A Artifact Appendix

A.1 Abstract

Minefield is a probabilistic undervolting protection for SGX enclaves implemented via a compiler extension. The general idea is to place instructions highly susceptible to undervolting faults between regular instructions. In the artifact evaluation, we include all the tools needed to reproduce each result of the paper to follow the conclusion of our mitigation. First, we provide the instruction finding framework that automatically scans the x86 instruction set for instructions susceptible to undervolting faults. Second, we show a benchmark for the minimal time between voltage transitions. Third, we include the compiler infrastructure to automatically generate hardened enclaves and the required modifications to the SGX-SDK. Finally, we provide the tools to reproduce the performance, size, compile-time, and detection rate benchmarks of Minefield. Due to the nature of the paper, we require Intel hardware that supports SGX and a runtime environment where possible data corruption is *acceptable*. We recommend a clean installation of Ubuntu 20.04, with Intel CPUs between the 6^{th} and 10^{th} generation. Furthermore, if applicable, undervolting faults will lead to repeated system freezes during the profiling phase. Therefore, an automatic way to restart the system would be beneficial.

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

- **Program:** The used programs are provided, or how to install them is described.
- **Compilation:** We require a modified Clang 11 compiler. Download and build scripts are provided.
- **Transformations:** We provide the patches used to allow compilation of the SGX-SDK with Clang.
- **Data set:** We provide the framework to use the `https://uops.info` x86 instruction-set list.
- **Run-time environment:** Requires a native Linux installation that supports SGX, and we strongly recommend Ubuntu 20.04. The provided installation scripts require internet access.
- **Hardware:** Intel CPUs with SGX support between the 6^{th} and 10^{th} generation and MSR `0x150` available. Undervolting-based faults are highly dependent on the actual hardware and even differ between cores on the same CPU. We recommend one of the CPUs of the paper.
- **Execution:** For executing the benchmarks, we require a stable frequency, isolated cores, a modified grub command line, and software-based undervolting.
- **Security, privacy, and ethical concerns:** Due to the undervolting **data-corruption** can occur on the used system.
- **Metrics:** The benchmarks report performance in iterations per second, faulting points in mV, execution time in seconds, code size in bytes, and detection rate factors.
- **Output:** The resulting outputs are CSV files. We provide visualization scripts where possible.
- **Experiments:** We include installation scripts and readmes describing the process and how to execute the benchmarks.
- **How much disk space required (approximately)?:** 4-5 GB
- **How much time is needed to prepare workflow (approximately)?:** 3-4 hours
- **How much time is needed to complete experiments (approximately)?:** 1-5 days depending on the depth of the analysis.
- **Publicly available (explicitly provide evolving version reference)?:** `https://github.com/iaik/minefield`
- **Code licenses (if publicly available)?: MIT**
- **Archived (explicitly provide DOI or stable reference)?:** `https://github.com/iaik/minefield/tree/ae`

A.3 Description

A.3.1 How to access

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

A.3.2 Hardware dependencies

We require Intel CPUs which support SGX and have an available software undervolting interface (MSR `0x150`) available. We recommend CPUs between the 6^{th} and 10^{th} generation and recommend a desktop CPU shown in the paper. Our experience showed that the susceptibility to undervolting faults is highly dependent on the used hardware and even differs across cores from the same CPU. We recommend a system with physical access as undervolting faults will repeatedly crash the system and lead to system freezes.

A.3.3 Software dependencies

We strongly recommend Ubuntu 20.04 as it has official support for SGX, and we tested all the provided tools there. The components of the paper have to be built from source, hence the systems requires tools for compiling software (`build-essentials` on Ubuntu). Access to MSRs via the `msr-tools` interface is also necessary. Finally, we require a setup that allows frequency pinning via `cpupower` to fix the frequency at a given operating point during the undervolt.

A.3.4 Data sets

To speed up the finding of the susceptible instructions, we provide our found faultable instruction data set in the repository. Furthermore, we rely on the complete x86 instruction set list from `https://uops.info`, which is automatically used in the framework.

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

During our experiments with undervolting, we observed **data corruption** in recently used files. 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 used for undervolting. However, we cannot ensure that this is generally the case, but we find it highly unlikely to damage the used hardware.

A.4 Installation

Follow the readmes in the top-level directory, which will guide you through installing all the necessary tools and components of the paper. The installation scripts are written in bash and *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. Furthermore, due to the enormous complexity of SGX, some packages might need manual installation if not found correctly.

A.5 Experiment workflow

After building the components for the benchmarks, they can be executed via scripts for a given *placement density*. These scripts should be executed with a fixed frequency to allow a fair comparison between the runs. The benchmark results are exported in the CSV format, and we provide additional scripts to convert the measurements into relative overhead percentages with respect to the baseline.

A.6 Evaluation and expected results

The reproduced results from Table 1 and Table 2 should show that `imul` is, across multiple CPUs, the instruction most susceptible to faults. Some concrete instances might require extended instructions to detect the fault at the highest undervolting point correctly. With this assumption, the compiler extension can rely on `imul` as trap instruction.

For the performance results, we should see a nearly linear performance decrease (Figure 8) and a rising code size (Figure 10) when increasing the *placement density*. Some benchmarks are more affected by the *placement density* than others. For the mbedTLS (Figure 9) benchmark, some configurations with different key lengths and disabled redundancy checks in the library itself show better performance as the baseline depending on the number of leading zeros in the key. The compile-time (Figure 11) should also rise with increasing *placement density*. However, the absolute time increase should be minimal.

Finally, we provide test enclaves to test the detection rate of the mitigation (Figure 6) in the worst-case scenario and a more realistic scenario when protecting mbedTLS (Figure 7).

A.7 Experiment customization

Since the undervolting offset is highly dependent on the hardware and even the core executing the code, some benchmarks might need manual adjustment. The instruction finding framework automatically detects system freezes when using our remote system with a remote power switch. The overall runtime of the performance benchmark can be adapted via the number of runs.

A.8 Notes

Undervolting faults are highly dependent on the used systems. Even our two identical systems from Table 2 show different faulting behavior. Furthermore, we observed different undervolting offsets on cores of the same CPU. Therefore it is likely that the undervolting-related results from the artifacts differ.

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.