Ahoy SAILR! There is No Need to DREAM of C: A Compiler-Aware Structuring Algorithm for Binary Decompilation

Zion Leonahenahe Basque, Ati Priya Bajaj, Wil Gibbs, Jude O'Kain, Derron Miao, Tiffany Bao, Adam Doupé, Yan Shoshitaishvili, and Ruoyu Wang, *Arizona State University*

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

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

USENIX Security '24 Artifact Appendix: Ahoy SAILR! There is No Need to DREAM of C: A Compiler-Aware Structuring Algorithm for Binary Decompilation

Zion Leonahenahe Basque, Ati Priya Bajaj, Wil Gibbs, Jude O'Kain, Derron Miao,
Tiffany Bao, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang
Arizona State University
{zbasque,atipriya,wfgibbs,judeo,derronm,tbao,doupe,yans,fishw}@asu.edu

A Artifact Appendix

A.1 Abstract

The SAILR paper introduces a new decompiler, control flow structuring algorithm, and decompiler evaluation framework. As such, the main artifacts of SAILR are the ANGR DECOMPILER (with the novel SAILR structuring algorithm), the SAILR evaluation framework, and the decompilation created as a result of the former artifacts. Additionally, ANGR DECOMPILER makes public 3 algorithms previously untestable (Phoenix, DREAM, and rev.ng). The ANGR DECOMPILER and the SAILR evaluation framework are written in Python 3 and are provided as a package. Both packages are runnable through a provided Dockerfile.

SAILR was evaluated on 26 Debian packages compiled on optimization level O2. The compiled, decompiled, and measurement artifacts are provided publicly for all decompilers evaluated in the SAILR paper, including closed-source decompilers. This artifact evaluation aims to show that decompiler, structuring algorithms, and evaluation framework are accessible and functional.

A.2 Description & Requirements

All artifacts should be used on an Ubuntu 22.04 machine with at least 8 cores and 32GB of RAM. You need to have Python 3, Docker, Java 17, and Graphviz installed on the machine. In most cases, all experiments are run inside the provided container. We provide a frozen version of ANGR DECOMPILER with SAILR and the SAILR evaluation framework. We also provide a link to all decompilation generated to create the results found in Section 8 and Appendix of the main paper.

A.2.1 Security, privacy, and ethical concerns

ANGR DECOMPILER and the SAILR evaluation framework are Python packages that pose little risk to the user's machine. While running Joern, a subcomponent of the SAILR evaluation framework, a series of ports are used on localhost (9000+). These ports give access to Joern, which can do arbitrary code execution if exploited. These ports are only accessible on localhost.

A.2.2 How to access

All artifacts can be centrally accessed through the SAILR evaluation framework: https://github.com/mahaloz/sailr-eval. All instructions are provided through the README as well as links to all stable links listed below. The stable link for the evaluation framework and angr decompiler can be found at https://github.com/mahaloz/sailr-eval/tree/e1af48353c1c5b32cc53cbaa015722d57767bd6e and https://github.com/mahaloz/angr-sailr/tree/be3855762a84983137696aa14efe2431a86a7e97 respectively. The decompilation results for each decompiler can be found on the Arizona State University Dropbox at this link.

A.2.3 Hardware dependencies

To run all artifacts you must have an x86 machine with at least 8 cores and 32GB of RAM. We recommend having 80GB of storage free on the machine.

A.2.4 Software dependencies

All artifacts should be run on Ubuntu 22.04 with Python 3, Docker, Java 17, and Graphviz installed on the machine. A `setup.sh` script is provided in the evaluation framework code to install these dependencies on Ubuntu 22.04.

A.2.5 Benchmarks

SAILR was evaluated on 26 Debian packages found at the Dropbox link in A.2.2. This set should be downloaded and untared for use in validating the functionality of the evaluation framework. The total size of the uncompressed dataset is 38gb.

