

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

An Integer Overflow Endgame: Compiler and Architecture Support for Default-On Integer Overflow Mitigation

Zheng Zhang, Qi Ling, and Kian Kasad, *Purdue University;*
Brendan Ryan Sweeney, *UT Austin;* Deepak Gupta, *Meta Platforms Inc.;*
Tal Garfinkel, *Google;* Kazem Taram, *Purdue University*

https://www.usenix.org/conference/usenixsecurity26/presentation/zhang-zheng

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: An Integer Overflow Endgame: Compiler and Architecture Support for Default-On Integer Overflow Mitigation

Zheng Zhang
Purdue University

Qi Ling
Purdue University

Kian Kasad
Purdue University

Brendan Ryan Sweeney
UT Austin

Deepak Gupta
Meta Platforms Inc.

Tal Garfinkel
Google

Kazem Taram
Purdue University

A Artifact Appendix

A.1 Abstract

To support the replicability of our results and foster future research into integer overflow mitigation, we have made our complete artifact suite available. It contains the following components: modified LLVM and Rust compiler toolchains with opX support, all benchmarks used to evaluate performance (except SPEC CPU 2017), and scripts to build and orchestrate the evaluation. In addition, a modified version of gem5 is also included, although it is only used for functionality tests and not for performance evaluation.

The artifacts also come with a top-level and Rust-specific `README.md` to explain the setup details in case the automatic script runs into issues.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Except for installing dependencies, the software does not send or receive data over the Internet. All computations are performed locally. The code does not execute any destructive steps. For performance stability, address space layout randomization (ASLR) is temporarily disabled during the evaluation and restored afterwards.

A.2.2 How to access

The artifacts are available on Zenodo[1]. They are also available as a public CloudLab profile[2].

A.2.3 Hardware dependencies

A **bare-metal** x86 machine with at least 32 GB of RAM and 150 GB of free disk space is required to evaluate the artifacts. The CPU must support AVX instructions. The evaluation takes approximately 2 days to complete on a server with a 28-core Intel(R) Xeon(R) Gold 5512U CPU on CloudLab.

Note that evaluating on virtual machines (VMs) might result in severe result noises, so it is highly discouraged. If you have to run on VMs, make sure to run `/opt/overflow/scripts/bench.sh` on host to disable SMT and fix frequency.

In addition, compiling the LLVM and Rust toolchains (the build script uses `-j16` by default) can consume significant memory; if the machine has low RAM, ensure that sufficient swap space is configured to prevent Out-Of-Memory (OOM) compiler crashes.

A.2.4 Software dependencies

The installation script assumes a Debian-based Linux distribution, but can be easily adapted to other distributions. The script installs all dependencies. It has been tested on Ubuntu 22.04 LTS.

A.2.5 Benchmarks

We use the following benchmarks: SPEC CPU 2017, LFI-Bench, SQLite, RocksDB, Redis, NPB-Rust, and Hashbrown. All benchmarks are included in the artifacts except SPEC CPU 2017, which we cannot redistribute due to licensing. Please obtain it separately (version 1.1.9 has been tested for the evaluation).

To install SPEC CPU 2017, you can either place the installer ISO image (`cpu2017-1.1.9.iso`) under `/opt` and run `/opt/overflow/scripts/install_spec.sh`, or manually install SPEC CPU 2017 directly into `/opt/overflow/spec2017`.

[1] https://doi.org/10.5281/zenodo.20317245
[2] https://www.cloudlab.us/p/int-overflow-hw/int-overflow-ae

A.3 Set-up

A.3.1 Installation

If a CloudLab account is available, using the CloudLab profile is more convenient. Once the profile is instantiated as an experiment, everything is set up on the machine. If the default `c6620` node is unavailable, other x86 nodes can also be used.

Otherwise, the artifacts can be installed manually from the Zenodo archive:

```
# requires root
sudo tar xfz overflow.tar.gz -C /opt
sudo chown -R $USER /opt/overflow
```

Regardless of the previous step, install dependencies and build toolchains (estimated 30 minutes):

```
cd /opt/overflow
./install.sh
```

Verify that the installation completed successfully:

```
source "$HOME/.cargo/env"
rustc --version
# expected: rustc 1.86.0-dev
/opt/overflow/llvm/bin/clang -v
# expected: clang version 19.1.0
```

A.3.2 Basic Test

Perform a basic test on the built toolchains to verify their functionality:

```
cd /opt/overflow/test
./basic_test.sh
# expected: Basic Functionality Test Complete!
```

A.4 Evaluation Workflow

A.4.1 Major Claims

(C1): In SPEC CPU 2017, opX achieves near-zero-cost overflow checks (0.3% without SIMD and 1.9% with SIMD checks) compared to the 14.9% overhead of UBSan. Software checks with LCI achieve 4.4% (11.7% with SIMD checks) overhead, which is significantly lower than UBSan. The results are presented in Figures 8 and 9.

(C2): In popular libraries and CoreMark, opX achieves a 4% overall overhead compared to UBSan's 41.7%. The software-only LCI version also achieves a 19.6% overhead. The results are shown in Figure 12.

(C3): In the Hashbrown benchmark and NPB-Rust, opX achieves near-zero overhead compared to the 3.5% overhead of Rust's built-in overflow checks. The software-only LCI version also achieves near-zero overhead. The results are shown in Figure 13.

A.4.2 Experiments

(E1): *[Benchmarks] [5 human-minutes + 48 compute-hours + 150 GB disk space]: Run all benchmarks to validate our major claims.*

Preparation: Ensure that the installation is complete and the basic test passes.

Execution: Run the following command in tmux to start the orchestration script:

```
cd /opt/overflow
bash reproduce_all.sh > reproduce.log 2>&1
```

`SKIP_SPEC`, `SKIP_LFI`, `SKIP_DB`, and `SKIP_RUST` can be set to 1 to skip SPEC CPU 2017, popular libraries and CoreMark, databases (SQLite, RocksDB, and Redis), and Rust benchmarks if needed.

Results: The figures are generated under `/opt/overflow/results` as PDF files:

- `fig8.pdf`: SPEC CPU 2017 performance with opX
- `fig9.pdf`: SPEC CPU 2017 performance with LCI
- `fig12.pdf`: Popular libraries and CoreMark performance
- `fig13.pdf`: Hashbrown and NPB-Rust performance
- `fig16.pdf`: Database performance (optional)

Compare these figures with the results in the paper.

A.5 Notes on Reusability

Our LLVM and Rust compiler toolchains are currently experimental. They can be used to compile other C/C++ and Rust programs, but may encounter some back-end lowering errors. However, the compilers remain useful even without hardware-accelerated integer overflow checking. For instance, we show that software-only LCI can still outperform standard LLVM UBSan. By default, overflows are logged to a file instead of being printed to stdout. This behavior can be changed in `ubsan_handlers.cpp`.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.