel-related information, without disclosing the hardware implementation details. The ISA design principles are two-fold: First, the software can denote data as *public* and *private*, with only private data protected by the hardware. Second, each instruction operand is either *safe* or *unsafe*. An unsafe operand induces attacker-observable data-dependent execution which leaks information about the consumed data.

With the definition of data and operand types, OISA imposes strict security policies that hardware must enforce to avoid leaking private data. Public data can be consumed by both safe and unsafe operands. Private data is only allowed to be consumed by safe operands, which hardware implementation promises to hide attacker-observable data-dependent behaviors. Private data consumed by unsafe operands are strictly prohibited and will throw a hardware exception before the instruction execution starts. In addition, to prevent the leakage of any information about private data, OISA requires hardware to employ the information-flow tracking mechanism to track all data dependent on the user-labeled private data at the hardware level.

Aside from the strong and comprehensive security guarantees, with the safe operand semantics, OISA can incorporate existing hardware-based side-channel mitigation strategies depending on each microarchitecture and its performance requirements. For example, to protect secret address operands, one can break memory instruction into simpler data-oblivious instructions, or opt for cryptographic techniques [14] or hardware paritioning [15] which yield asymptotically better performance with more hardware cost.

Based on the design methodology described above, in this work, we design a concrete OISA based on the existing RISC-V instruction set, with a hardware prototype and simulation framework on top of RISC-V BOOM processor [2]. We demonstrate that OISA is capable of enhancing existing instruction sets and commodity processors with provable and holistic side-channel security guarantees. At the same time, OISA provides great compatibility in incorporating existing defense techniques on commodity processors to further reduce the performance cost of side-channel defenses.

2.2 Speculative Taint Tracking

Despite being a holistic side-channel defense, OISA is limited by its requirement of manual data labeling (which is often impractical) and intrusive hardware modifications in various hardware structures. With the surge of speculative execution attacks [6, 9] in recent years, we present *Speculative Taint Tracking* [20], or STT, as a comprehensive mitigation for speculative execution attacks.

Similar to classic side-channel attacks, speculative execution attacks leverage data-dependent hardware resource usage (e.g., cache) to infiltrate program secrets. What separates speculative attacks from other side channels is its capability of utilizing hardware speculation to acquire secret data that are not reachable from valid program flows, when data is accessed during a mis-speculation. Therefore, a natural (and conservative) approach to defeat speculative attacks is to delay the execution of instructions that reads those secrets, until the instructions become non-speculative.

STT's premise is that it is safe to execute the selectively forward outputs of speculative instructions which read secrets, as long as the forwarded outputs do not reach micro-architectural covert channels. Since user-labeling is impractical and cannot cover all the secret information that is accessible with the presence of speculative execution, STT deems any data returned by speculative memory accesses secrets. STT also leverages hardware-based taint tracking logic for tracking the information flow of secrets, akin to OISA.

To achieve its security goal, STT presents the first taxonomy of all micro-architectural covert channels that a speculative attacker could harness, and provides sound protection for secrets in the context of speculative attacks by preventing speculatively accessed data from reaching the covert channels. Specifically, STT classifies micro-architectural covert channels into *explicit channels* and *implicit channels*. Explicit channels refer to the interactions between certain instruction types and various data-dependent hardware resources, such as cache, TLB. STT delays the execution of instructions that deliver secret data into explicit channels. Implicit channels refer to the influence of secret data on subsequent speculative execution, by making the state of hardware predictors depend on the secrets. Similarly, for eradicating implicit channels, STT delays the impact of secret data on hardware predictors. On the other hand, it is essential to relax the protection for secret data (i.e. canceling the delays) as soon as doing so is safe. We identify that the earliest time is when the instruction(s) producing the protected data become non-speculative, and design a novel hardware technique for disabling protection at that moment.

Overall, STT delivers comprehensive and provable security protection against speculative execution attacks in general, with a modest performance overhead (8.5% to 22.4% depending on the threat model and the tested benchmark suite).

2.3 Speculative Data-Oblivious Execution

STT provides strong and comprehensive protection against speculative execution attacks, but its performance still remains to be improved. After analysis of STT's performance overhead, we found that 97% of the overhead is due to delaying imposed by explicit channels, i.e., delaying of operations such as memory accesses that cause secret-dependent hardware resource usages. Therefore, a major design objective is to execute those unsafe operations in a safe manner rather than simply delaying them.

Speculative Data-Oblivious Execution [19], or SDO, extends STT for this design objective without sacrificing the security claim of STT. When a speculative secret is about to be processed by unsafe execution units which have data-dependent behavior, SDO changes the behavior of those execution units from data-dependent to data-oblivious.

To illustrate how such transformation is conducted, consider float-point operations on modern processors. Floating-point operations typically have operand-dependent behavior: for normal operand values, the operations are executed by a fast hardware floating-point unit, resulting in a fixed, short latency. When the operand is exceptionally small (which is usually referred to as

subnormal), the instruction's execution requires microcode assist and takes significantly longer [1]. Consequently, the execution of floating-point operations creates two execution equivalence classes: slow and fast, and by inferring which equivalence class an operation belongs to by, e.g., monitoring program runtime, information about the operand values is revealed.

