Collide+Power: Leaking Inaccessible Data with Software-based Power Side Channels

Andreas Kogler, Jonas Juffinger, and Lukas Giner, *Graz University of Technology;* Lukas Gerlach, *CISPA Helmholtz Center for Information Security;* Martin Schwarzl, *Graz University of Technology;* Michael Schwarz, *CISPA Helmholtz Center for Information Security;* Daniel Gruss and Stefan Mangard, *Graz University of Technology*

https://www.usenix.org/conference/usenixsecurity23/presentation/kogler

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

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: Collide+Power: Leaking Inaccessible Data with Software-based Power Side Channels

Andreas Kogler[1] Jonas Juffinger[1] Lukas Giner[1] Lukas Gerlach[2]
Martin Schwarzl[1] Michael Schwarz[2] Daniel Gruss[1] Stefan Mangard[1]

[1]Graz University of Technology [2]CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

We present Collide+Power, a technique that extends software-based power side channels to exploit the mere co-location of attacker-controlled data with victim data within CPU buffers, e.g., CPU caches. Collide+Power exploits that the *collision* of these values exposes the Hamming distance, *i.e.*, the bit difference between the values, in the power domain. Collide+Power can be mounted purely from software with any power-related signal, e.g., power consumption interfaces or throttling-induced timing variations.

The artifacts demonstrate the fundamental leakage enabling Collide+Power. First, we analyze the power leakage of the caches and evaluate our differential measurement method. Second, we compute the performance of the Correlation Power Analysis (CPA) for the raw channel and show that we leak precise victim data. Third, we analyze the effects of untargeted victim data within the cache lines. Finally, we demonstrate the attack PoCs for Collide+Power.

All the PoCs are tested on Intel, and some of the PoCs were also validated on AMD CPUs. Therefore, we only recommend Intel x86 CPUs to test the artifacts, for the best case, an *Intel Core i7-8700K*, *Intel Core i9-9980HK*, or *Intel Core i9-9900*.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifacts do not perform any destructive steps, and the worst risk for system security is a system freeze due to an unavailable Model Specific Register (MSR) read in the kernel module we provide. This *should* not happen if the system is set up as described above. We verified on our machines that the kernel module works as intended. If such a freeze occurs, data loss of unsaved files *could* happen. Therefore, we recommend saving all the work and doing a clean reboot before conducting the experiments. The PoCs only target the victim data of the provided programs. We do **NOT** target any other data on the system, nor do we read the personal files of the users. Furthermore, the power traces are saved locally and are not shared with the authors, nor does the provided framework send any information to us or any other server.

A.2.2 How to access

We provide the artifacts in a public GitHub repository. The most recent version of the artifacts is provided here: `https://github.com/iaik/collidepower` . The stable version of the artifacts with the included feedback from the artifact evaluation is provided here: `https://github.com/iaik/collidepower/tree/ae` .

A.2.3 Hardware dependencies

To reproduce the artifacts, we recommend a native *bare-metal* Intel CPU. We strongly recommend an *Intel Core i7-8700K*, *Intel Core i9-9980HK*, or *Intel Core i9-9900* CPU, as these CPUs showed the best leakage during our analysis (cf. Table 4 in the paper). For other CPUs not in the list, we designed the PoCs for an 8-way L1 cache and a 4-way L2 cache design with a pseudo-LRU replacement policy (cf. Section 4 in the paper) which can be checked with the `cpuid` command. If the cache uses a different number of ways, the PoCs need adaption, or the leakage *cannot* be guaranteed. Finally, we require an unfiltered Intel Running Average Power Limit (RAPL) energy measurement interface, meaning that for CPUs that support Intel Software Guard Extension (SGX), the Platypus patches might be active and obfuscate the energy measurements over the RAPL interface. Although we can exploit the throttling-induced timing variations with SGX enabled, we recommend disabling SGX in the bios to get *unmitigated* RAPL readings and significantly increase the practicality of the measurements.

