USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

From Alarms to Real Bugs: Multi-target Multi-step Directed Greybox Fuzzing for Static Analysis Result Verification

Andrew Bao, *University of Minnesota, Twin Cities;* Wenjia Zhao, *Xi'an Jiaotong University;* Yanhao Wang, *Independent Researcher;* Yueqiang Cheng, *MediaTek;* Stephen McCamant and Pen-Chung Yew, *University of Minnesota, Twin Cities*

https://www.usenix.org/conference/usenixsecurity25/presentation/bao-andrew

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: From Alarms to Real Bugs: Multi-target Multi-step Directed Greybox Fuzzing for Static Analysis Result Verification

A Artifact Appendix

A.1 Abstract

*Our artifact, **Lyso**, is a multi-target, multi-step directed greybox fuzzing (DGF) designed to verify static analysis results by leveraging semantic program flow information and alarm correlations. Lyso introduces novel seed selection, power scheduling, and step-tracking mechanisms to efficiently explore and confirm vulnerabilities flagged by static analysis tools. In our evaluation, Lyso achieved a 12.17× speedup over state-of-the-art fuzzers while confirming the highest number of true bugs and discovering eighteen new vulnerabilities.*

This artifact includes:

- ***Source Code:** Contains the implementation of Lyso, which is built on AFL v2.57b and uses LLVM 11.0.0.*
- ***Sequences of steps:** Sequences of steps derived from CodeQL, which guide the fuzzing process.*
- ***Evaluation Scripts:** Scripts to reproduce the experimental results on the Magma benchmark.*

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

- ***Execution Risks:** Lyso executes real-world applications with dynamically generated inputs. It is advised to run the artifact in an isolated virtual machine (VM) or a controlled testing environment to prevent potential security issues.*
- ***Crash-Induced Data Corruption:** As Lyso is designed to trigger crashes and anomalous behaviors, there is a risk of file corruption or unintended system behavior. Evaluators should ensure that no critical system files or sensitive data are accessible during execution.*
- ***Data Handling:** Lyso does not interact with personal or confidential data. However, test cases generated during fuzzing should be managed carefully to avoid exposing any unintended sensitive information.*
- ***Vulnerability Disclosure:** If evaluators identify new vulnerabilities using Lyso, they should follow responsible disclosure practices when reporting bugs to software maintainers.*

A.2.2 How to access

*Our artifact, **Lyso**, is publicly available and archived to ensure long-term accessibility. The artifact can be accessed at the following stable Zenodo reference: https://zenodo.org/records/14714504. Additionally, we will maintain a GitHub repository to facilitate continued development.*

A.2.3 Hardware dependencies

Lyso does not require any specialized hardware. It runs on general-purpose x86-64 architectures and has been evaluated on systems with Intel processors. There are no strict hardware dependencies such as GPUs, FPGAs, or specialized interconnects.

A.2.4 Software dependencies

Since all required software (e.g., compilers, fuzzing frameworks, Python packages) is handled through Lyso's Magma setup scripts inside the Docker container, evaluators do not need to install any additional dependencies manually and only need to ensure that they can run Magma on their system. To successfully launch Magma, we recommend running it on Ubuntu 22.04. Other Linux distributions may work, but they are not officially tested.

A.2.5 Benchmarks

*Lyso requires the **Magma** benchmark as the primary dataset for evaluating its fuzzing effectiveness. The Magma benchmark provides a diverse set of real-world programs containing known bugs, which serve as targets for Lyso's experiments. Buggy programs from Magma are included in the provided Docker environment and do not need to be manually downloaded or configured by the evaluator.*

A.3 Set-up

A.3.1 Installation

The installation of Lyso and its dependencies is fully automated using the Magma benchmark's Docker-based setup. Follow these steps to install and set up the artifact.

Step 1: Install Required Dependencies *Before using Magma and its scripts, install the necessary system dependencies:*

```
apt-get update && apt-get install -y \
  util-linux inotify-tools docker.io git
```

Step 2: Clone the Magma Repository *Download the Magma benchmark by cloning its repository:*

```
git clone https://github.com/HexHive/magma.git
```

Step 3: Move Lyso into Magma *To integrate Lyso with Magma, move the Lyso folder into the Magma fuzzers directory:*

```
mv /path/to/lyso magma/fuzzers/
```

This ensures that Lyso is correctly placed within the Magma environment.

