TLB;DR: Enhancing TLB-based Attacks with TLB Desynchronized Reverse Engineering

Andrei Tatar, *Vrije Universiteit, Amsterdam;* Daniël Trujillo, *Vrije Universiteit, Amsterdam, and ETH Zurich;* Cristiano Giuffrida and Herbert Bos, *Vrije Universiteit, Amsterdam*

https://www.usenix.org/conference/usenixsecurity22/presentation/tatar

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

D Artifact Appendix

D.1 Abstract

The artifact reproduces the reverse engineering experiments outlined in §4 and §A with results summarized in Table 1 and Table 2, the computation of optimized eviction sets presented in §5, as well as the case studies discussed in §6. Specifically, we provide 3 code trees: `TLB/` contains Linux kernel modules used for reverse engineering TLB properties, along with helper userspace programs; `cache-ninja/` models set-associative caches with replacement policies and computes optimized eviction sets; `case-studies/` contains the case study experiments. The hardware used for `TLB` is detailed in Table 1 and Table 2, whereas the case studies were developed for Intel Kaby Lake processors. Finally, `cache-ninja` is architecture independent and can run on any computer. All our source code is available at https://github.com/vusec/tlbdr.

D.2 Artifact check-list (meta-information)

- **Compilation:** `cache-ninja` requires a rust compiler and cargo version $\geq$ 1.49. `TLB` and `case-studies` work with the system gcc.
- **Run-time environment:** the kernel modules in `TLB` and `TLB/AMD` are programmed against kernel versions 5.4 through 5.18. The kernel module in `TLB/PCID` assumes kernel version 5.4. Module insertion requires root access and a policy allowing loading of unsigned modules.
- **Hardware:** `case-studies` require an Intel Kaby Lake i7-7700 CPU with Hyperthreading enabled. Other Kaby Lake CPUs might be usable with small tweaks. `TLB` runs on the architectures shown in Table 1 and Table 2.
- **Execution:** The reverse engineering experiments under `TLB` are best run on a quiescent system, or at the very least one idle core. The experiments under `case-studies` are best run pinned on idle cores, ideally enforced via e.g., cpusets.
- **Output:** Each piece of code produces bespoke output, usually textual, representing its results. We describe this output in more detail in the README file of each directory, and provide tools to process this output.
- **Experiments:** The README file under each directory provides instructions on how to set up and run each experiment. A convenience script and/or Makefile is also included.
- **How much disk space required (approximately)?:** a few MiB for source and build.
- **How much time is needed to prepare workflow (approximately)?:** a few minutes for each experiment, mostly for setting up environment.
- **How much time is needed to complete experiments (approximately)?:** runtime usually depends on the number of measurements taken, which can be adjusted; by default the reverse engineering experiments in `TLB` and `TLB/PCID` run within 2–3 minutes, the experiments under `TLB/AMD` may take up to 30 minutes, `cache-ninja` runs within a minute, `case-studies/anc` takes up to one hour, while `case-studies/pthammer` and `case-studies/tlbleed` take around 10 minutes each.
- **Publicly available (explicitly provide evolving version reference)?:** All the source code is available at https://github.com/vusec/tlbdr.
- **Code licenses (if publicly available)?:** The kernel modules under `TLB` and `case-studies/pthammer/ptsim` are licensed GPLv2, the rest of the code is licensed under Apache 2.0.
- **Archived (explicitly provide DOI or stable reference)?:** The `sec22-ae-final` git tag marks the tree with the artifacts submitted for evaluation.

D.3 Description

D.3.1 How to access

All the source code is available at https://github.com/vusec/tlbdr, under the tag `sec22-ae-final`.

D.3.2 Hardware dependencies

- `TLB`: The experiments in `TLB` and `TLB/PCID` were run on the CPUs listed in Table 1, while the experiments under `TLB/AMD` were run on the CPUs listed in Table 2.
- `case-studies`: The experiments make microarchitectural assumptions that require an Intel Kaby Lake CPU and were run on an i7-7700K, as described in more detail in §6. A different model within the same family should also work, but might require some manual parameter tuning.

D.3.3 Software dependencies

- `TLB`: Building the kernel module requires kernel headers and system build tools. Running `prepare.sh` will install these dependencies on Ubuntu systems.
- `cache-ninja`: Building and running require rust and cargo version $\geq$ 1.49; earlier versions may work, although not tested.
- `case-studies`: Building and running requires a C compiler and, depending on the experiment, kernel headers and Python.

