

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

InstrSem: Automatically and Generically Inferring Semantics of (Undocumented) CPU Instructions

Lorenz Hetterich, Fabian Thomas, Tristan Hornetz, and Michael Schwarz, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity26/presentation/hetterich-lorenz-instrsem

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: InstrSem: Automatically and Generically Inferring Semantics of (Undocumented) CPU Instructions

Lorenz Hetterich, Fabian Thomas, Tristan Hornetz, Michael Schwarz
CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

This artifact provides the source code of InstrSem. InstrSem is a tool that allows automatic reverse-engineering of undocumented instructions across a range of architectures. Alongside the source code of InstrSem, we provide the required scripts to reproduce the main claims of the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact contains proof-of-concepts to show how InstrSem can reverse-engineer a wide range of instructions. The experiments mostly run in QEMU inside a Docker container and should not affect the host device. No personal data is collected or transmitted in any way.

A.2.2 How to access

The artifact is available on Zenodo:
https://zenodo.org/records/17974657

A.2.3 Hardware dependencies

The artifact was tested on an x86-64 based machine running Ubuntu 24.04. The experiments required to verify the main claims of the paper do not rely on specific hardware. For completeness, the code for experiments relying on specific hardware is also included in the artifact.

A.2.4 Software dependencies

The following software is required on the reviewer's machine.

- `python3`.
- `docker`: The active user must be part of the `docker` group.
- `sh`.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Download the artifact from Zenodo and extract it. Navigate to the root of the artifact and run the `setup_docker.sh` script. This script sets up the required docker container. Most experiments can then be run by executing a python script.

A.3.2 Basic Test

After setting up the docker container, a basic test is provided that reverse-engineers multiple instructions across different architectures.

Execution:

```
python3 basic_demonstration.py
```

Execution may take up to two hours.

Expected results: The output should contain semantics for the reverse-engineered instructions. It should roughly match the sample output provided in `results/basic_demonstration.txt`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): InstrSem is generic and can recover and generalize instruction semantics across ISAs (**E1, E2, E3, E4**).
(C2): InstrSem can uncover the semantics of undocumented LoongArch64 vector instructions (**E4**).
(C3): InstrSem can recover 34 of 38 64-bit RISC-V integer instructions (**E2**).
(C4): InstrSem is also applicable to complex ISAs like x86-64 (**E3**).
(C5): InstrSem can be run on a full RISC instruction set (**E4**).

(C6): The amount of covered encodings drastically increases due to InstrSem's generalization step (**E4**).
(C7): InstrSem triggers QEMU crashes (**E5**).

A.4.2 Experiments

If required, more information for each experiment is provided in the `README.md` file of the artifact. Additional experiments that rely on special hardware and are not required to verify the main claims of the paper are also described in the `README.md` file of the artifact.

(E1): [basic_demonstration] [5 human-minutes 2 compute-hours]: Basic demonstration of InstrSem.
Preparation:

```
sh setup_docker.sh
```

Execution:

```
python3 basic_demonstration.py
```

Results: Instruction semantics for multiple instructions should be successfully recovered and generalized. The output should roughly match the sample output provided in `results/basic_demonstration.txt`.

(E2): [eval_riscv64] [5 human-minutes 2 compute-hours]: Recovering the semantics of RISC-V 64-bit Integer Instructions.
Execution:

```
python3 eval_riscv64.py > table_rv64.md
```

Results: The resulting table in `table_rv64.md` should match the sample table provided in `results/eval_riscv64.md` apart from possible differences in execution time. In rare cases, `slti` or `sltiu` can fail to recover and generalize semantics successfully. If this is the case, you can repeat the experiment.

(E3): [eval_x86-64] [5 human-minutes 2 compute-hours]: Recovering the semantics of common x86-64 instructions.
Execution:

```
python3 -u eval_x86-64.py > table_x86-64.md
```

Results: The resulting table in `table_x86-64.md` should match the sample table provided in `results/eval_x86-64.md` apart from possible differences in execution time.

(E4): [eval_loongarch64] [5 human-minutes 10 compute-hours]: Running InstrSem on the full LoongArch instruction space.
Execution:

```
python3 eval_loongarch64.py
```

We encourage you to run this evaluation for 10h but shorter or longer runs are also possible. When you are done, you can use CTRL+C to terminate the evaluation. This might still lead to an active docker container which you can terminate. You can also interrupt the evaluation using CTRL+C (only shortly press it to raise a single `KeyboardInterrupt`). Then, when you later start the script again, it should continue instead of starting from scratch.
Results: The output of InstrSem for newly discovered instructions is written to one file in the `out` directory for each instruction. We expect a large number of files to exist in this directory. Further, whenever a new instruction is discovered, the script prints the runtime (in seconds) and total number of recovered encodings to stdout. If you run the evaluation for 10 hours, we expect at least 600 million instruction encodings to be covered.
Further, we expect the semantics of several vector instructions to be recovered. To filter for possible vector instructions, you can use the following commands:

```
grep -r out -e vector
```

(E5): [eval_qemu_crash] [1 human-minute]: QEMU crashes.
Execution:

```
python3 eval_qemu_crash.py
```

Results: The execution should print multiple error-messages:

```
ERROR:../target/loongarch/tcg/insn_trans/
    trans_vec.c.inc:3574:vldi_get_value: code
    should not be reached
Bail out! ERROR:../target/loongarch/tcg/
    insn_trans/trans_vec.c.inc:3574:
    vldi_get_value: code should not be reached
**
```

Finally, InstrSem will give up recovering semantics with a `too many sample collections failed!` exception:

```
Exception: too many sample collections failed!
```

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/usenixsec2026/.