Fuzztruction: Using Fault Injection-based Fuzzing to Leverage Implicit Domain Knowledge

Nils Bars, Moritz Schloegel, Tobias Scharnowski, and Nico Schiller, *Ruhr-Universität Bochum;* Thorsten Holz, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity23/presentation/bars

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix
Fuzztruction: Using Fault Injection-based Fuzzing to Leverage Implicit Domain Knowledge

Nils Bars*, Moritz Schloegel*, Tobias Scharnowski*
Nico Schiller*, Thorsten Holz‡

*Ruhr-Universität Bochum
‡CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

FUZZTRUCTION's artifact contains the source code necessary to run our fuzzer (as well as competing fuzzers). This document describes how to set-up our fuzzer prototype, gives a brief overview of the resource requirements to replicate the experiment (i. e., a coverage comparison with other state-of-the-art fuzzers) conducted in our paper, and contains instructions for reproducing our results.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact is shipped via a Docker image used to spawn a unified containerized environment to ease evaluation and development independent of the underlying system. The container's life cycle is managed by the `env/start.sh` script which by default forwards the `ssh-agent` (if any) and the `.gitconfig` into the container. If this is undesired behavior, the related functionality should be removed from the script before conducting any experiments.

A.2.2 How to access

The artifact's source code is accessible at https://github.com/fuzztruction/fuzztruction/tree/91ba684d2b8fa21ae19e403496b507f3729c4ff5. The repository and sub repositories contain extensive documentation to make the artifact evaluation process as easy as possible.

A.2.3 Hardware dependencies

For evaluation, we used two Intel(R) Xeon(R) Gold 6230R CPUs, totaling 52 cores, 128 GB RAM, and about 1 TB SSD disk space. Since some targets produce many test cases, which are stored in `/tmp`, i. e., in RAM, we advise resizing the `tmp` folder to 600 GiB and backing the amount exceeding the RAM capacity via a swap file. The evaluation script will walk you through these steps.

In our paper, we evaluated 12 targets, which we run five times for 24 hours, and assigned all 52 cores to one experiment. Consequently, a vast amount of computational power is required to replicate the exact experiments conducted in our paper. We believe this computational power is out-of-scope for artifact review, thus we provide instructions on how to approximate our experiments using significantly less resources in Section A.4.2.

A.2.4 Software dependencies

For running the artifact, a working Docker installation is required. All scripts that must be executed on the host system (i. e., outside of the container) have been tested exclusively on Ubuntu 22.04. However, since the scripts are rather simple, they should work on any Linux distribution.

A.2.5 Benchmarks

All data required for the evaluation is part of the linked repository.

A.3 Set-up

The set-up is explained in detail in the main repository's `README.md`. For your convenience, we provide pre-built versions of FUZZTRUCTION. We recommend using these pre-built versions of FUZZTRUCTION since slight changes in, for example, libraries linked into a fuzzing target might cause deviations from the results presented in the paper. Furthermore, compiling all targets takes considerable time, because we use AFL++'s collision free encoding, which causes link time to increase significantly. Overall, the compilation of all targets takes multiple hours.

A.3.1 Basic Test

Testing the set-up is possible using the steps provided in the *Fuzzing a Target using Fuzztruction* section in the `README.md`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): We demonstrate the generic capabilities of our approach by fuzzing even complex cryptographic procedures, such as the parsing and validation of encrypted RSA keys, automatically and without custom-crafted seeds.

(C2): We implement and evaluate our prototype, called FUZZTRUCTION, against the state-of-the-art fuzzers AFL++, SYMCC, and WEIZZ. Our results show that our approach achieves significant gains in terms of coverage and number of software faults found.

Please note that claim C1 is a subset of C2, since outperforming all other competitors on a cryptographic target also indirectly shows our approach's applicability to complex cryptographic targets.

A.4.2 Experiments

As described in the requirements section, we conducted an extensive evaluation requiring considerable CPU time for reproduction. Since comparing each individual result from the paper with an experiment using fewer resources is impossible, we suggest concentrating the evaluation efforts on a subset of the fuzzing targets.

According to our statistical analysis presented in Table 2 in our paper, the three targets `objdump`, `readelf` and `unzip` show no statically significant difference between the evaluated fuzzer configurations. Thus, we advise excluding these targets from the reproduction since they are no proxy for our claims. For the remaining targets, we advise only considering the best competitor (cf. Table 2) to further reduce the amount of CPU time required.

Following our recommendations, the artifact evaluation effort is composed of 52 CPUs * 9 targets * 2 fuzzers (FUZZTRUCTION, best competitor) * 24h, which equates to running a single 52 CPU machine for 18 days. If desired, the list of targets might be further reduced by skipping targets not employing cryptographic primitives (i. e.,, targets in Table 2 that are not marked with a lock) since targets using cryptographic primitives are required to support both our claims. For example, if fuzzing only 3 targets (e. g., `rsa`, `vfychain`, and `7zip-enc`) with 2 fuzzers (FUZZTRUCTION, best competitor) for 24 hours, your fuzzing run will take 6 days on a single 52 CPU machine.

Notably, since fuzzing is inherently non-deterministic, a single run per target does not necessarily exactly align with the results presented in the paper. Consequently, reducing the number of tested targets in favor of doing multiple runs for some targets might be desirable. Certainly, this trade-off is primarily driven by the available hardware resources.

As a result of this experiment, we expect a plot similar to Figure 3 in our paper. All steps required to run the experiment, and to plot the data, are explained in the documentation found in the artifact's git repository. Please mind that some of the targets are not supported by SYMCC.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.