USENIX Security '26 Artifact Appendix: Charting the Cache Side-Channel Frontier: A Systematic Study of Eviction Set Construction on Apple Silicon

Han Wang[†] Yakun Wu[†] Yusi Feng[†] Yinqian Zhang[†‡]
Southern University of Science and Technology

A Artifact Appendix

A.1 Abstract

This paper presents a systematic study of eviction-set construction on Apple Silicon, focusing on the Apple M4 microarchitecture. Eviction sets are a fundamental primitive for conflict-based cache side-channel attacks, but constructing reliable eviction sets on recent Apple Silicon is challenging due to undocumented microarchitectural behavior. Our work reverse engineers the root causes of these challenges, including an SLC victim buffer that obscures latency-based verification, a DRAM-to-Cache Prefetcher that disrupts strict LRU assumptions, and an opaque SLC allocation policy influenced by L2 insertion and promotion behavior.

The artifact is designed to support the main experimental claims of the paper. It includes benchmark programs, reverse-engineering experiments, eviction-set construction algorithms, data-processing scripts, and plotting scripts. Through the artifact, evaluators can reproduce the characterization of key Apple M4 cache behaviors, validate the effectiveness of our optimized L2 eviction-set construction strategy, and evaluate the hierarchical method used to construct fine-grained SLC eviction sets.

Because the results depend on Apple M4-specific microarchitectural behavior, full artifact evaluation requires evaluator-side access to a local Apple M4 machine. In the absence of such hardware, the artifact remains useful for reviewing the implementation and experimental workflow, but the main timing measurements and eviction-set construction results cannot be fully reproduced. On the target platform, the artifact provides the necessary code and scripts to reproduce the paper's core cache-behavior observations, timing measurements, and construction results. Exact timing distributions and success rates may vary slightly depending on system noise, OS configuration, core placement, and runtime conditions.

[‡]Corresponding author.
[†]Affiliated with the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering.

A.2 Description & Requirements

This artifact accompanies *Charting the Cache Side-Channel Frontier: A Systematic Study of Eviction Set Construction on Apple Silicon*. It contains the source code, benchmark programs, eviction-set construction algorithms, reverse-engineering experiments, data-processing scripts, and plotting scripts used to reproduce the main experimental results in the paper. The artifact is organized by experiment modules, covering cache-latency characterization, DRAM-to-Cache Prefetcher (DCP) analysis, L2 Insertion/Promotion Vector (IPV) and SLC allocation characterization, optimized L2/SLC eviction-set construction, and downstream security-evaluation experiments.

The main experimental setup used in the paper is an Apple M4 Mac mini. The experiments rely on high-resolution cycle measurements through the Apple PMU register `S3_2_c15_c0_0` and on data-cache maintenance instructions such as `dc civac`. Therefore, evaluators should prepare a dedicated Apple Silicon machine and enable these instructions following the PacmanPatcher setup described below.

A.2.1 Security, privacy, and ethical concerns

This artifact is security-sensitive and dual-use. It studies cache side channels and eviction-set construction on Apple Silicon. The released artifact is scoped to reproduce the paper's scientific claims, including microarchitectural measurements and eviction-set construction, and is not intended to provide a weaponized exploit or a turnkey exploitation pipeline.

Running the artifact may require patching the macOS kernel to enable user-space access to performance counters and cache-maintenance instructions. This can reduce system security, may require changes to boot/security settings, and can cause system instability, crashes, or boot failures. Evaluators should run the artifact only on a dedicated, non-production machine that they own or are explicitly authorized to use. The machine should not contain sensitive user data, and evaluators should keep a recovery plan and backup before applying any kernel patch.

The experiments operate on locally allocated memory buffers and do not require third-party accounts, network ser-

vices, personal data, or access to external systems. Evaluators should not run the artifact on shared, production, or third-party systems and should avoid collecting data from non-consenting users or workloads.

