USENIX Security '25 Artifact Appendix: Save what must be saved: Secure context switching with Sailor

Neelu S. Kalani [*]
EPFL & IBM Research — Zurich

Thomas Bourgeat
EPFL

Guerney D. H. Hunt
IBM T. J. Watson Research Center

Wojciech Ozga
IBM Research — Zurich

A Artifact Appendix

A.1 Abstract

We introduce a tool called Sailor that automatically identifies ISA-state that must be swapped across context switches to prevent information leaks or computational integrity breaches. Sailor parses traces generated from an existing tool, Isla, a symbolic execution engine for ISA specifications written in the domain-specific language, Sail. We demonstrate Sailor for the RISC-V architecture. In this work, we extend Isla to automatically generate traces for multiple instructions, in the same format as provided by the RISC-V specification manuals. Based on the identified ISA-state, we automatically generate tests to check if the context switch implementations on existing systems swaps the state correctly or not.

The associated artifacts include:

1. The patch to Isla for generating traces for all instructions in a general-purpose RISC-V processor (rv64gc).
2. The python scripts to compute the ISA-state relevant for context switching. This includes parsing of the Isla traces (ISA-Inspector) and the classifier algorithm (Analyzer), as shown in Figure 3 of the paper.
3. The automatic test generator (also shown in Figure 3) that generates tests to provide experimental evaluation of existing systems.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Executing the artifacts does not impose any risk for the evaluators, since our work is more focused on the defensive side of security using ISA specifications in Sail and the Isla symbolic execution engine to generate insights useful for implementing secure context switching. We do have tests that help us uncover bugs in existing systems. However, these tests don't perform attacks, rather serve as proof-of-concept for information leakage. Further, these tests require testing on the StarFive VisionFive2 board (we provided access to it via ssh) and not on the evaluators machine.

There are no data privacy or other ethical concerns, since we do not handle private or sensitive data in our work.

A.2.2 How to access

The artifact is accessible on GitHub for the purpose of functionality and reproducability checks. (GitHub link: `https://github.com/neeluk7/sailor_artifact.git`). The artifact is also hosted on Zenodo: `https://doi.org/10.5281/zenodo.15613680`.

A.2.3 Hardware dependencies

We ran our experiments on a Linux (v6.8.0) Ubuntu 24.04.2 LTS server equipped with Intel Xeon Gold 5520+, with 112 cores and 504 GB RAM. Running our experiments on any machine with fewer resources should not affect reproducability for all experiments except for the Isla trace generation process. The trace generation takes 20 hours on our server, however, it might take more time depending on the machine being evaluated on. We provide pre-generated traces which require about 3 GB of storage and tests to generate full traces and also a subset of traces.

A.2.4 Software dependencies

We have tested the experiments mainly on Linux (reproducing them on a MacOS should be possible, but we recommend the evaluators to use a Linux system if available). The software dependencies are shown in Table 1 and the Makefile has all the required commands to install these.

A.2.5 Benchmarks

None.

[*]Majority of the work was done while the author was at IBM Research — Zurich.

Software	Version
RISC-V Toolchain	13.3.0
Opam	2.1.5
Rustc	1.86.0
Python3	3.12.3

Table 1: Software dependencies.

A.3 Set-up

A.3.1 Installation

The instructions and commands to install the dependencies and setup the artifact are included in the README.md and the Makefile. This includes setting up the software dependencies mentioned in Table 1 as well as Sail and Isla. Evaluators can follow the steps below. (In case of difficulties,

1. Run `make setup-dependencies` to install all dependencies.
2. Run `make setup-sail` and `make setup-isla`. These commands will apply the patches to Sail and Isla and build Isla.
3. Download the Isla traces into the *isla_traces_dir* directory inside the root directory of the artifact from Google drive (`https://drive.google.com/drive/folders/1FI_wnHABUfFjuzru2wMaTpUQf9I1Qw3r?usp=share_link`); the traces consume about 3 GB of disk space.

A.3.2 Basic Test

For the basic test, once the installation is complete, Isla's functionality and the parsing of Isla's traces using the python scripts in Sailor must be checked. More specifically, the following two tests must pass.

1. Isla runs correctly after applying the patch with our modifications. For this, run `make isla-traces-test`. It will produce test traces in the *isla_traces_test* directory and use the `diff` command to compare against expected output in *expected_results/isla_test_traces*.
2. Our script parses the Isla traces correctly. Run `make isla-parse-test`, which will generate the output in the *parse_test_output.txt* and compare it against the expected result using `diff`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Our framework Sailor, leverages existing tool, Isla, and the machine-readable ISA-specification in Sail, to identify ISA-state that must be swapped during context switching. The framework produces results as shown in Tables 1, 2, and 3 in the paper. We will regenerate these results using experiment (E1).

(C2): Based on the identified results, we perform a security analysis of three existing systems: Keystone, Komodo, and StarFive VF2 (as shown in Table 4 in the paper). We will corroborate these findings using experiment (E2).

A.4.2 Experiments

(E1): *[Produce security-sensitive ISA-state using Sailor] [30 human-minutes + 1 compute-hour]:*
Preparation: The Isla traces must be downloaded and present in the *isla_traces_dir*, before performing this experiment.
Execution: Run the `make run-sailor` command which will parse all the Isla traces and produce the ISA isnights corresponding to Tables 1, 2, and 3 in the paper as well as run the analyzer (classifier algorithm) to generate insights on which state to swap during context switches. This command will also use the `diff` utility to compare against expected results and automatically generate the tests.
Results: CSVs corresponding to Tables 1, 2, and 3:
1. CSVs/csr_(read/write)_access.csv
2. CSVs/csr_footprint_per_instruction_(machine/ supervisor/user).csv
3. CSVs/instruction _access_per_mode.csv

Further, the security-sensitive insights from the analyzer will be generated in `switch-from-U-to-U.csv`. The tests will be generated in the *test-generator/tests* directory.

(E2): *[Security analysis] [30 human-minutes + 30 compute-minutes]:*
Preparation: Part of this step requires the StarFive VisionFive 2 board to run the automatically generated tests. The time to run the actual tests is less than 5 minutes. We provided remote access to our board via ssh for the evaluation.
Execution: The aim of this step is to verify the results reported in Table 4 of the paper. Keystone and Komodo are both included in the repository as submodules for helping with the simple check required to confirm the finding using `grep`. Run the `grep -nriI "senvcfg" .` commands from within the *keystone* and the *serval/monitors/komodo* directories. The expected output of the command is that no match will be found, since these monitor implementations do not correctly configure/swap the `senvcfg` CSR. Next, the tests automatically generated using experiment (E1) should be run on the StarFive VisionFive 2 board. There are screenshots of the scripts run and expected output in the drive link `https://drive.google.com/drive/folders/1EDGzDCXFdQ6UgCNybJUyhg6bmKnJuuGY?usp=share_link`. As done in the scripts, it is important

to use taskset to run the tests on the same core to observe the findings.

Results: Once the execution is done and produces expected output, the findings reported in Table 4 are verified.

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/.