A Artifact Appendix

A.1 Abstract

This paper presents Half-Double, a new Rowhammer effect extending the reach of Rowhammer beyond the immediate neighbors. We show that this effect can not only circumvent current state-of-the-art mitigations like TRR, but defensive refreshes to *distance-1* rows also assist Half-Double. The general idea is to induce flips into a victim by combining many *distance-2* accesses with a few *distance-1* accesses.

In the artifact evaluation, we present experiments to underline the impact of Half-Double. Due to obligatory constraints, we cannot share parts of the initial root-cause analysis. Nevertheless, the artifacts presented show all the necessary steps to mount the Half-Double Attack on commodity systems protected by TRR and ECC.

We split the artifacts into the described challenges, which finally form the end-to-end exploit. First, the artifacts for Challenge **C1** “Memory Allocation” demonstrate three different ways to reconstruct contiguous memory. Second, for Challenge **C2** “Alternatives to Memory Templating”, we show both ECC-aware hammering and Blind-Hammering and provide the utility to count the overall bitflips on a device. Third, Challenge **C3** “Memory Preparation” shows the *Child Spray* technique to fill the memory with attackable data, i.e., page tables. Fourth, we provide the artifacts for **C4** “Robust Bit-Flip Verification”, namely the speculative oracle and the architectural `vfork` alternative. Finally, the Half-Double Attack built upon the previous parts to mount the end-to-end attack.

The end-to-end exploit is optimized for the chromeOS operating system and, more precisely, for our Chromebook setup. Nevertheless, all the components are compileable for both x86 and aarch64 architectures. We recommend ARM-v8 and Intel x86 CPUs for this artifact evaluation.

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

- **Program:** We provide the programs and represent how to install them.
- **Compilation:** We require gcc for cross-compilation. Download instructions are provided.
- **Run-time environment:** We require a native Linux installation for compilation. Some artifacts can be directly executed under Linux. For this purpose, we strongly recommend Ubuntu 20.04. For the end-to-end exploit, we require a chromeOS installation. The provided installation instructions need internet access.
- **Hardware:** We require either Intel x86 CPUs or ARM-v8 CPUs. Half-Double bitflips depend highly on the actual hardware and even differ between identical DRAM modules.
- **Execution:** For executing some benchmarks, we require a stable frequency.
- **Security, privacy, and ethical concerns:** Due to the Half-Double bitflip effect, **data corruption** can occur on the used system.
- **Metrics:** The benchmarks report nanosecond execution time, data size in bytes, and throughput in mega- or gigabytes per second.
- **Output:** The artifacts print the results to the terminal.
- **Experiments:** We include the source code, build scripts, and readmes describing the artifact and the process of how to execute the benchmarks.
- **How much disk space required (approximately)?:** Less than 1 GB.
- **How much time is needed to prepare workflow (approximately)?:** Below 4 hours.
- **How much time is needed to complete experiments (approximately)?:** Up to two days, depending on the hardware.
- **Publicly available (explicitly provide evolving version reference)?:** `https://github.com/iaik/halfdouble`
- **Code licenses (if publicly available)?: MIT**
- **Archived (explicitly provide DOI or stable reference)?:** `https://github.com/iaik/halfdouble/tree/ae`

A.3 Description

A.3.1 How to access

Check out the Git repository from `https://github.com/iaik/halfdouble` and follow the provided readmes.

A.3.2 Hardware dependencies

We recommend ARM-v8 CPUs with (LP)DDR4(x) DRAM supporting both TRR and ECC, like the Chromebooks in the paper. Most of the artifacts can also be executed on Intel x86 CPUs. Our experience showed that the susceptibility to Half-Double is highly dependent on the used DRAM modules.

A.3.3 Software dependencies

We strongly recommend Ubuntu 20.04 as a platform for compilation as we tested all the building steps there. The operating system to execute the artifacts should either be an Ubuntu or chromeOS operating system with root access for debugging. The components of the paper have to be built from the source. Hence the system requires tools for compiling software (`build-essentials` on Ubuntu). Finally, access to operating system interfaces as root is necessary for debugging, e.g., `/proc/self/pagemap` and `/dev/mem`.

A.3.4 Data sets

N/A

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

During our experiments with Half-Double, we observed **data corruption** in the operating system resulting in corrupted file systems. Therefore, we highly recommend a fresh installation with an operating system image not used for personal or important data. We *never* observed persistent damage on the hardware. However, we cannot ensure this is generally the case, but we find it highly unlikely to damage the used hardware.

A.4 Installation

Follow the readmes in the repository's top-level directory, which will guide you through installing all the necessary tools and components of the paper. The "Makefiles" *should* automate most of the process. However, we cannot rule out that some parts might need manual adjusting, and therefore, knowledge of C, C++, python3, bash, and Makefiles is beneficial.

A.5 Experiment workflow

Each artifact contains a readme, the source code, and a build script to build the source. After the binary is compiled, we can reuse the build script to *deploy* the binary to the test systems where the binary is executed. Note that some binaries require additional arguments passed via the terminal. The binary prints debug output to the terminal, and the results are also reported in this way.

A.6 Evaluation and expected results

The evaluation is split into multiple parts. First, we use the provided Half-Double hammering tool to verify the results from Table 1. The tool uses the *Quad pattern* to hammer and induce flips on commodity devices, e.g., the provided Chromebooks. The tool should report similar flip frequencies if performed on the provided hardware. Second, we execute the artifacts of Challenge **C1** to verify the general functionality and the performance numbers of Section 6.1 when detecting contiguous memory. Third, for Challenge **C2** we reuse the hammering tool with a slightly different configuration to demonstrate both Blind-Hammering and ECC-aware templating from Section 6.2. Fourth, Challenge **C3** uses an executable to demonstrate the *Child spray* of Section 6.3 to circumvent some ARM CPUs' reduced virtual address space and verify the performance numbers. Finally, the artifacts of Challenge **C4** scan memory and test the bitflip verification of Section 6.4 if a page table is corrupted.

A.7 Experiment customization

The artifacts use a timing side channel to find addresses belonging to the same DRAM bank. Therefore, the threshold of the timing side channel is configurable and usually passed via a command-line argument. We provide an additional utility to evaluate this threshold empirically. Nevertheless, this threshold might need manual adjustment. Finally, we can adjust the number of repetitions of a benchmark and the performed accesses in the hammer loop via compile-time parameters.

A.8 Notes

Rowhammer bitflips depend highly on the used DRAM, the device's battery state, and the environment. Similar to Table 1, identical commodity systems can behave differently. Therefore it is likely that results from the artifacts may differ.

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.