Loki: Hardening Code Obfuscation Against Automated Attacks

Moritz Schloegel, Tim Blazytko, Moritz Contag, Cornelius Aschermann, and Julius Basler, *Ruhr-Universität Bochum;* Thorsten Holz, *CISPA Helmholtz Center for Information Security;* Ali Abbasi, *Ruhr-Universität Bochum*

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

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

"Loki: Hardening Code Obfuscation Against Automated Attacks" is a paper on code obfuscation that focuses on hardening VM handlers with the goal of thwarting automated attacks such as symbolic execution.

Our artifact includes both the source code of our prototype, which allows to create obfuscated binaries, and the attack tooling used in the evaluation (on Github) as well as the data generated during the evaluation (published as dataset on Zenodo). Our experiments cover various aspects from measuring Loki's overhead to measuring its resilience w.r.t. to automated attacks. All experiments come with a README.md explaining the individual scripts and a wrapper script to improve usability. We use a Docker container to minimize setup problems and make the artifact accessible to different setups.

Evaluating this artifact will require

1. Building a docker container (and potentially downloading up to 50GB of data from an accompanying Zenodo artifact)
2. Creating a number of obfuscated binaries (we provide convenience wrapper scripts doing all the work)
3. Running 14 experiments (most of which have multiple steps):
   - Validating correctness and measuring overhead
   - Running multiple attacks

Individual experiments may run multiple hours (depending on whether (1) you want to use binaries created by us or create them yourself and whether you focus on replicating the results on a subset or intend to test all binaries).

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

- **Data set: `https://zenodo.org/record/6686932`**
- **Hardware: 52 cores + 64GB RAM + 25GB disk space**
- **Experiments: 14 different ones, covering all aspects**
- **How much disk space required (approximately)?: 25GB**
- **How much time is needed to prepare workflow (approximately)?: 1h**
- **How much time is needed to complete experiments (approximately)?: 40h (experiments can run unattended after being launched)**
- **Publicly available (explicitly provide evolving version reference)?: Yes**
- **Code licenses (if publicly available)?: AGPL 3**
- **Data licenses (if publicly available)?: AGPL 3**
- **Archived (explicitly provide DOI or stable reference)?: 10.5281/zenodo.6686932**

A.3 Description

Our artifact is split into two parts: The core component is the source code of our prototype and evaluation tooling (published on Github). Beyond that, we published artifacts such as produced binaries and raw results in a dataset on Zenodo. In essence, our artifact includes 14 experiments, which create obfuscated binaries, evaluate their overhead/correctness, or attack them using a number of automated simplification attacks.

Our code is intended to be run in a (provided) Docker container.

A.3.1 How to access

- Download the source code from Github: `https://github.com/RUB-SysSec/loki/commit/86134c1318347547debaf9b77e867d5b16d79d1d`
- Download the dataset from Zenodo: `https://zenodo.org/record/6686932`

A.3.2 Hardware dependencies

We recommend a server with many CPU cores (to reduce the experiment runtime and speed-up evaluation); We recommend more than 52 cores, at least 64 GB RAM, and about 25 GB disk space. These are no hard requirements: Less cores may work, but will increase the runtime of all experiments. Internet access is recommended.

A.3.3 Software dependencies

We provide our code in form of a Docker image. We have not tested the artifact on any OS other than Linux; most distributions should work fine. Optimally, your kernel supports Kernel Samepage Merging (https://www.kernel.org/doc/html/latest/admin-guide/mm/ksm.html).

A.3.4 Data sets

There is a data set containing evaluation results, binaries, and other artifacts resulting from our evaluation on Zenodo at `https://zenodo.org/record/6686932`. It is 4.9GB when zipped and 16GB when unzipped on disk.

A.4 Installation

1. Download the source code from Github:
   `git clone https://github.com/RUB-SysSec/loki.git`
2. Use our wrapper script to build the Docker container:
   `cd loki && ./docker_build.sh`
3. Start the docker container using our wrapper script:
   `./docker_run.sh`

4. Running this script again will connect you to the container:
   `./docker_run.sh`

5. Within the docker container, install Loki and all dependencies:
   `./setup.sh`

Noteworthy, `docker_run.sh` will mount the loki directory within the container as volume: Simplified speaking, anything within the container that is located within `/home/user/loki/` will be available outside the docker container in the `loki` folder. This can be convenient, e. g., if you want to copy the dataset from Zenodo into the Docker container: Simply place it in the `loki/` folder and it will be accessible from within the container. A more detailed explanation can be found in the Github README.md.

A.5 Experiment workflow

All experiments are located in `loki/experiments/` with the experiment number matching the one in the paper. Each experiment is documented in a `README.md` and usually consists of two to four steps. For your convenience, we provide Python scripts automating that part (`experiment_N.py`). As some experiments can share data (such as binaries generated), we recommend you do not change the default paths suggested (data will almost always be placed in `/home/user/evaluation`). Experiments can be customized by setting command line flags, changing values of globals in the Python wrapper script, or patching the scripts themselves (which we don't recommend generally). Our experiments usually run at least hours up to days (the standard timeout used is always 1 hour for each task; oftentimes there are at least $1,000$ tasks per experiment. We suggest you test the experiment on a subset of tasks (e.g., by generating only 10 binaries instead of $1,000$). **All experiment scripts already propose a more sensible value of tasks.** If this is not desired, changing `NUM_INSTANCES` (or similar-named constants at top of the scripts) allows you fine-granular control of how much tasks are executed.

A.6 Evaluation and expected results

Our experiments cover all relevant aspects. A detailed approach on how to reproduce them can be found in the `README.md` files we provide for each experiment. The expectations of the experiment are outlined in the paper.

- Dead code elimination: Only a few instructions (1-2%) of Loki's handler can be removed
- Experiment 1 Correctness: The obfuscated binaries produced by Loki maintain the same functionality
- Experiment 2 Coverage: Full code and path coverage is achieved.
- Experiment 3 Overhead: The obfuscated binaries produced by Loki have an overhead factor of 300 to 500 (runtime) and 20 to 50 (size)
- Experiment 04 Key Encodings: The SMT solver cannot solve the Factorization-based key encoding and 70% of the point functions
- Experiment 05 Key Encodings on Binary Level: The SMT solver finds a correct key in 31% of the cases
- Experiment 06 Taint Analysis: Taint analysis taints all but 17% of the instructions
- Experiment 07 Backward Slicing: Backward slicing slices all but 5% to 8% of the instructions
- Experiment 08 Symbolic Execution: SE simplifies no handler (static scenario) or 18% (dynamic attacker; depth 3) / 15% (dynamic attacker; depth 5)
- Experiment 09 MBA Diversity: Loki uses $5,482/7,000 = 78\%$ unique MBAs
- Experiment 10 MBA Formula Deobfuscation: LokiAttack significantly outperforms MBA Blast (the best competitor); resuls should be similar to Figure 3
- Experiment 11 Complexity of Core Semantics: Using superoperators increases the number of core semantics from 16 to 59; with superoperators, the semantic depth ranges from 5 to 13 with a peak at depth 9
- Experiment 12 Limits of Program Synthesis: Synthesis falls of w.r.t. synthesizing expressions of higher semantic depth; shape should be similar to Figure 5
- Experiment 13 Superoperators on the binary level: Syntia manages to synthesize about 19% of Loki's expressions

Due to non-determinism (in both our obfuscator and analysis tooling) and the scope of this artifact evaluation (evaluating 10 binaries instead of $1,000$), we expect quite some fluctuations. In some cases, it may be necessary to evaluate 100 binaries instead of 10 to reproduce our results.

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20220119.