

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

RecStruct: Recovering Nested Struct Types from Stripped Binaries via Stack-Driven Unification

Yuxin Chen, Zhiyang Fang, Shiyi Wu, Yixin Xu, and Yuhang Wang, *Sichuan University;* Xiaokang Yin, *Information Engineering University;* Junfeng Wang, *Sichuan University*

https://www.usenix.org/conference/usenixsecurity26/presentation/chen-yuxin

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: RecStruct: Recovering Nested Struct Types from Stripped Binaries via Stack-Driven Unification

Yuxin Chen, Zhiyang Fang, Shiyi Wu, Yixin Xu, Yuhang Wang, Xiaokang Yin, Junfeng Wang

A Artifact Appendix

A.1 Abstract

The artifact contains the complete implementation of RecStruct, a tool for recovering nested and recursive structure types from stripped binaries using Stack-Driven Unification. It includes the Rust source code, seven benchmark datasets with stripped inputs and pre-extracted YAML ground truth, and tools for running the analysis and computing the metrics used in the paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None. The experiments analyze pre-compiled open-source binaries and do not process sensitive data or modify system configuration. Network access is only needed during setup if Cargo dependencies are not already cached.

A.2.2 How to access

The artifact is archived on Zenodo: version DOI https://doi.org/10.5281/zenodo.20628819; concept DOI https://doi.org/10.5281/zenodo.20255915.

A.2.3 Hardware dependencies

The reference experiments used Ubuntu 24.04 on a server with 32 vCPUs and 46GB RAM. We recommend 16GB RAM for the small and medium datasets, 32GB RAM for QEMU and OpenSSL, and at least 20GB free disk space for the downloaded archives, extracted datasets, builds, and results. No GPU or specialized hardware is required.

A.2.4 Software dependencies

- Linux x86-64 (Ubuntu 22.04 or later; tested on Ubuntu 24.04)
- Rust toolchain $\geq$ 1.88 (edition 2024; tested with Rust 1.93)
- Cargo and standard build tools (e.g., Ubuntu's `build-essential`)
- Python $\geq$ 3.10 only for optional auxiliary scripts

Cargo fetches and builds all Rust dependencies. IDA is not required for the experiments below because the archive includes pre-extracted ground truth; the IDA script is provided only for optional ground-truth re-extraction.

A.2.5 Benchmarks

The archive contains `coreutils-O1`, `coreutils-O2`, `coreutils-O3`, `utils`, `binutils`, `qemu`, and `openssl`. Each dataset contains `bin/` (with debug information), `strip/` (the stripped analysis inputs), and `groundtruth.yaml`.

A.3 Set-up

Download `RecStruct-source.tar.gz` and `RecStruct-dataset.tar.gz` from Zenodo, place them in an empty directory, and run:

```
tar -xzf RecStruct-source.tar.gz
tar -xzf RecStruct-dataset.tar.gz
cd RecStruct/struct-recover-rs
cargo build --release
cd ../struct-recover-eval/tools
cargo build --release
```

The commands below assume the resulting layout in which `RecStruct/` and the seven dataset directories share a parent directory. They explicitly set `-stride` and `-output`; this is required because the legacy defaults in version 5 contain `/home/user` paths.

A.3.1 Basic Test

From `RecStruct/struct-recover-rs`, run:

```
./target/release/stride pipeline \
  --bin testcase/old-case --mode full \
  --output target/release/output
```

The command should finish without error after analyzing 24 functions and write `old-case.h` and `old-case.yaml` under `target/release/output/`. The output contains the four principal interdependent structures and their nested/cyclic pointer relationships shown in Figure 3 and Table 2 of the paper (additional internal type groups are also reported).

A.4 Evaluation Workflow

A.4.1 Major Claims

(C1): RecStruct executes end-to-end on a stripped ELF binary and recovers nested and recursive structure definitions (RQ1, Figure 3 and Table 2). This is exercised by E1.

(C2): RecStruct's dataset pipeline computes layout accuracy, nesting recognition, and Trex-style hierarchical scores (RQ2, Tables 3–5). This is exercised by E2.

(C3): RecStruct records end-to-end runtimes and supports analysis of large binaries with a two-hour timeout per binary (RQ3–RQ4, Tables 6–8). This is exercised by E2.

(C4): The runner exposes LOCAL, INTRA, and GLOBAL analysis scopes used by the Stack-Driven Unification ablation (RQ5, Figure 6). This is exercised by E3.

A.4.2 Experiments

(E1): End-to-end micro-benchmark [5 human-minutes + less than 1 compute-minute]. Run the Basic Test and inspect `old-case.h` and `old-case.yaml`. Successful file generation and the four mapped structures validate C1.

(E2): Dataset-scale analysis and evaluation [10 human-minutes + dataset-dependent compute time]. From `RecStruct/struct-recover-eval/tools`, run, for example:

```
D=coreutils-O1
V=baseline-ae
STRIDE=../../struct-recover-rs/target/\
release/stride
./target/release/run-dataset \
  --dataset ../../../$D \
  --variant "$V" \
  --stride "$STRIDE" \
  --output ../results
./target/release/run-eval \
  --dataset ../../../$D \
  --results ../results/$D-stride-$V
```

Replace `coreutils-O1` in both commands with another dataset name to evaluate it. The first command creates `all.yaml`, `summary.md`, and `run_stats.json`. The second prints an evaluation summary and creates `eval2`, `eval3`, and `eval4` reports in Markdown/JSON plus `eval_summary.json`. Eval2 corresponds to layout accuracy (Table 3), Eval3 to nesting recognition (Table 4), and Eval4 to the Trex-style score (Table 5). On the archived inputs, representative F1 scores are approximately 42%/17% for `coreutils-O1` and 18%/5% for `utils` (layout/nesting); minor differences may occur across toolchains. Runtime is machine-dependent: the reference totals are about one minute for `coreutils-O1`, 1.95 hours for `binutils`, and 19.9 aggregate compute-hours for `qemu`. One QEMU binary may reach the documented two-hour timeout.

(E3): Analysis-scope ablation [5 human-minutes + three E2 runs]. Repeat the first E2 command with `-analysis-mode 0`, `1`, and `2`, using distinct variant names such as `ablation-local`, `ablation-intra`, and `ablation-global`; then run `run-eval` on each resulting directory. These modes correspond to LOCAL, INTRA, and GLOBAL in Figure 6 and exercise C4. Detailed baseline and batch-analysis instructions are in the archived README.

The artifact received the Artifacts Available and Artifacts Functional badges; Results Reproduced was not applied for. The commands above therefore document the functional evaluation workflow and do not assert a Results Reproduced badge.

A.5 Version

Based on the Artifact Evaluation template V20231005. The evaluation and badging methodology is documented at https://secartifacts.github.io/usenixsec2026/.