USENIX'23 Artifact Appendix: Speculation at Fault: Modeling and Testing Microarchitectural Leakage of CPU Exceptions

A Artifact Appendix

A.1 Abstract

The goal of this artifact is to validate the microarchitectural leakage of CPU exceptions against the formal leakage models we proposed in the paper (named contracts). Concretely, this means reproducing a representative subset of the results described in Table 1 of the paper using our tool Revizor. The exceptions we tested form the rows of this table and the contracts are given as the columns. The contracts are ordered according to permissiveness, i.e., *CT-SEQ* does not allow any transient leakage, whereas *CT-VS-All* allows arbitrary speculative values.

Revizor is a random testing tool, i.e., all test cases are generated randomly. While we observed stable results during our experiments, we therefore cannot 100% guarantee that all results are reproduced within the indicated time frame.

The artifact of this paper includes the source code of Revizor, a set of scripts to run the experiments, and a description of how to run them.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Revizor includes a kernel module that disables the hardware prefetcher and initializes the performance counters. The tool also overwrites the OS-defined IDT to suppress the handling of exceptions on the running core. This may affect other jobs running on your system.

Revizor executes randomly generated programs in kernel space. These programs that are intended to throw exceptions. Even though the executor provides a stable and isolated environment, it may adversely affect the stability of your system.

A.2.2 How to access

The artifact is available on GitHub at https://github.com/vusec/SpeculationAtFault-AE/tree/cf2fa27ff5145a2dedfa8d4302a16d6e32aa5581

A.2.3 Hardware dependencies

Evaluating this artifact requires at least one physical machine with root access. Ideally, the reviewer has access to both one machine with Intel (KabyLake or CoffeeLake) and AMD (Zen+ or Zen3) CPU. If only one such machine is available, the experiments can still be reproduced for just that machine. For AMD Zen2, we expect to obtain the same results as for Zen3. Remote access to some of our machines may be granted upon request. To obtain stable results, the machine(s) should not be actively used by other software.

A.2.4 Software dependencies

- Linux v5.1+ and Kernel Headers
- python 3.9+, python3.9-venv, and pip

A.2.5 Benchmarks

None

A.3 Set-up

In this section, we provide a short version of the installation and configuration steps required to prepare the environment and run Revizor. Please refer to the README file of the repository for detailed installation steps.

A.3.1 Installation

1. Clone the repository.
2. Install software requirements:
```
# on Ubuntu
> sudo apt install linux-headers-$(uname -r)
> sudo apt install python3.9 python3.9-venv
```
3. Install Revizor python package: In the base directory:
```
# on Ubuntu
> cd revizor
> python3 -m venv ~/venv-revizor
> source ~/venv-revizor/bin/activate
```

```
> pip install revizor_fuzzer-1.2.3-py3-none-any.whl
> cd -
```

4. Check installation:
```
> rvzr # Should print the following:
# usage: rvzr {fuzz,analyse,reproduce,minimize,
generate,download_spec} ...
# rvzr: error: the following arguments are
required: subparser_name
```

5. Install the executor:
```
> cd revizor/executor
> make uninstall
> make clean
> make
> make install
> cd -
```

6. Download the ISA spec:
```
> rvzr download_spec -a x86-64 -extensions BASE
SSE SSE2 CLFLUSHOPT CLFSH MPX -outfile base.json
```

A.3.2 Basic Test

From the base directory, on Intel CPU, `cd` into `intel/`. On AMD CPU, `cd` into `amd/`.

Run the basic test:

```
> rvzr fuzz -s ../base.json -c basic/seq-BP.yaml -i
10 -n 100
```

This command will start a small fuzzing campaign testing the Breakpoint exception with 100 test cases, each tested with 10 inputs. The command is expected to terminate without reporting a violation.

A.4 Evaluation workflow

The main results are summarized in Table 1 of the original paper. The evaluation workflow is designed to validate our leakage models. The list of experiments needed to do so depends on the CPU microarchitecture.

A.4.1 Major Claims

For each combination of exception and architecture, the following are the least permissive of our contracts that model the transient leakage induced by that exception.

C1 `#PF` complies with *CT-VS-All* on Intel Kaby Lake, with *CT-VS-NI* on Intel CoffeeLake, and with *CT-DH* on AMD.

C2 `#GP` complies with *CT-VS-CI* on AMD. On Intel, `#GP` does not satisfy any contract.

C3 (Intel only) `#BR` complies with *CT-DH*. (E5)

C4 ucode-assists comply with *CT-SEQ* on AMD, with *CT-VS-All* on Intel Kaby Lake, and with *CT-VS-NI* on CoffeeLake.

C5 `#DE` complies with *CT-VS-Ops* on Intel and AMD Zen3, and with *CT-VS-All* on AMD Zen+.

