MetaSafe: Compiling for Protecting Smart Pointer Metadata to Ensure Safe Rust Integrity

Martin Kayondo and Inyoung Bang, *Seoul National University;* Yeongjun Kwak and Hyungon Moon, *UNIST;* Yunheung Paek, *Seoul National University*

https://www.usenix.org/conference/usenixsecurity24/presentation/kayondo

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

USENIX Security '24 Artifact Appendix: METASAFE: Compiling for Protecting Smart Pointer Metadata to Ensure Safe Rust Integrity

M. Kayondo
Seoul National University

I. Bang
Seoul National University

Y. Kwak
UNIST

H. Moon
UNIST

Y. Paek
Seoul National University

A Artifact Appendix

A.1 Abstract

This artifact provides an outline of steps to reproduce the METASAFE + TRust benchmarks of the above mentioned paper. We provide notes on system requirements, and how to navigate possible challenges for those interested in building the project from scratch.

A.2 Description & Requirements

METASAFE and TRust depend on Intel MPK, therefore, the user of the artifact must build and execute the benchmarks on an Intel Machine. As explained in the paper, we expect the user to use a workstation running Ubuntu Jammy 22.04.2 LTS, with an Intel 11th Gen CPU with at least 8 cores and 72GB of RAM because TRust depends on SVF which requires a substantial amount of memory at compile time. If the user is interested in running the Docker image, we advise to run on a machine with a minimum of 64GB disk space and 64GB RAM.

A.2.1 Security, privacy, and ethical concerns

The artifact requires you install dependencies whose versions may conflict with already existing versions on your system. The PoCs provided have baselines with possible Buffer Overflow bugs that METASAFE is meant to prevent. Whereas such buffer overflows are benign for the purposes of this test, we would like to inform the users intending to execute the PoCs of this fact. The PoCs are only meant for proving METASAFE's security guarantees.

A.2.2 How to access

A user interested in the artifact can find it at our github repository: `https://github.com/seccompgeek/trust23-metsafe24/tree/v0.1.0` *. At the repository, we provide instructions on how to obtain a Docker images of an already setup built environment.*

A.2.3 Hardware dependencies

For a user interested in building the artifact from scratch, we recommend running a workstation with at least 72GB RAM, at least 64GB free disk space, and an Intel CPU with MPK and at least 8 CPU cores. For a user interested in simply running the benchmarks in from the Docker images, we recommend using a machine with at least 64GB disck space and at least 64GB RAM. If RAM is an issue, the user may modify the SVF building/execution commands to lessen the analysis budget.

A.2.4 Software dependencies

The required dependencies are well explained on the github repository. We expect the user to run an Ubuntu OS of version no later than 18.04 LTS. For our experiments, however, we use Ubuntu 22.04 LTS.

A.2.5 Benchmarks

For this artifact, we present the user with the METASAFE + TRust benchmarks shown in Section 6.4 of the paper. On the Github repository, we explain how to run each benchmark.

A.3 Set-up

Setup instructions are provided on the github repository.

A.3.1 Installation

Installation instructions are provided on the github repository.

A.3.2 Basic Test

Run the Simple PoC as given in the Github repository. You may also run the SmallVec PoC to test METASAFE's insertion of metadata checks.

A.4 Evaluation workflow

The evaluation involves running the baseline (without TRust or METASAFE). Then another run with TRust is required to

obtain the TRust overhead compared to the baseline. Finally, another run with TRust + METASAFE is required to obtain the overhead of METASAFE on TRust. The benchmarks are based on Rust's Bencher. At the end of each run, averaging the runtimes is required. We provide a simple tool that will anlalyse the results of each run to provide the average runtime. For memory overhead, we read the Maximum Resident Size provided by /usr/bin/time with the -v option, as explained in the Github repository.

A.4.1 Major Claims

Trust incurs performance and memory overhead compared with the baseline. To that end, we expect METASAFE to incur added overhead on top of TRust.

(C1): *We expect TRust + METASAFE to incur a runtime overhead of 13% geomean over the baseline, compared to Trust's 11%. Therefore, we expect METASAFE to incur about 3% added overhead to TRust or any similar system.*

(C2): *Additionally, TRust + METASAFE incurs about 83% memory overhead (geomean) compared to TRust's 69%.*

(C3): *We expect METASAFE to prevent or catch the memory bugs related to smart pointers as shown in the PoCs provided. In absence of METASAFE, we expect the memory bugs to cause issues such as Segmentation Faults or allocator-side errors. With METASAFE, we expect the bugs to be caught at runtime before they cause any errors.*

A.4.2 Experiments

For a user intending to build the artifact from scratch, we expect more man hours. For a user intending to simply run experiments from the Docker image, we expect less man hours, mostly spent on running experiments and analyzing them.

(E1): *[Building the artifact from scratch] [30 human-minutes + 2 compute-hour + 35GB disk]:*
How to: *Access the Github repository and follow the instructions. Install dependencies as instructed.*
Preparation: *Install the dependencies and clone the artifact from Github.*
Execution: *Build the rust compiler and create a custom toolchain. Build SVF.*
Results: *Should a custom rust compiler installed, and SVF setup, with its LLVM linked to the rust compiler's version.*

(E2): *[Running and analyzing the Benchmarks:] [2 human-hour + 5 compute-hour]: Mostly due to SVF analysis. Some benchmarks don't take that long. Only Hyper and Tokio take long. At the end, the user should have results to compare with the claims in the paper.*

A.5 Notes on Reusability

The benchmarks provided for Trust + METASAFE are highly reusable. Especially the Docker image version with the already built environment. Reproducing and reducing the Servo results is particularly challenging due to the huge number of dependencies (each with a differing version), and that's the major reason we do not present it here. Please run each benchmark, saving the output in a file. Then run the *results analyzer* mentioned on the `Github` README with the output file as the parameter. The *results analyzer* prints the average runtime. After gathering the average runtime for the baseline, TRust and TRust + METASAFE, you can obtain the normalized overhead.

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/usenixsec2024/.