A.2.2 How to access

A stable archived snapshot of the artifact is available at:

https://doi.org/10.5281/zenodo.20336892

A.2.3 Hardware dependencies

The main results require an Apple M4 Mac mini, matching the experimental setup used in the paper. The relevant microarchitectural features are the Apple M4 CPU cache hierarchy, including private L1 caches, the shared L2 cache within a CPU cluster, and the system-level cache (SLC). The paper's measurements assume 128-byte cache lines, a 16 MiB L2 cache, and an 8 MiB SLC on the evaluated M4 platform.

A machine with at least 16 GiB of RAM is recommended, because some L2 candidate-pool experiments sweep candidate pools up to 1 GiB. No GPU, FPGA, or special accelerator is required. Physical or administrative access to the machine is required because the setup may involve installing a patched kernel collection and booting through macOS recovery/1TR mode.

A.2.4 Software dependencies

Please set up the environment by following the instructions in the PacmanPatcher code repository: https://github.com/jprx/PacmanPatcher.

In addition, the artifact requires the following software environment:

- macOS on Apple Silicon.
- The Apple Kernel Debug Kit (KDK) matching the evaluator's macOS build, as required by PacmanPatcher.
- A C/assembly build environment for ARM64/AArch64 code.
- Python 3 for data processing and plotting.
- Python packages required by the plotting and analysis scripts, such as `numpy`, `pandas`, `scipy`, and `matplotlib`.

A.2.5 Benchmarks

None.

A.3 Set-up

This section describes the installation and configuration steps needed to prepare the evaluation environment.

A.3.1 Installation

Set up PacmanPatcher by following the instructions in: https://github.com/jprx/PacmanPatcher. In particular, download the KDK matching the macOS build, build PacmanPatcher, patch the kernel as instructed, create a patched kernel collection, and boot the machine using the patched kernel collection. After booting into the patched kernel, verify that PMU access and cache-maintenance instructions work:

```
cd <PacmanPatcher-root>
./test_pmc
./test_flush
```

Both tests should complete without illegal-instruction failures or crashes. The PMU test should show that cycle-counter reads are available from user space, and the flush test should show that cache-maintenance instructions are executable from user space.

Each experiment module should be built according to its local README or Makefile. Build the relevant module before running it, for example:

```
cd <artifact-root>/01_Benchmark/01_cache_latency
make
```

A.3.2 Basic Test

A minimal functionality test should verify three things: PMU access, cache flush support, and the ability to compile and execute one benchmark.

Step 1: Run the PacmanPatcher checks:

```
cd <PacmanPatcher-root>
./test_pmc
./test_flush
```

Expected output: The tests should terminate normally. The PMU test should report usable cycle-counter behavior, and the cache-flush test should confirm that cache-maintenance instructions are available from user space.

Step 2: Build and run a small cache-latency benchmark:

```
cd <artifact-root>/01_Benchmark/01_cache_latency
make
python3 run_test_dcache_single_avarage.py
python3 plot_test_dcache_single.py
```

Expected output: The benchmark should produce timing samples or an intermediate result file. The corresponding analysis/plotting script should classify accesses into latency regions consistent with the paper: L1/L2 hits below roughly 130 cycles, SLC hits around 170–300 cycles, and DRAM accesses around 400–600 cycles.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): C2SCF characterizes the Apple M4 cache hierarchy and identifies latency regions corresponding to L1/L2

hits, SLC hits, SLC victim-buffer accesses, and DRAM accesses. It further shows that random access leads to low effective SLC utilization and that the SLC victim buffer masks eviction events. This is proven by the experiments in Sec. 3, whose results are shown in Fig. 1 and Fig. 2.

(C2): C2SCF uncovers a DRAM-to-Cache Prefetcher (DCP) on Apple M4. The DCP can reload flushed data into the private cache hierarchy, can be triggered by a small set of precursor accesses, follows constrained spatial patterns, and can interfere with Flush+Reload-style decoders. This is proven by the experiments in Sec. 4, whose results are shown in Fig. 3–Fig. 6 and discussed in Sec. 4.5.