A.3 Set-up

A.3.1 Installation

Clone `sailr-eval` repo and use the `setup.sh` script to install the needed dependencies and start building the evaluation pipeline docker container. This will use approximately 6.5gb. Note, you will also need to build in the ANGR DECOMPILER docker container found in the "using angr decompiler" section of the README.

A.3.2 Basic Test

Use the `motivating_example` found in the `tests` folder to validate the ANGR DECOMPILER. This can be found in the README after setting up the ANGR DECOMPILER. To validate the `sailreval` package installed successfully, run `./eval.py --help`. If the script does not crash, the package is installed correctly.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *The* ANGR DECOMPILER *reimplements the Phoenix, DREAM, and rev.ng (combing) control flow structuring algorithms. It also implements the novel SAILR structuring algorithm which produces decompilation that is deoptimized. This is proven by experiment (E1).*

(C2): *The SAILR evaluation framework can be used to measure decompilation by the same metrics shown in Section 8 of the SAILR paper. This is proven by experiment (E2).*

(C3): *The decompilation used to generate the results in Table 3 of the SAILR paper is public and aggregatable. This is proven by experiment (E3).*

A.4.2 Experiments

[Mandatory for Artifacts Functional & Results Reproduced, optional for Artifact Available] Link explicitly the description of your experiments to the items you have provided in the previous subsection about Major Claims. Please provide your estimates of human- and compute-time for each of the listed experiments (using the suggested hardware/software configuration above). Follows an example:

(E1): *[Decompiler Test] [15 human-minutes + 1 build-hour + 9GB disk]: In this experiment, the decompilation shown in Figure 1 of the SAILR paper is reproduced by running the decompiler*

Preparation: *First, build the* ANGR DECOMPILER *Docker container shown in the "using angr decompiler" section of the README. This should take no longer than an hour of build time.*

Execution: *Using the execution line shown in that section of the README. To run each structuring algorithm, change the —structuring–algo flag. Use sailr, phoenix, dream, and combing. The combing structuring algorithm is rev.ng.*

Results: *Compare the results of sailr, phoenix, and dream outputs to that in Figure 1 of the SAILR paper. Each output should look nearly the same (with the exception of variable names and spacing).*

(E2): *[Eval Framework Test] [20 human-minutes + 3 compute-hours + 9GB disk]: In this experiment, a simple measurement is run on one of the 26 Debian packages, Cronie*

Preparation: *Make sure you already ran the setup.sh script.*

Execution: *Following the examples, use the —compile, —measure, and —decompiler for the cronie package with as many cores as you can (specified with —cores). Use the gotos, bools, func_calls, and cfged metrics to match the SAILR paper's metrics.*

Results: *Following the README, find the sailr_compiled, sailr_decompiled, and sailr_measured folders. You should find all decompilers decompilation in the decompilation folder. You should find tomls in the measured folder with the results for each decompiler.*

(E3): *[Data Validation] [30 human-minutes + 5 compute-minutes + 40GB disk]: In this experiment, decompilation and measurements from the 26 Debian packages are used to reproduce the paper results.*

Preparation: *Download the dataset from Dropbox linked in A.2.2 and decompress it.*

Execution: *First, take a look at a few folders, like cronie, found in the O2 folder of the results. You should notice that these results look very similar to your results collected in (E2). At the very least, there should be decompilation for each decompiler and some tomls. Next, use the —summarize–targets command of the eval.py file as shown in the "SAILR evaluation results files" section of the README. You only need to edit the —opt–levels in that command to only have O2.*

Results: *Look at the last results printed by the script. These numbers should match up with the numbers shown in Table 3 of the SAILR Paper.*

A.5 Notes on Reusability

The ANGR DECOMPILER can produce decompilation that may vary in variable names and white spacing on each run. Due to this, the exact names of each variable shown in the produced decompilation may differ from the names shown in the SAILR paper. This is also true for the white spacing.

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/.