To achieve data obliviousness, one can perform all execution equivalence classes, such as executing the operation with both FPU and microcode assist in the example above. Although this idea achieves security from data obliviousness, as the hardware usage is not independent of the secret input, it is highly inefficient since we must wait for the result of the slowest (e.g., subnormal) mode completes, to hide which mode was actually needed.

The key insight of SDO in addressing this issue is to *predict* that one (or several) equivalence class is valid and only execute the operation with the predicted class. When the prediction is accurate enough, the hardware will execute the unsafe operation efficiently and generate valid outputs in most cases. However, an obvious pitfall with this approach is the prediction and the produced output might reveal private information. SDO avoids this by inheriting the security mechanisms from STT, such as hardware taint tracking, and protection against implicit channels. STT's implicit channel protection ensures hardware prediction and resolution never depend on speculative secrets, thus also covering the new prediction introduced by SDO. STT's taint tracking property propagates the taint from the secret input to the output of the predicted execution equivalence class, therefore even the invalid output (due to predicting the wrong equivalence class) cannot be leaked via dependent instructions.

The above approach summarizes SDO, and can be generalized across different instruction types beyond just float-point operations. Importantly, we propose an SDO mechanism for memory loads, since most performance overhead in blocking speculative execution attacks is due to loads [16, 20]. The SDO mechanism for loads includes a new speculative data-oblivious load operation including multiple equivalence classes and their implementations, and a novel *location predictor* for predicting the equivalence class for a given load instruction. Overall, SDO improves STT's performance by an average of 36% to 55%, depending on the microarchitecture and attack model without weakening STT's security guarantees.

3 Ongoing Works

3.1 NightVision

In reality, most micro-architectural side channels have no hardware fixes. Instead, people rely on software-based mitigations that replace code patterns vulnerable to existing side channels with new code structures. However, the software strategy is only based on the knowledge of the existing side-channel attack, which is only related to certain behaviors of a subset of hardware structures. Consequently, software mitigations for side channels usually result in a cat-and-mouse game, in which new side-channel vulnerabilities constantly emerge and new software transformations are proposed on top of existing ones for mitigating the new vulnerability.

As an example, consider an important class of side-channel attacks that leaks the secret-dependent program control flow, such as the branch direction. There has been significant work to mitigate these attacks individually, such as through branch balancing [12], control-flow randomization [3], and instruction aligning [13]. The end result is that every proposed side-channel attack could be properly mitigated at the software level without actually eliminating secret-dependent control flow (which is preferred for performance reasons).

We aim to show that pure software mitigations without assistance from the hardware are inherently incapable of providing robust security guarantees. To demonstrate this, we visit Branch Target Buffer (BTB), an important hardware structure used by modern high-performance processors for accurate branch prediction. This structure has been studied by prior side-channel attacks for leaking control-flow information about branches. Through careful reverse-engineering of BTB in existing Intel processors, we identify previously overlooked, yet important BTB features that reveal the control-flow information about arbitrary instruction types aside from just branches.

With this finding, we develop an attack framework named NightVision, which is capable of recovering the dynamic PC of every executed instruction by the victim. The attack basically works as a Prime+Probe attack on BTB, with complimentary mechanisms to shorten the victim's execution in between the Prime and the Probe for achieving fine-grained measurement of the victim's dynamic PCs. Since a full dynamic PC trace essentially represents the entire program control flow, any mitigation that does not eliminate secret-dependent control flow becomes susceptible to NightVision by default. NightVision, therefore, shows how important it is to provide compre-

hensive hardware-based side-channel defenses rather than relying on ad hoc software changes in reaction to each individual attack.

NightVision's capability of recovering the program's PC pattern can be utilized in different attack scenarios. First, we use NightVision to attack existing cryptographic implementations which contain secret-dependent control flow. Second, in response to the recent effort in confidential cloud computing that hides the program code for hindering attacks [17], NightVision can deduce the exact PC of every victim dynamic instruction and employ binary fingerprinting techniques to reverse-engineer private programs with the extracted PC trace.

3.2 LoadPrime+StoreProbe

Mitigating micro-architectural side-channel attacks is extremely challenging given the numerous hardware units producing data-dependent state changes which eventually lead to side channels. However, from the attacker's perspective, the hardware state (such as cache occupation) is inaccessible, thus a successful side-channel attack must rely on certain mechanisms to transform secret-dependent hardware state changes into software state changes. The most widely-used mechanism in existing side-channel research is hardware performance counters, especially hardware timers, which are commonly used for measuring the duration of a given piece of code. Other mechanisms such as power [8] and temperature [5] measurement also reveal information about the hardware activities but are far more coarse-grained and noisy than timers.