A.2.4 Software dependencies

We require a Ubuntu 20.04 installation with Linux kernel version `5.4.` or `5.15`. The best case would be a fresh installa-

tion. The newer 5.19 kernel no longer supports the *nosmap* kernel argument, which is required for the initial leakage analysis. We detail this requirement in the provided readmes of the repository. Furthermore, we require access to the RAPL interface, which implies that the experiments must run on a bare metal machine and *should* not be a virtual machine as hypervisors block access to this interface. We require root privileges to insert a kernel module for the PoCs and to configure the Linux kernel boot command line. To build the PoC we require a built-essentials setup with gcc and make, which we list in the repositories readmes. Furthermore, we require the PTEditor to modify page tables. To post-process the recorded power traces, we use python3 with additional packages to provide installation steps in the readmes. Finally, the system should not be used during the measurements, *i.e.*, no other user must be logged in, and no program should be executed. We recommend using `ssh` to deploy and connect to the given machine.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

The artifacts use two components: First, a C++ program with a kernel module performs the experiments and records the power traces. Second, a post-processing python framework to analyze the recorded traces. Installation of the required packages is performed using the `apt install` command. The python packages are installed with `pip3`. Finally, we adapt some kernel boot parameters to make the analysis more straightforward. For the detailed apt and python3 packages, please follow the provided readmes in the repository.

A.3.2 Basic Test

We provide the basic leakage analysis in the repository, which evaluates if the system exposes the exploited leakage. This is the smallest possible *basic test* we implemented since we cannot identify if the leakage exists on the system with other means. For detailed instructions, please follow the provided readmes in the repository.

A.4 Evaluation workflow

A.4.1 Major Claims

We provide artifacts verifying the following claims:

(C1): Using attacker-controlled data and victim data within the memory hierarchy exposes the combined Hamming distance leakage of both values in the power domain. We prove this claim with the initial leakage analysis experiment (E1) described in Section 4, whose results are reported in Table 2.

(C2): Using the differential measurement technique improves the correlation coefficients and the factors for the Hamming distance. We prove this claim with the same data as the initial leakage analysis (E1) using a different post-processing technique (E2). The differential measurement technique is described in Section 5, and the results are reported in Table 3.

(C3): We evaluate the raw channel leakage rates using our CPA and demonstrate that we can leak single nibbles as described in Section 7.2, where the results are shown in Figure 9. We prove this claim in the raw channel evaluation (E3).

(C4): We show that unmasked data does not influence the CPA success probability due to the differential measurement. This claim is described in Section 7.2, and the results are shown in Figure 10. We prove this claim in the victim data fill experiment (E4).

(C5): We show that Collide+Power with MDS-Power leaks data that is actively used on the hyperthread. This claim is described in Sections 6.1 and 7.3, and the results are shown in Figure 12a. We prove this claim in MDS-Power experiment (E5).

(C6): We show that Collide+Power observes a signal with Meltdown-Power for data that is only accessible within the Linux kernel. This claim is described in Sections 6.2 and 7.5; the results are shown in Figure 12a. We prove this claim in the MDS-Power experiment (E6).

A.4.2 Experiments

(E1): [30 human-minutes + 10 compute-hour + <5GB disk]:
How to: Follow the general setup guide. Build the provided PoC with a defined macro. Let the PoC record the power traces. Use the provided post-processing script to obtain the results.
Preparation: Reboot the machine and connect via SSH to the test machine. Build the program and the kernel module. Load the kernel module, stop all other programs, and follow the overall system preparation.
Execution: Execute the c++ program and pipe the output into a CSV file. Let the script run for at least the specified compute hours. Please note that the compute hours are estimates as the program only records samples for analysis. The more samples are recorded, the more accurate the analysis will get.
Results: Run the provided analysis script on the CSV.

(E2): [30 human-minutes + 0 compute-hours + <5GB disk]:
How to: Reuse the data from E1, the data for E2 is already included in the csv of E1.
Preparation: The same steps as E1.
Execution: None

Results: The same steps as E1.

(E3): [10 human-minutes + 5 compute-hour + <5GB disk]:
How to: Follow the steps of E1 but with another macro.

(E4): [10 human-minutes + 5 compute-hour + <5GB disk]:
How to: Follow the steps of E1 but with another macro.

(E5): [10 human-minutes + 20 compute-hour + <5GB disk]:
How to: Follow the steps of E1 but with another macro.

(E6): [10 human-minutes + 50 compute-hour + <5GB disk]:
How to: Follow the steps of E1 but with another macro.

Important Notes: The execution times for the experiments are estimates based on our CPUs that show a high correlation for the used interface. The longer the experiments is run, the more data is collected, resulting in a more accurate analysis. Furthermore, the experiments are designed to be terminated (`CTRL+C`) after the desired amount of data is collected. Finally, the created CSV files should always be valid during the experiments, which allows them to be copied to a different machine and be analyzed without the experiment to be stopped.

A.5 Notes on Reusability

The framework to analyze the traces is a general framework that can be reused for plotting and performing correlation analysis beyond Collide+Power.

A.6 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/.