SpectrEM: Exploiting Electromagnetic Emanations During Transient Execution

Jesse De Meulemeester, Antoon Purnal, Lennert Wouters,
Arthur Beckers, and Ingrid Verbauwhede, *COSIC, KU Leuven*

https://www.usenix.org/conference/usenixsecurity23/presentation/de-meulemeester

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

USENIX'23 Artifact Appendix: SpectrEM: Exploiting Electromagnetic Emanations During Transient Execution

Jesse De Meulemeester
COSIC, KU Leuven

Antoon Purnal
COSIC, KU Leuven

Lennert Wouters
COSIC, KU Leuven

Arthur Beckers
COSIC, KU Leuven

Ingrid Verbauwhede
COSIC, KU Leuven

D Artifact Appendix

D.1 Abstract

This appendix describes our artifacts for SpectrEM, a physical transient execution attack. In our paper, we discuss how the physical effects of transient instructions can be leveraged to extract secret information. In this artifact, we provide the source code for our proof-of-concept implementations and the scripts to take and evaluate the traces. To allow the reproduction of our work, we also provide the side-channel traces that were used to produce the results in our paper.

D.2 Description & Requirements

D.2.1 Security, privacy, and ethical concerns

None.

D.2.2 How to access

Our artifact consists of two repositories. The first one, hosted on GitHub, contains all source code and scripts related to our POC implementations. This repository can be accessed at `https://github.com/KULeuven-COSIC/SpectrEM/tree/1c0207db3d55580b7f31dfb22f57100ea5544707`.

Additionally, to enable the reproduction of our results, we also provide the side-channel traces for our work in KU Leuven's Research Data Repository (RDR). This dataset can be accessed at `https://doi.org/10.48804/AHTI1A`.

D.2.3 Hardware dependencies

To allow the reproduction of our results without requiring an extensive EM side-channel setup, we provide our pre-recorded traces, as discussed above. Therefore, we do not require any specific hardware setup.

To download all pre-recorded traces, at least 520 GB of free disk space is required. Additionally, the provided Python scripts to evaluate these traces use up to 12 GB of RAM. We, therefore, recommend at least 16 GB of RAM.

D.2.4 Software dependencies

The evaluation of the side-channel traces was performed using Python 3.11.4. The exact required Python packages are detailed in our GitHub repository.

D.2.5 Benchmarks

None.

D.3 Set-up

D.3.1 Installation

1. Create a new Python environment and install all dependencies as described in `readme.md` in our GitHub repository.

2. Download the traces from the data repository. We provide a Python script to download these folders in our GitHub repository (`traces/download-traces.py`). Specifically, download the following directories:

   - `0-base-experiments/` (33 GB)
   - `1-additional-experiments/` (39 GB)
   - `2-reducing-assumptions/` (21 GB)
   - `3-case-study/` (2 GB)

   The directory containing the MLP training data (`4-mlp-data/` (89 GB)) may also be downloaded but is not required as we provide pre-trained MLP networks along with the evaluation traces.

D.3.2 Basic Test

Activate the created Python environment and start Jupyter Notebook. Open the notebook `scripts/evaluate/evaluate_extraction_methods.ipynb` and run the first cell. If no errors are displayed, all packages are installed correctly.

To make sure the traces are downloaded to the correct location, step through the notebook. If the traces are downloaded to a different location, the path pointing to these traces can be changed by modifying the `prerecorded_traces_dir` variable.

If no errors are encountered when stepping through this Jupyter Notebook, everything is set up correctly.

D.4 Evaluation workflow

D.4.1 Major Claims

(C1): Variable-time instructions and control flow dependencies enable physical transient execution attacks. In optimal conditions, both SpectrEM and MeltEMdown can achieve low BER, even when only considering a single trace (cf. Section 6 of our paper).

(C2): The simplifications introduced for evaluation can be removed by additional post-processing (cf. Section 7 of our paper). Specifically, we show that SpectrEM attacks can still be carried out when the clock frequency is not locked, when the POC is not pinned to a specific core, and when using cache thrashing.

(C3): The code pattern that forms control flow gadgets can be found in OpenSSH. With only minor changes, the two uncovered gadgets can be exploited through the network interface (cf. Section 8 of our paper).

D.4.2 Experiments

Before running the experiments, we recommend stepping through the following Jupyter notebook: `scripts/evaluate/evaluate_extraction_methods.ipynb` [30 human-minutes + 10 compute-minutes]. This notebook details how the traces are evaluated. For each of the following experiments, we provide Python scripts that use the techniques discussed in this notebook to automatically evaluate the traces.

(E1.1): [Base experiments] [15 human-minutes + 20 compute-minutes + 92 GB disk + 4.2 GB RAM]: This experiment evaluates the baseline performance of the SpectrEM and MeltEMdown POCs.

Preparation: Download the traces in folder `0-base-experiments` from the data repository.

Execution: Run the following Python script: `scripts/reproduce/0-base-experiments.py`. This script will output the BERs for each POC.

Results: The Python script will print the BERs for the 5 different base POCs. The expected outputs are included in `scripts/readme.md`. These results can be compared with the results in our paper in Table 1 and Section 6.2.

(E1.2): [Additional experiments] [15 human-minutes + 20 compute-minutes + 110 GB disk + 4.5 GB RAM]: This experiment evaluates the performance of the POCs under different numbers of training packets and different numbers of `udiv` instructions.

Preparation: Download the traces in folder `1-additional-experiments` from the data repository.

Execution: Run the following Python script: `scripts/reproduce/1-additional-experiments.py`. This script will output the BERs for each POC.

Results: The Python script will print the BERs for the different conditions and produce two figures. The expected outputs are included in `scripts/readme.md`. These results can be compared with the results in our paper in Figure 5 and Figure 7.

(E2): [Reducing assumptions] [15 human-minutes + 2 compute-hours + 58 GB disk + 12 GB RAM]: This experiment evaluates the effect of removing the evaluation assumptions.

Preparation: Download the traces in folder `2-reducing-assumptions` from the data repository.

Execution: Run the following Python script: `scripts/reproduce/2-reducing-assumptions.py`. This script will output the BERs for each experiment.

Results: The Python script will print the BERs for the 5 different base POCs. The expected outputs are included in `scripts/readme.md`. These results can be compared with the results in our paper in Table 1 and Section 6.2.

(E3): [Case study] [30 human-minutes + 2 compute-minutes + 5 GB disk + 2.8 GB RAM]: This experiment evaluates the performance of two real-world code patterns in OpenSSH.

Preparation: Download the traces in folder `3-case-study` from the data repository.

Execution: Verify that the two code snippets in Listings 4 and 5 in our paper are indeed taken from the latest version of OpenSSH at the time of submission (9.3). To evaluate the performance of these two gadgets, run the script `scripts/reproduce/3-case-study.py`.

Results: The Python script will print the BERs for the two gadgets. The expected outputs are included in `scripts/readme.md`. These results can be compared with the results in our paper in Section 8.

D.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/.