A.3.2 Basic Test

To verify that Lyso is correctly installed and functional, perform the following steps. This test ensures that all required software components are in place and that Lyso can interact with the Magma benchmark.

Step 1: Using Captain Scripts for Fuzzing Campaigns *Magma provides the Captain scripts (located in `magma/tools/captain`) to build, start, and manage fuzzing campaigns. The script `magma/captain/run.sh` automates the process by:*

- *Building the required fuzzing images.*
- *Configuring parallel campaign execution.*
- *Managing fuzzing runs based on a configuration file (`captainrc`).*

Step 2: Configuring a Fuzzing Campaign *To run a 24-hour Lyso fuzzing campaign on a Magma target (e.g., `libpng`) one time and store the result in the workdir directory, update the `captainrc` file with the following settings:*

```
# WORKDIR: Directory for shared volumes
WORKDIR=./workdir

# REPEAT: Number of campaigns per program
REPEAT=1

# TIMEOUT: Campaign duration
# (supports s/m/h/d suffixes)
TIMEOUT=24h

# FUZZERS: List of fuzzers to evaluate
# (from magma/fuzzers/*)
FUZZERS=(lyso)

# Targets to fuzz with Lyso
# (from magma/targets/*)
lyso_TARGETS=(libpng)
```

Step 3: Running the Campaign *Once the `captainrc` file is configured, execute the run.sh script in the same directory.*

Step 4: Verifying the Output *During execution, the build and run logs of the campaign are stored in the workdir/log directory. In addition, the fuzzer logs and outputs can be found in workdir/ar/lyso/libpng/libpng_read_fuzzer/0/findings.*

To collect TTE and TTR data generated by Magma, refer to the monitor files located inside workdir/ar/lyso/libpng/libpng_read_fuzzer/0/monitor. Each monitor file is named using a timestamp (in seconds) representing the elapsed time since the start of the campaign. These files contain a CSV header and data rows that track the global campaign's bug reach and trigger counters at each timestamp.

If Lyso is functioning correctly, you should observe log entries indicating successful campaign execution and active fuzzer operations.

A.4 Evaluation workflow

A.4.1 Major Claims

The evaluation of Lyso aims to validate its effectiveness in verifying static analysis alarms. The major claims made in the paper are:

(C1): *The evaluation demonstrates that Lyso achieves an average 12.17× speedup over existing fuzzers in triggering bugs while also discovering the highest number of vulnerabilities in the Magma benchmark.*

A.4.2 Experiments

To validate Claim C1, we conducted experiments using Lyso on the Magma benchmark, comparing its performance

against state-of-the-art fuzzers, including AFL, AFL++, MOPT, AFLGo, Titan, FishFuzz, SelectFuzz, Parmesan. The primary objective is to measure Time-to-Exposure (TTE), Time-to-Reach (TTR), and Success Rate (SR) results.

(E1): *Performance Evaluation of Lyso [24 compute-hour per program]: This experiment evaluates Lyso's effectiveness in reducing Time-to-Exposure (TTE) and Time-to-Reach (TTR) while improving Success Rate (SR). The expected outcome is a significant decrease in TTE and TTR compared to eight state-of-the-art fuzzers, along with a higher SR in successfully triggering bugs.*

Preparation:

- *Set up the Magma benchmark following the A.3.1 instructions.*
- *Place Lyso in the magma/fuzzers directory.*
- *Ensure all baseline fuzzers are configured in the magma/fuzzers directory.*
- *Configure the* `captainrc` *file (located in magma/tools/captain) to define fuzzing parameters: FUZZERS=(lyso) TARGETS=(libpng libtiff libxml2 libsndfile lua poppler sqlite3 php openssl). This ensures that Lyso is executed on all buggy programs listed in Table 2.*

Execution: *Start the fuzzing campaigns for Lyso by executing the run.sh script located in magma/tools/captain. This will initiate the fuzzing process and generate a workdir directory, which will store all results, including TTE, TTR, SR, logs, and fuzzing statistics.*

Results: *To collect TTE, TTR and SR results, execute the following command: tools/benchd/exp2json.py workdir bugs.json*

A.5 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/usenixsec2025/.