Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data

&

Revizor: Testing Black-Box CPUs against Speculation Contracts

Yash Shah (ys562)

November 2024, Word count: 1638

1. Contributions of the STT paper

A transient execution attack consists of two parts. Firstly, a program exploits speculative execution to read arbitrary program data (the *secret*) into the CPU's architectural state via an *access instruction*. Secondly, this data is exfiltrated through a microarchitectural covert channel formed by *transmit instructions*.

Delaying execution of any instruction capable of accessing a secret until it is non-speculative is overly conservative, resulting in high performance degradation. STT [1] achieves safety and performance by tracking instruction data-dependencies and focussing only on blocking execution of instructions that can form covert channels. Efficiency is maintained by disabling protection as soon as the access instruction becomes non-speculative.

Tracking is achieved by leveraging existing register-renaming logic to taint the output register of unsafe access instructions. The taint is then propagated—an instruction's output register is tainted if any of its input registers are tainted, with transmit instructions being stalled in reservations stations (RSs) based on taint. Output registers are untainted when an access instruction becomes safe (crosses visiblity point), with untaint propogated and transmit instructions resumed or squashed.

2. Contributions of the Revizor paper

Revizor [2] is a Model-based Relational Testing (MRT) framework for empirically identifying contract violations in black-box CPUs. A (speculation) contract declares ISA operations observable by an attacker through a side channel (*observation clause*) and operations that can speculatively change control/data flow (*execution clause*). The contract is violated if a CPU allows an attacker to observe more than what is possible through the contract.

Revizor works as follows:

1. Instruction sequences (test cases) and random inputs are generated in a way that increases probability of speculation and information leakage.
2. A contract trace is collected by executing the contract on an ISA-level emulator (with checkpointing to explore both correctly and incorrectly predicted speculation) and test case. The trace contains data based on observation clauses encountered with speculative control flow based on the execution clause.
3. A hardware trace is collected by executing the test case on the CPU and measuring the observable microarchitectural changes.
4. The relational testing step is performed—the inputs to a test case are partitioned into equivalence classes according to the contract traces produced under those inputs (with singleton classes discarded). If for any class, there are at least two inputs with differing hardware traces, we have found a contract violation, and a counter example (program + two sets of inputs).
5. If no violation is found, diversity of random generation is increased and process restarted (or we give up).

3. Reliability

STT's design is entirely deterministic, enabling leveraging of formal analysis to provide guarantees regarding the CPU's design, most importantly that arbitrary program memory cannot be leaked (but only retired register file state) under malicious speculation. This increases confidence in designs leveraging STT.

Comparatively, Revizor provides no formal guarantees on its detection capabilities and suffers from both false positives and false negatives from noise and non-determinism. Since there is no way to deterministically set microarchitectural state e.g. branch predictors are not architecturally accessible, divergent traces (false positives) occur even in the absence of leakage.

Revizor introduces *priming* of predictors to combat this where traces for a large number of pseudorandom inputs to same test case are collected in a sequence. Filtering of false positives is then achieved by testing inputs causing divergent hardware traces in the other input's context. Additionally, external software noise is minimized to avoid false positives by executing the executor as a single-core kernel module with hyperthreading, prefetching, and interrupts disabled.

False negatives, e.g. due to speculation not being triggered, are mitigated by controlling the randomness in test and input generation. Since there is no true

microarchitectural coverage metric available, PRNG entropy is gradually increased, test cases lengthened and generated to cover more pairwise instruction patterns (like store-after-store) to reduce false negative probability. However, this costs input effectiveness as contract traces also diverge with increased probability.

Hardware traces are captured multiple times and merged to increase reproducibility in face of measurement noise. However, outliers observed only once during repeated testing are hard to reproduce and verify manually. Thus, vulnerabilities may be missed.

Although lack of direct microarchitectural visibility causes non-determinism, this is a major advantage as Revizor can be applied to commercial, black-box CPUs, whereas implementing STT requires microarchitectural knowledge and alterations.

4. Configurability, granularity, and security-performance trade-offs

STT's configuration revolves around the threat model chosen to protect against, and is key in dictating an instruction's *visibility point* (VP), after which an instruction is considered safe. In the Spectre model, VP is reached once all older control-flow instructions have resolved, but in Futuristic, it is when the instruction cannot be squashed. Futuristic is stronger, blocking additional attacks such as Meltdown, but increases the performance penalty over Spectre (8.5–14.5%).

Additionally, STT allows for customizable taint/untaint propogation rules, and selective enabling of protection mechanisms like prediction- and resolution-based channel protection. By choosing a threat model and appropriate STT customizations, designers can precisely control their security-performance trade-off. However, these choices are designed into the hardware, and cannot be configured at runtime.

Revizor offers a similar, but coarser trade-off with its information leak detection. The speculation contract specifies the allowed leakage in the system. Combinations of observation and execution clauses correspond to modelling particular vulnerabilities e.g. CT-COND models a CPU vulnerable to Spectre V1. An architect could e.g. use the contract to allow for leakage already known to exist in the system, and have Revizor find subtler vulnerabilities, but requires more time as more observations required reduce input effectiveness.

Test generation can also be extensively configured for Revizor. There are adjustible parameters for test length (number of basic blocks, block size), number of registers used, and memory sandbox size. Instruction patterns, whose configuration controls interaction between speculation types, can be used for rough threat targeting. Increasing the richness of these tests is sometimes necessary to find Spectre V4-like violations requiring a longer speculation window, but the decreased input effectiveness greatly increases detection time, with V4 detection taking 62 inputs on average, compared to just 4 for V2.