C6 `#UD`, `#DB`, and `#BP` comply with *CT-SEQ* on all machines.

A.4.2 Experiments

Our experiments serve two purposes: (1) validating our claims regarding which contract satisfies which exception on which machine, and (2) confirming Revizor's effectiveness in generating counterexamples. For each combination of CPU architecture and exception, we therefore propose one experiment that validates the correct contract and one experiment that finds a counterexample for the next more restrictive contract (if one exists).

In the interest of time, we run each experiment for 12h or until a violation is found. The timeout can be increased with the `timeout` option we included in the scripts. For our paper, each experiment ran for 24h. Remember though that Revizor is based on random testing, it is thus possible that a violation is not found within 12h. If this is the case, we suggest to repeat the experiment and increase the timeout.

How-to. We split our experiments according to the type of machine under test. Scripts for the experiments are grouped into a directory for Intel and one for AMD. For example, the scripts to reproduce **Intel E1** are stored inside `./intel/experiment_1/`. Inside each directory, there is one `run.sh` script to start the experiment. An optional timeout (given in seconds) can be set with the `--timeout` option (e.g., `./run.sh --timeout=86400` for a 24h timeout).

Running the script will create a subdirectory `results` inside the experiment directory, where logs are stored. When the script terminates, you can inspect the log to determine whether Revizor detected a violation. Violations (if any) are stored in subdirectories inside `results/violations/`. Each violation directory will contain the program, the inputs, and the configuration file.

Intel. On Intel, our claims can be confirmed with the following experiments.

E1: C1 - page faults - violation [1/2 machine hours]: Test each page fault class (invalid, read-only, SMAP) against *CT-DH*.
Result: violation (for all classes)

E2: C1 - page faults - correct [36 machine hours]: Test each page fault class (invalid, read-only, SMAP) against *CT-VS-NI* on CoffeeLake (and newer), resp. against *CT-VS-All* (on KabyLake and older).
Result: no violation

E3: C2 - non-canonical accesses - violation [12 machine hours]: Test non-canonical accesses against *CT-VS-All*.
Result: violation. Due to the complexity of the contract, finding a violation may take several hours (it was 11h

when we ran the experiment). Please increase the timeout if no violation is found within 12h.

E4: **C3 - Mpx - correct** [12 machine hours]: Test MPX against *CT-DH*.
Result: no violation

E5: **C4 - ucode-assists - violation** [1/6 machine hours]: Test both variants of ucode-assists (Access bit and Dirty bit) against *CT-DH*.
Result: violation (for both variants)

E6: **C4 - ucode-assists - correct** [24 machine hours] Test both variants against *CT-VS-NI* on CoffeeLake (and newer), resp. against *CT-VS-All* (on KabyLake and older).
Result: no violation

E7: **C5 - division - violation** [2 machine hours] Test both types of division errors (divide-by-zero and division overflow) against *CT-VS-NI*.
Result: violation (for both variants)

E8: **C5 - division - correct** [24 machine hours] Test both types of division errors (divide-by-zero and division overflow) against *CT-VS-Ops*.
Result: no violation

E9: **C6 - others - correct** [36 machine hours] Test `#UD`, `#DB` and `#BP` against *CT-SEQ*.
Result: no violation

AMD. On AMD, our claims can be confirmed with the following experiments.

E1: **C1 - page faults - violation** [1/6 machine hours]: Test each page fault class (invalid, read-only, SMAP) against *CT-SEQ*.
Result: violation (for all classes)

E2: **C1 - page faults - correct** [36 machine hours]: Test each page fault class (invalid, read-only, SMAP) against *CT-DH*.
Result: no violation

E3: **C2 - non-canonical accesses - violation** [1/12 machine hours]: Test non-canonical accesses against *CT-DH*.
Result: violation

E4: **C2 - non-canonical accesses - correct** [12 machine hours]: Test non-canonical accesses against *CT-VS-CI*.
Result: no violation

E5: **C4 - ucode-assists - correct** [24 machine hours] Fuzz both variants (Access bit and Dirty bit) against *CT-SEQ*.
Result: no violation

E6: **C5 - division - violation** [2 machine hours] Test both type of division errors (divide-by-zero and division overflow) against *CT-VS-NI*.
Result: violation (for both variants)

E7: **C5 - division by zero - correct** [12 machine hours] Test division-by-zero errors against *CT-VS-Ops* on Zen3 (or newer), resp. against *CT-VS-All* on Zen+ (or older). For Zen2 (which was not part of our setup), we expect *CT-VS-Ops* to hold as well.
Result: no violation

E8: **C5 - division overflow - correct** [12 machine hours] Test division overflows against *CT-VS-Ops*.
Result: no violation

E9: **C6 - others - correct** [36 machine hours] Test `#UD`, `#DB` and `#BP` against *CT-SEQ*.
Result: no violation

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.