WOOT'26 Artifact Appendix: SEMSAN: a Configurable Sanitizer for Detecting System-Level Semantic Bugs

Moritz Sanft
Ruhr-Universität Bochum

Flavio Toffalini
Ruhr-Universität Bochum

A Artifact Appendix

This document describes the artifacts accompanying the WOOT'26 publication SEMSAN*: a Configurable Sanitizer for Detecting System-Level Semantic Bugs*. The artifact includes the SEMSAN prototype, its four Sanitizer Primitive implementations, test suites, and benchmark scripts used to reproduce the evaluation results.

A.1 Description & Requirements

A.1.1 Security, privacy, and ethical concerns

SEMSAN loads eBPF programs into the Linux kernel and therefore requires root (or `CAP_BPF`/`CAP_PERFMON`) privileges during installation and execution. The artifact does not transmit any data externally and does not modify production system state. All vulnerabilities reported in the paper have been responsibly disclosed and either patched by the respective vendors or are under embargo.

A.1.2 How to access

The artifact is available at the GitHub repository https://github.com/AutoSecLab/SemanticSanitizer in the branch `artifact`. The repository contains the SEMSAN source code, Sanitizer Primitive definitions, test suite, and benchmark automation scripts. In addition, we make the artifact available on Zenodo https://doi.org/10.5281/zenodo.20020039.

A.1.3 Hardware dependencies

Experiments were conducted on a physical machine equipped with an AMD EPYC 9B45 (Turin) CPU (x86_64, 4 cores / 8 threads) and 32 GB of RAM. Exact reproduction of numeric results is most likely on an identical or similar configuration. An x86_64-Linux machine is *required*; ARM and other architectures are not supported by the eBPF backend. No GPU or specialized hardware is needed.

A.1.4 Software dependencies

The artifact relies on the *Nix* package manager to provide a fully reproducible build environment. All other dependencies (LLVM, libbpf, Rust toolchain, Go, `just`) are managed by Nix and do not need to be installed manually. A Linux kernel version ≥ 5.15 with BTF support is required.

A.2 Set-up

A.2.1 Installation

Install the Nix package manager (Determinate Systems installer recommended for Ubuntu):

```
curl --proto '=https' --tlsv1.2 -sSf -L \
  https://install.determinate.systems/nix \
  | sh -s -- install
source /nix/var/nix/profiles/default/etc/profile.
```

Enter the reproducible development shell:

```
nix develop
```

Generate the eBPF bindings (may take up to 5 minutes):

```
just codegen
```

Build the SEMSAN CLI:

```
just build
```

A.3 Evaluation workflow

A.3.1 Major Claims

(C1): SEMSAN discovers five previously unknown real-world vulnerabilities across Git, Docker (Moby), SoftServe, ViewVC, and Grafana (Section 7.1). This is supported by experiment (E1).

(C2): SEMSAN correctly detects true vulnerabilities and produces zero false positives during two months of real-world deployment and throughout the performance benchmarks (Section 7.2). This is supported by experiment (E2).

(C3): SEMSAN's performance overhead ranges from 3.15% (System Call Sanitizer) to 20.85% (Directory Ownership Sanitizer) in the micro-benchmarks, and introduces negligible overhead ($< 5\%$) on real-world workloads (PostgreSQL pgbench, Apache HTTP server) in the macro-benchmarks (Section 7.3). This is supported by experiment (E3).

(C4): SEMSAN can be combined with coverage-guided fuzzing to surface semantic violations that do not cause crashes, as demonstrated by the `crun` fuzzing campaign (Section 7.4). This is supported by experiment (E4).

A.3.2 Experiments

(E1): [20 human-minutes + 10 computer-minutes] This experiment replays the proof-of-concept exploits for the publicly disclosed vulnerabilities discovered with SEMSAN (C1). The Grafana bug is excluded as it remains under embargo.
How to: From within the Nix shell (nix develop), follow the per-vulnerability instructions in `ARTIFACT_EVAL.md` (Section RQ1) to replay the PoC for each public bug (Git, Docker Moby, Soft-Serve, ViewVC) with the corresponding Sanitizer Primitive enabled.
Results for Claim 1: SEMSAN must trigger an alarm for each replayed PoC. None of these bugs produce a crash or are caught by standard sanitizers such as ASan or UBSan.

(E2): [10 human-minutes + 5 computer-minutes] This experiment verifies SEMSAN's detection accuracy by running the bundled test suite, which contains positive and negative cases for each Sanitizer Primitive. To assess false positives, we leave detailed indication for installing SEMSAN in the system.
How to: From within the Nix shell (nix develop), run the test entrypoint:

```
just test
```

Follow "System Level Installation" for detailed installment instructions.
Results for Claim 2: All test cases should pass with no false positive or false negative. While the system installation should not write "All sanitizers are running" log in the journal.

(E3): [10 human-minutes + 1.5 computer-hours] This experiment measures the runtime overhead of SEMSAN using micro- and macro-benchmarks.
How to: Run the micro-benchmark (approx. 20 minutes):

```
nix run .#artifact-eval.micro-benchmark
```

Then run the macro-benchmark (approx. 1 hour):

```
nix run .#artifact-eval.macro-benchmark
```

Results for Claim 3: The micro-benchmark produces a table comparable to Table 5, with overhead between 3.15% and 20.85%. The macro-benchmark produces results comparable to those shown in the Apache and PostgreSQL figures in Section 7.3, with overhead below 5%.

(E4): [30 human-minutes + variable computer-time] This experiment demonstrates SEMSAN's integration with a grammar-guided fuzzer (LibAFL/Nautilus) targeting the `crun` OCI container runtime.
How to: From the repository root (as root, since `crun` requires Linux namespaces and cgroups), run:

```
nix run .#artifact-eval.crun-campaign
```

This opens a terminal multiplexer showing the SEMSAN output and the LibAFL/Nautilus fuzzer output side by side. The campaign can be stopped after any duration.
Results for Claim 4: The fuzzer panel should show steadily increasing coverage (`shared_mem`). SEMSAN reports semantic violations (e.g., container escape attempts) in its panel that do not manifest as crashes and would therefore be missed by the fuzzer alone.

A.4 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.