Repurposing Segmentation as a Practical LVI-NULL Mitigation in SGX

Lukas Giner, Andreas Kogler, and Claudio Canella, *Graz University of Technology;* Michael Schwarz, *CISPA Helmholtz Center for Information Security;* Daniel Gruss, *Graz University of Technology*

https://www.usenix.org/conference/usenixsecurity22/presentation/giner

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

The public repository[1] contains all the code necessary to reproduce the data for all performance graphs/tables in the paper, as well as PoCs to demonstrate that the mitigation works. This includes patches and build instructions for LLVM11, the Intel SGX SDK and PSW, as well as the benchmarks. The artifact requires SGX to evaluate, and is easiest to run on Ubuntu 18.04 or 20.04.

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

- **Program:** Adapted versions of `nbench` and `sgxbench` are downloaded & installed via included scripts.
- **Compilation:** Requires a modified Clang 11, install & download script is included.
- **Transformations:** A tool to fix up relocations is included (relocator).
- **Run-time environment:** Needs a native Linux installation that supports SGX, Ubuntu 18.04 or 20.04 are strongly recommended. Build scripts need internet access at several points. Requires root for installation and evaluation. PoCs require the PTEditor kernel module.
- **Hardware:** Intel CPU with SGX support, needs to be vulnerable to LVI-Null for PoC tests (affected CPUs).

  The PoCs need a kernel module, which means either self-signing or disabling secure boot. This may require physical access to the machine.
- **Run-time state:** As this artifact includes performance benchmarks, a stable CPU frequency and isolated cores are recommended.
- **Execution:** For ideal testing, the system should have isolated cores, fixed frequency, and not much other activity.
- **Metrics:** Benchmarks report cycle count or iterations/s, PoCs report leakage percentage.
- **Output:** Benchmark outputs are .csv tables with performance, an included spreadsheet can convert to a graph similar to the paper.
- **Experiments:** Installation scripts are included and described here and in READMEs.
- **How much disk space required (approximately)?:** 4-5GB
- **How much time is needed to prepare workflow (approximately)?:** 2-3h
- **How much time is needed to complete experiments (approximately)?:** 3-6h, depends on hardware
- **Publicly available?:** `https://github.com/IAIK/LVI-NULLify`
- **Code licenses (if publicly available)?:** zlib

[1] `https://github.com/IAIK/LVI-NULLify`

A.3 Description

A.3.1 How to access

Clone `https://github.com/IAIK/LVI-NULLify/tree/ae_final` and follow the README.md from there.

A.3.2 Hardware dependencies

As this is a mitigation for Intel SGX, SGX support is a hard requirement. To fully evaluate the PoCs, and not just mitigation performance, the CPU also needs to be vulnerable to LVI. You can check if your CPU is vulnerable here: `https://software.intel.com/content/www/us/en/develop/topics/software-security-guidance/processors-affected-consolidated-product-cpu-model.html`

A.3.3 Software dependencies

We strongly recommend Ubuntu 18.04 or 20.04 as these are officialy supported by Intel, and all our tools were tested on them.

Beyond standard compilation tools (ninja, cmake etc) our PoCs require the PTEditor kernel module [2]. Other requirements are listed in the README files at the appropriate points.

A.4 Installation

Follow the detailed README in the top-level directory to set up our modified clang compiler and relocator and install the SGX driver as well as our modified SGX SDK and PSW.

Once that is done, you can already test your installation with the PoCs by following the README file in the POC directory.

With a working PSW and driver, you can follow the README in the benchmarks directory to download and build the benchmarks.

A.5 Experiment workflow

After building the benchmarks, follow along in the README to start all or a subset of them. An important aspect to keeping benchmarks comparable is to fix the CPU's frequency to a sustainable level, and idealy run them on an isolated core.

PoCs can be run according to the README in the POC folder.

A.6 Evaluation and expected results

The main results in our paper are contained in Figure 4/Table 3. These are the performance overheads of our LVI-NUll mitigation compared to other, similar mitigations. The second, more implicit result is the efficacy of LVI-NULLify.

For the benchmarks, the absolute performance overheads vary significantly between different machines and architectures (compare Figure 4 and Figure 5), but the relative differences should be roughly similar. That is: LVI-Nullify should be the fastest mitigation, or at least very close to Intel CFI, typically followed, with some distance, by Intel's optimized-cut mitigation.

For the PoCs, starting once without and once with mitigation should produce qualitatively similar results to the examples shown in the README. That means, for the 3 PoCs where LVI-Nullify is

[2] `https://github.com/misc0110/PTEditor/`

effective, leakage rate should drop to zero, or a level that is comparable to the noise-catching output "other". While absolute leakage rates before applying the mitigation may differ significantly from system to system, they should be clearly differentiable from "other".

The respective READMEs for benchmarks and PoCs detail how to reproduce these results.

A.7 Experiment customization

Attack PoCs need a cache miss threshold, which is automatically determined. If this doesn't work, it can be set manually in the corresponding *App.cpp* file. All PoCs include a conf.h file, in which the character that should be leaked can be changed if desired.

Both benchmark run-scripts contain a variable called "isolated_core" that sets the core on which they should be run on. Set this to an isolated core, if available.

sgx-nbench contains a parameter to change the number of iterations in the file, see the benchmarking README.