(C3): C2SCF shows that L2 replacement behavior and SLC allocation are coupled through L2 Insertion/Promotion Vectors (IPVs). Random access patterns receive higher residency priority than strided patterns, repeated evictions can elevate cacheline priority, and high-IPV evictions tend to trigger rapid propagation toward DRAM rather than stable SLC residency. This is proven by the PIP and I2P experiments in Sec. 5, whose results are shown in Fig. 8, Fig. 10, Fig. 11, and Table 1.

(C4): C2SCF's Apple-specific L2 construction optimizations substantially improve L2 eviction-set construction. Generic PPP fails under Apple M4 noise, while optimized PPP reaches the reported L2 eviction-set construction success rate of 88.2%. This is proven by the experiments in Sec. 6.1–Sec. 6.2, whose results are shown in Fig. 12, Fig. 13, Table 2, Table 3, and Table 4.

(C5): C2SCF constructs the first practical fine-grained SLC eviction sets on Apple Silicon using hierarchical filtering. The optimal SLC construction reaches the reported success rate of 90%, and the resulting primitives provide useful temporal/spatial resolution for downstream side-channel evaluation. This is proven by the experiments in Sec. 6.3–Sec. 6.5, whose results are shown in Fig. 14–Fig. 17.

A.4.2 Experiments

For each experiment, the exact executable names, command-line arguments, output files, and plotting commands are provided in the corresponding module-local README.

(E1): Cache hierarchy and SLC victim-buffer characterization. This experiment supports claim C1.
How to: Use the benchmark modules `01_Benchmark/01_cache_latency`, `01_Benchmark/02_cache_flush_latency`, and `01_Benchmark/03_slc_victim_buffer`.
Preparation: Build the C benchmark programs in each directory according to the module-local README or Makefile. Ensure that PMU timing and cache flushing work by running `test_pmc` and `test_flush` from the PacmanPatcher setup.
Execution: Run the dense random-access benchmark, the flush-after-random-access benchmark, and the sparse conflict-access benchmark. The sparse conflict-access benchmark should use a page-size interval to induce high contention and expose the SLC victim-buffer behavior.
Results: Process the generated timing data with the corresponding Python scripts. The expected outcome is the reproduction of Fig. 1a, Fig. 1b, Fig. 1c, Fig. 2a, and Fig. 2b. The results should show distinct latency regions for L1/L2, SLC, and DRAM, low effective SLC utilization under random access, anomalous low-latency hits after flushing, and a split in the SLC-latency range consistent with an SLC victim buffer.

(E2): DCP behavior, triggering pattern, mitigation, and case study. This experiment supports claim C2.
How to: Use the modules `02_DCP/01_behavior`, `02_DCP/02_minimal`, `02_DCP/03_pattern`, `02_DCP/04_mitigation`, and `02_DCP/05_case`.
Preparation: Build the benchmark programs and activate the Python environment.
Execution: First, run the DCP behavior experiment using pseudo-random linear-shift access and fully random access patterns. Next, run the reduction-based minimal-trigger experiment to identify precursor cachelines. Then run the spatial-pattern enumeration experiment across different address intervals. Finally, run the mitigation experiment with temporal delays and address offsets, and run the Flush+Reload decoder case study.
Results: Process and plot the results to reproduce Fig. 3, Fig. 4, Fig. 5, and Fig. 6, and to reproduce the qualitative result discussed in Sec. 4.5. The expected outcome is that forced flushing does not always lead to DRAM-latency reloads, because the DCP can bring data back into the cache hierarchy. The minimal-trigger results should show that a small precursor sequence can activate the DCP. The spatial-pattern results should show constrained power-of-two-style address deltas. The mitigation results should show that large address intervals and temporal perturbation reduce DCP interference.