D.4 Installation

- `TLB`: Run `make` to build the kernel module and `insmod mmuctl/mmuctl.ko` to insert the kernel module (as root). To run the `PCID` or `AMD` experiments, navigate to the corresponding subdirectory before running the commands.
- `cache-ninja`: Run `cargo build [--debug|--release]` to compile the binary. Internet access might be required for cargo to download dependencies.
- `case-studies`: Run `make` in each case study subdirectory to build the experiment.

D.5 Experiment workflow

- `TLB`: Start with a freshly booted Linux system running on bare metal and ensure the build dependencies are satisfied. Run `make test`; this will build the experiment kernel module, load the module and execute the trigger binary. If the kernel module is already inserted it sufficies to run the trigger binary directly. After the experiment run `make unload` to remove the module from the kernel. To run the `PCID` or `AMD` experiments, navigate to the corresponding subdirectory before running the commands. Warning: these experiments run kernel code that might crash/hang a core or otherwise leave the kernel in an invalid state, we recommend you monitor the kernel log and immediately reboot your system after any error.
- `cache-ninja`: Run `cargo run` to build and run the program, which then computes and prints the optimized eviction sets discussed in §5.
- `case-studies`: Execute `make run` in each directory to run experiments with default settings. Consult the individual README files under each directory for detailed instructions.

D.6 Evaluation and expected results

In this work we make several main claims: (i) We introduce TLB desynchronization as a reliable primitive for TLB reverse engineering and validate it against previous work; (ii) We show how TLB desynchronization, due to its precision and reliability, can be used to reverse engineer previously undocumented TLB features; (iii) We show how knowledge of one of these properties—replacement policies—enable knowledgeable manipulation of TLB state, leading to vastly more efficient adversarial evictions; and (iv) We show how these more efficient adversarial evictions bring significant improvements to various classes of attacks that make use of TLB eviction.

- `TLB` is the reverse engineering code using TLB desynchronization which supports claims (i) and (ii).
- `cache-ninja` implements a model of TLB state along with the replacement policies that we reverse engineered, and uses this model to compute optimal adversarial eviction sets, validating claim (iii).
- `case-studies` implements proofs-of-concept for integrating the previously computed optimal eviction sets into several existing attacks, in support of claim (iv).

We now describe the goal and expected output of each of our experiments.

- `TLB`
  - *Goal:* Measure TLB properties.
  - *Implementation:* As described in §4 and §A.
  - *Results:* On the relevant systems, we expect results in line with Table 1 and Table 2.
- `cache-ninja`
  - *Goal:* Produce optimal eviction sets for a Kaby Lake TLB
  - *Implementation:* As described in §5, using BFS to search through the state graph.
  - *Results:* We expect the optimized eviction sets as presented in §5.2, §5.3, and §6.2.
- `case-studies/anc`
  - *Goal:* Measure and compare the speed of AnC attacks using naive and optimized eviction sets.
  - *Implementation:* As described in §6.1.
  - *Results:* On relevant hardware we expect results comparable to Figure 7.
- `case-studies/pthammer`
  - *Goal:* Reproduce Figure 8 from raw data; optionally produce a histogram of all data.
  - *Implementation:* As described in §6.1.
  - *Results:* We expect a faithful rendering of Figure 8. The histogram should also show distinctly tri-modal output, as described in §6.1.
- `case-studies/pthammer/ptsim`
  - *Goal:* Simulate, measure and compare the potential hammer rate of a PTHammer-like attack using naive and optimized eviction sets.
  - *Implementation:* As described in §6.1.
  - *Results:* On relevant hardware we expect results similar to those described in §6.1 and shown in Figure 8.
- `case-studies/tlbleed`
  - *Goal:* Measure and compare the speed of TLBleed-style attacks using naive or optimized eviction sets.
  - *Implementation:* As described in §6.2.
  - *Results:* On relevant hardware, we expect raw sample rates in line with those shown in Table 5. Similarly, we expect peak sustained covert channel bandwidth in line with that described in §6.2.

D.7 Experiment customization

Customization and tweaking of experiments is possible to some degree, consult the individual README files of each subdirectory for more details.

D.8 Version

Based on the LaTeX template for Artifact Evaluation V20220119.