Correspondingly, people consider restricting access to high-precision timers as a simple yet comprehensive mitigation against side channels [11], since the lack of other reliable methods to convert hardware state changes to software state changes. However, there could still be unexplored primitives, or new primitives that appear in the future that can achieve this hardware-to-software state conversion. In our attacker called LoadPrime+StoreProbe, we identify such a primitive in existing Apple M-series processors, and thus claim that mitigating side-channel attacks with comprehensive, hardware-based mechanisms is the more reliable approach and provides real security guarantees.

LoadPrime+StoreProbe exploits Apple M-series processors' implementation of load-linked/store-conditional instructions to present a timer-less cache side-channel attack, which is equally powerful as the traditional timer-based cache side-channel attack [10]. Load-linked/store-conditional (LL/SC) instructions are commonly used for implementing atomic operations such as read-modify-write, where LL and SC are applied to the same address. For a pair of LL/SC accessing the same address, the SC fails when there exist any updates (from the current thread or other threads) to the address. Hardware thus uses metadata to label the address that has been LL-ed and is waiting for the corresponding SC. This metadata is normally piggybacked to the cache coherence state in most commodity processors such as ARM.

We made a key observation that the cache coherence state can be invalidated when data is evicted from the cache, in which SC might conservatively fail. We tested Apple M1 and M2 processors and verified in those processors, SC indeed returns fails when the previous LL-ed data is evicted from L1 data cache. Since cache eviction can be caused by other processes sharing the same processor, this enables an attacker to observe the cache access pattern of other processes from the return state of SC.

With this observation, we aim to develop the first cross-core cache side-channel attack that relies on LL/SC instead of high-precision timers. Our attack starts by constructing an *eviction set* for a target victim address. The attacker accesses the first element in the eviction set using LL, and traverses the rest of the set such that the element is in the LRU state in the shared cache. In this way, when the victim performs an access to the victim address, the load-linked data will be evicted, and a subsequent SC will return fail. Using LoadPrime+StoreProbe, we will construct Proof-of-Concept attacks such as constructing a covert channel between two processes, and attacking cryptographic implementations such as T-Table AES. If time permits, we will also attempt more realistic settings such as cross-browser-sandboxes or cross-VMs.

References

[1] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *2015 IEEE Symposium on Security and Privacy*, pages 623–639. IEEE, 2015.

[2] Christopher Celio, Pi-Feng Chiu, Borivoje Nikolic, David A Patterson, and Krste Asanovic. Boomv2: an open-source out-of-order risc-v core. In *First Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2017.

[3] Shohreh Hosseinzadeh, Hans Liljestrand, Ville Leppänen, and Andrew Paverd. Mitigating branch-shadowing attacks on intel sgx using control flow randomization. In *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, pages 42–47, 2018.

[4] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In *European Symposium on Research in Computer Security*, pages 97–110. Springer, 1998.

[5] Taehun Kim and Youngjoo Shin. Thermalbleed: A practical thermal side-channel attack. *IEEE Access*, 10:25718–25731, 2022.

[6] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. *Communications of the ACM*, 63(7):93–101, 2020.

[7] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[8] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. Platypus: Software-based power side-channel attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371. IEEE, 2021.

[9] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, et al. Meltdown: Reading kernel memory from user space. *Communications of the ACM*, 63(6):46–56, 2020.

[10] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pages 605–622. IEEE, 2015.

[11] Robert Martin, John Demme, and Simha Sethumadhavan. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 118–129. IEEE, 2012.

[12] Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. Copycat: Controlled instruction-level attacks on enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 469–486, 2020.

[13] Ivan Puddu, Moritz Schneider, Miro Haller, and Srdjan Čapkun. Frontal attack: Leaking control-flow in sgx via the cpu frontend. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 663–680, 2021.

[14] Sajin Sasy, Sergey Gorbunov, and Christopher W Fletcher. Zerotrace: Oblivious memory primitives from intel sgx. *Cryptology ePrint Archive*, 2017.

[15] Mohit Tiwari, Xun Li, Hassan MG Wassel, Frederic T Chong, and Timothy Sherwood. Execution leases: A hardware-supported mechanism for enforcing strong non-interference. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 493–504, 2009.

[16] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher Fletcher, and Josep Torrellas. Invisispec: Making speculative execution invisible in the cache hierarchy. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 428–441. IEEE, 2018.

[17] Jiyong Yu, Xinyang Ge, Trent Jaeger, Christopher W Fletcher, and Weidong Cui. Pagoda: Towards binary code privacy protection with sgx-based execute-only memory. In *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 133–144. IEEE, 2022.

[18] Jiyong Yu, Lucas Hsiung, Mohamad El'Hajj, and Christopher W Fletcher. Data oblivious isa extensions for side channel-resistant and high performance computing. In *The Network and Distributed System Security Symposium (NDSS)*, 2019.

[19] Jiyong Yu, Namrata Mantri, Josep Torrellas, Adam Morrison, and Christopher W Fletcher. Speculative data-oblivious execution: Mobilizing safe prediction for safe and efficient speculative execution. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 707–720. IEEE, 2020.

[20] 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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 954–968, 2019.