(E3): L2 IPV dynamics and SLC allocation policy. This experiment supports claim C3.
How to: Use the modules `03_IPV/01_PIP`, `03_IPV/02_I2P/01_heatmap`, and `03_IPV/02_I2P/02_transition`.
Preparation: Build the PIP and I2P benchmark programs.
Execution: Run the PIP experiment groups that vary prime, interfere, and probe patterns. Then run the I2P experiment over multiple rounds and working-set sizes to track how lines move through L2, SLC, the victim buffer, and DRAM.
Results: Generate the plots and tables corresponding to Fig. 8, Fig. 10, Fig. 11, and Table 1. The expected outcome is that random priming yields stronger L2 resi-

dency than strided priming, strided interference is less effective at evicting high-priority resident lines, repeated access/eviction cycles change retention behavior, and SLC allocation depends on the L2 state.

(E4): Optimized L2 eviction-set construction. This experiment supports claim C4.

How to: Use the modules `04_EV/01_L2/01_access_frequency`, `04_EV/01_L2/02_backtrack`, `04_EV/01_L2/03_candidates`, and `04_EV/01_L2/04_random`.

Preparation: Build the L2 eviction-set construction programs.

Execution: Run the baseline and optimized L2 construction algorithms, including SHM, GE, CT, and PPP variants. Evaluate the effects of access randomization, access frequency, backtracking or majority-voting robustness, and candidate-pool size.

Results: Reproduce Fig. 12, Fig. 13, Table 2, Table 3, and Table 4. The expected outcome is that generic PPP performs poorly on Apple M4 unless the Apple-specific optimizations are applied. With the optimized parameters, PPP should reproduce the reported high L2 eviction-set construction success rate, while large candidate pools should demonstrate the degradation caused by DCP interference.

(E5): Hierarchical SLC eviction-set construction and downstream security evaluation. This experiment supports claim C5.

How to: Use the modules `04_EV/02_SLC` and `04_EV/04_sec_eva`.

Preparation: First construct valid L2 eviction sets, because the SLC construction uses L2 filters as part of the hierarchical filtering process. Configure the number of L2 filters and select the construction strategy to evaluate, such as GE+GE, PPP+GE, or PPP+PPP.

Execution: Run the hierarchical SLC construction experiment over the configured number of L2 filters. Then run the stability evaluation by repeatedly applying the eviction sequence to successful eviction sets. Finally, run the temporal-resolution, spatial-resolution, and AES T-table cache-line classification experiments.

Results: Reproduce Fig. 14–Fig. 17 and the results discussed in Sec. 6.5. The expected outcome is that SLC construction success improves as the number of L2 filters increases, the best SLC construction reproduces the reported 90% success rate, and the optimized L2 eviction sets provide clear cache-line-level spatial resolution. The downstream AES T-table benchmark should show substantially better classification accuracy for the optimized construction than for the minimally adapted PPP baseline.

A.5 Notes on Reusability

The artifact is organized so that individual modules can be reused independently. For example, the `01_Benchmark` modules can be reused to calibrate cache latency thresholds on a new Apple Silicon machine, the `02_DCP` modules can be reused to study prefetcher behavior, and the `03_IPV` modules can be reused to characterize replacement and allocation policies.

When porting the artifact to another Apple M-series chip or another ARM64 platform, users should first rerun the cache-latency calibration benchmarks and update the latency thresholds, cache sizes, cache-line assumptions, and candidate-pool parameters accordingly. The eviction-set construction modules should then be rerun with the calibrated parameters. Results may differ across SoCs and macOS versions because the relevant prefetchers, replacement policies, PMU access mechanisms, and cache-maintenance behavior are microarchitecture- and OS-dependent.

For safe reuse, the artifact should remain limited to controlled local experiments on owned or authorized hardware. Users should avoid integrating the artifact into automated exploit pipelines or running it against third-party systems or sensitive workloads.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.