5. Scope, coverage, and assumptions

A key aim of STT is the elimination of the universal read gadget, and thus focuses only on protecting speculatively accessed data (non-transient access instructions cannot form a universal read gadget). In contrast, Revizor has broader scope since it can detect leakage of non-speculatively loaded data not protected by STT. Non-transient access leakage is out of scope for STT as it delegates its handling to compilers and complementary techniques [3].

Furthermore, while STT provides comprehensive, deterministic coverage, these are only against known attack vectors. Revizor has the potential to automatically discover novel attacks due to its randomized tests.

STT works with an idealized hardware model, explicitly making assumptions regarding hardware behaviour. Comparatively, Revizor acknowledges hardware complexity, treating it as a black-box, avoiding additional assumptions that may be invalidated. For example, STT assumes that stores do not modify cache state until retirement. However, Revizor shows that this assumption is invalid. When CT-COND is used to capture this assumption in the contract trace, a violation is found with a trace left by a speculative store.

6. Scalability

Both approaches suffer from some scalability issues. STT requires significant effort regarding blocking leakage through implicit branches. Although the mitigation mechanisms are similar to those used against prediction- and resolution-based channels (like updating predictors with only untainted data and delaying the resolution of branches until its predicate is untainted respectively), processor optimizations present in real-world CPUs add significant complexity if this technique were to be applied commercially.

Although Revizor presents itself as an automated testing platform, the counterexample produced must be manually inspected. Although the post-processor helps minimize the test case, rare counterexamples are often long and difficult to interpret and reproduce. False positives make this particularly problematic.

7. Future development and critique (STT)

STT still adds considerable overhead (14.5% in PARSEC workloads) and a key reason for this is that transmit instructions are delayed until either its corresponding access instruction becomes non-speculative or execution squashes due to mis-speculation. However, this is too conservative as execution of transmit instructions is safe as long as they do not require operand-dependent hardware resource usage. We can speculatively execute transmitters early if we can execute it in a data oblivious manner [4].

A floating-point instruction can form a covert channel as the operand values determine if it executes on fast or slow hardware. Executing both versions is safe as that is

data-oblivious, but we must wait for the slowest execution to finish. To mitigate this, an *equivalence class predictor* can speculatively execute the "correct" form, without leaking information. STT can then be used to prevent the predictor itelf from leaking information.

STT doesn't benefit from information from the compiler. Instructions can become speculation-invariant (where whether the instruction will execute and its operands are not a function of speculative state) before turning non-speculative. InvarSpec [5] shows that through program analysis identifying for each instruction, those that do not prevent it from becoming speculation invariant, permitted speculative instructions can be issued without protection, improving performance. However, this approach relies on hardware-software co-design and recompilation of existing programs. Nonetheless, it would be interesting to see a combination with STT.

8. Future development and critique (Revizor)

Revizor suffers from scalability issues e.g. as more side channels are covered. This could be addressed through borrowing ideas from constrained random testing (like Genesys-Pro [6]) where both the tests and their inputs are solutions to a Constraint Satisfaction Problem (CSP). This has further potential by architects being able to integrate their design and testing knowledge as additional constraints to help guide the random search further. This would allow Revizor to expand its range of testing targets and reduce false negatives, with good performance.

Additionally, Revizor does not cover CPU exceptions, which are the source of vulnerabilities such as Meltdown [7] and MDS [8]. A key challenge to extending Revizor to cover these is documenting the leakage behaviour of exceptions as a flexible formal model due to peculiar transient execution characteristics during exceptions. Existing work [9] provides a good base for building this formal model, but exception modelling is a key direction for future development.

Bibliography

[1] J. Yu, M. Yan, A. Khyzha, A. Morrison, J. Torrellas, and C. 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*, in MICRO '52. Columbus, OH, USA: Association for Computing Machinery, 2019, pp. 954–968. doi: 10.1145/3352460.3358274.

[2] O. Oleksenko, C. Fetzer, B. Köpf, and M. Silberstein, "Revizor: Testing Black-box CPUs against Speculation Contracts." [Online]. Available: https://arxiv.org/abs/2105.06872

[3] J. Yu, L. Hsiung, M. E. Hajj, and C. W. Fletcher, "Data Oblivious ISA Extensions for Side Channel-Resistant and High Performance Computing," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 808, 2018, [Online]. Available: https://api.semanticscholar.org/CorpusID:52825883

[4] J. Yu, N. Mantri, J. Torrellas, A. Morrison, and C. 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)*, 2020, pp. 707–720. doi: 10.1109/ISCA45697.2020.00064.

[5] Z. N. Zhao *et al.*, "Speculation Invariance (InvarSpec): Faster Safe Execution Through Program Analysis," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1138–1152. doi: 10.1109/MICRO50266.2020.00094.

[6] A. Adir *et al.*, "Genesys-Pro: innovations in test program generation for functional processor verification," *IEEE Design & Test of Computers*, vol. 21, no. 2, pp. 84–93, 2004, doi: 10.1109/MDT.2004.1277900.

[7] M. Lipp *et al.*, "Meltdown: Reading Kernel Memory from User Space," in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, pp. 973–990. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[8] D. Moghimi, "Data Sampling on MDS-resistant 10th Generation Intel Core (Ice Lake)." [Online]. Available: https://arxiv.org/abs/2007.07428

[9] J. Hofmann, E. Vannacci, C. Fournet, B. Kopf, and O. Oleksenko, "Speculation at Fault: Modeling and Testing Microarchitectural Leakage of CPU Exceptions," in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 7143–7160. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/hofmann