USENIX Security '26 Artifact Appendix: UncoreBleed: AEX-Free, High-Resolution, and Low-Noise Side-Channel Attacks on SGX Enclaved Execution

Decheng Chen, Zhi Zhang, Zhenkai Zhang, Xin Zhang, Yansong Gao, Yi Zou

A Artifact Appendix

A.1 Abstract

This artifact appendix describes the reproduction of the results from the paper *UncoreBleed: AEX-Free, High-Resolution, and Low-Noise Side-Channel Attacks on SGX Enclaved Execution*. This appendix includes the information needed to run the artifact and validate the claims made in the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

This artifact requires root privileges for PCIe configuration, CR0.CD modification, and kernel module loading. Evaluators should execute the artifact only on a dedicated experimental machine, since some steps may affect system stability or performance isolation. The artifact does not collect personal data and is intended solely for research and evaluation purposes on hardware owned or controlled by the evaluator.

A.2.2 How to access

The artifact is publicly available at the following stable URL: https://zenodo.org/records/17902416

A.2.3 Hardware dependencies

The following hardware is required:

- A 3rd, 4th, or 5th generation Xeon Scalable CPU with SGX support.
- Recommended processors:
  - Xeon Silver 4314 (Ice Lake-SP)
  - Xeon Bronze 3408U (Sapphire Rapids-SP)
  - Xeon Silver 4514Y (Emerald Rapids-SP)

A.2.4 Software dependencies

The following software is required:

- Linux kernel 5.4+ with SGX enabled (5.15 recommended)
- Intel SGX SDK 2.17+ installed and configured
- GCC 9.0 or newer
- CMake 3.16 or newer
- Root privileges for PCIe configuration, CR0.CD modification, and module loading

A.2.5 Benchmarks

This artifact does not rely on third-party benchmark suites. Instead, it includes custom workloads, demos, and experiment inputs developed for the evaluation of the claims in the paper. The artifact includes the following experimental components:

- A microbenchmark for comparing core and uncore PMCs during enclave and non-enclave execution.
- A PKT_MATCH analysis workload for studying programmable address filtering and 64-byte granularity.
- An address-bit flipping workload for reverse engineering physical-address-to-M2M mapping.
- A Libjpeg-based demo for recovering pictures from SGX enclaved execution.
- An RSA demo with TLBlur for demonstrating private-key extraction.

A.3 Set-up

A.3.1 Installation

Download the artifact from:

https://zenodo.org/records/17902416

Then extract the package and enter the root directory.

A.3.2 Basic Test

Run the following command in the root directory:

```
./detect_cpu.sh
```

If your CPU belongs to one of the supported generations, the script will print a message indicating compatibility. Otherwise, your platform may not support the artifact.

A.4 Evaluation workflow

A.4.1 Major Claims

- (C1): SGX enclaves are not entirely excluded from performance monitoring. While core PMCs are disabled, uncore PMCs remain active and expose enclave-correlated events.
- (C2): The uncore `PKT_MATCH` event supports programmable address filtering and achieves 64-byte granularity, enabling fine-grained monitoring of enclave memory accesses.
- (C3): The reverse-engineered `PKT_MATCH` event is consistently available across multiple generations of SGX-capable Xeon processors, despite being undocumented or inaccurately described in Intel manuals.
- (C4): As the first PMC-based side channel against SGX, UncoreBleed successfully recovers pictures during Libjpeg decompression, and extracts RSA private keys from a single decryption in production enclaves in the presence of TLBlur, the most recent software defense on off-the-shelf SGX platforms.

A.4.2 Experiments

(E1) Core vs. Uncore PMC Comparison [2 human-minutes + 10 compute-minutes]
This experiment validates that uncore PMCs remain active during SGX enclave execution, while core PMCs are largely suppressed. It supports claim (C1).

Preparation: Navigate to the `MicroBenchmark/` directory and read the `README.md` file. Ensure that the machine satisfies the SGX and privilege requirements described above.

Execution: Run the benchmark using `./run.sh` to sample both core and uncore events in enclave and non-enclave settings. After sampling completes, run `./summarize_perf.py` to summarize the collected measurements.

Results: The evaluator should compare the summarized event counts between enclave and non-enclave executions. The expected outcome is that core events such as `INSTRUCTIONS` and `BRANCHES` differ significantly between the two settings, while uncore events such as `TOR_INSERT`, `TAG_HIT`, `ACT_COUNT`, and `CAS_COUNT` remain within the same order of magnitude.

(E2) PKT_MATCH Event Analysis [1 human-minute + 10 compute-minutes]
This experiment reverse engineers the `PKT_MATCH` event and validates its filtering mechanism. It supports claim (C2).

Preparation: Navigate to the `EventAnalysis/` directory and read the `README.md` file. Confirm that the platform allows configuring and sampling the relevant uncore events.

Execution: Run `./Analysis` to generate high-frequency read/write operations, configure the `PKT_MATCH` filter, and collect event-count data. After the data is collected, run `./analyze_record.py` to analyze the resulting records.

Results: The evaluator should inspect the analyzed event-count output. The expected outcome is that `PKT_MATCH` reports DRAM traffic only when read/write operations occur near the physical address configured in the filter, with an effective granularity of 64 bytes.

(E3) Uncovering Address-to-M2M Mapping [5 human-minutes + 15 compute-minutes]
This experiment reverse engineers the physical-address-to-M2M mapping on the target platform. It supports claim (C3).

Preparation: Navigate to the `ReverseM2MMap/` directory and read the `README.md` file. `README.md` file.

Execution: Run `./ReverseMap.py`. This script flips selected address bits and monitors `PKT_MATCH` observations across M2M boxes in order to infer how memory accesses are routed.

Results: The evaluator should examine the output mapping generated by the script. The expected outcome is a recovered physical-address-to-M2M mapping consistent with the mapping pattern reported in Figure 6 of the paper.

(E4) UncoreBleed Attack on SGX-Based Libjpeg [15 human-minutes + 30 compute-minutes]
This experiment demonstrates picture recovery from enclaved Libjpeg execution using UncoreBleed. It supports claim (C4).

Preparation: Navigate to the `Demo/libjpeg/` directory and read the `README.md` file. Ensure that the SGX environment and all required dependencies are installed correctly.

Execution: Run `./run.sh`. The script automatically reads an example input image, captures the `PKT_MATCH` event stream, and processes the collected event counts to recover the memory-access trace.

Results: The evaluator should inspect the final recovered output produced by the script. The expected outcome is a recovered image qualitatively similar to the result shown in Figure 10 of the paper.

(E5) UncoreBleed Attack on RSA with TLBlur [30 human-minutes + 30 compute-minutes]
This experiment demonstrates RSA private-key extraction using UncoreBleed against a TLBlur-protected enclave. It supports claim (C4).

Preparation: Deploy TLBlur on the experimental platform. Follow the installation instructions provided by the TLBlur

open-source repository. Then navigate to the `Demo/RSA/` directory and read the `README.md` file.

Execution: Run `./run.sh`. The script captures `PKT_MATCH` events during RSA execution and then parses the recorded data to recover key-related information.

Results: The evaluator should inspect the script output and compare it with the paper. The expected outcome is a result qualitatively similar to Figure 11 of the paper, showing successful extraction of a RSA private key from a TLBlur-protected enclave.

A.5 Notes on Reusability

The artifact can also be reused to study uncore performance monitoring behavior across multiple Xeon generations, and to support further research on SGX side channels and uncore event reverse engineering.

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/.