NEURAL INFORMATION PROCESSING SYSTEMS

QCRI
معهد قطر لبحوث الحوسبة
Qatar Computing Research Institute
جامعة حمد بن خليفة
HAMAD BIN KHALIFA UNIVERSITY

NeurIPS 2026 · Evaluations and Datasets Track

VeriVul

A Verification-Guided Framework for Generating Realistic Vulnerability Benchmarks

Ahmed Lekssays, Hamza Mouhcine, Issa Khalil

Qatar Computing Research Institute, Hamad Bin Khalifa University

THE PROBLEM

LLM vulnerability benchmarks pair real code with noisy labels, or verified labels with small synthetic programs, and fixed benchmarks leak into training data.

OUR SOLUTION

VeriVul turns real CVE fixes into self-contained programs whose labels a model checker proves, and regenerates them from new CVEs on demand.

20–71%
label inaccuracy in commit-mined datasets

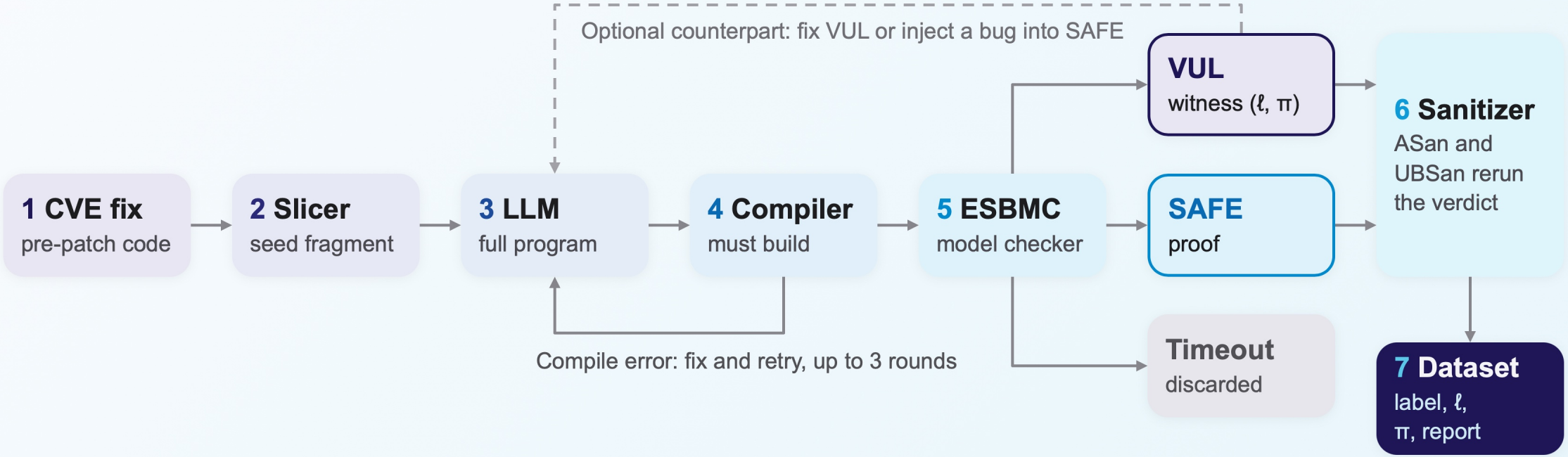
From scratch
FormAI's verified programs are small and template-like

Leakage
public CVEs and benchmarks enter LLM training corpora

Audits: Croft et al. 2023, Ding et al. 2024, Li et al. 2025

Approach

VeriVul turns a CVE fix into a verified program



Snapshot: 13,908 verified programs from 1,430 CVEs, 440 projects and 123 CWEs.

Framework overview, after Fig. 1 of the paper

CVE fix → **Slicer** → LLM → Compiler → ESBMC → Sanitizer → Dataset

The slicer keeps what the fix depends on

```
1   static int init_die(RzBinDwarfDie *die, ut64 abbr_code, ut64 attr_count) {
2 –   if (!die) {
3 –     return -1;
4 –   }
5 ›   die->attr_values = calloc(sizeof(RzBinDwarfAttrValue), attr_count);
6 ›   if (!die->attr_values) {
7 ›     return -1;
8     }
9     die->abbrev_code = abbr_code;
10    die->capacity = attr_count;
11    die->count = 0;
12 –   return 0;
13  }
```

13 → 9 lines

rizin, CVE-2021-43814, CWE-787

- › Slicing criterion: lines the CVE fix changed
- – Removed by the slicer

The seed has no main and no callers.
It does not compile on its own.

Seed: rizin commit aa6917772d, function init_die

An LLM expands the seed into a full program

```
16  typedef struct {
17    uint16_t capacity;
18    uint32_t total;
19    LogEntry *entries;
20  } Log;
 ⋮
39    log->capacity = (uint16_t)count;
40    log->total = count;
 ⋮
48    log->entries = calloc(log->capacity, sizeof(LogEntry));
 ⋮
66    for (uint32_t i = 0; i < log->total; i++) {
67      if (read_entry(f, &log->entries[i]) < 0) return -1;
68    }
```

93 lines

8 functions, reads binary log records from stdin

- 16-bit capacity sizes the buffer (lines 17, 39, 48)
- 32-bit total bounds the loop (lines 18, 40, 66)

Both fields come from one count.
Each function looks correct on its own.

Generator: gpt-oss-120b. Excerpt of the 93-line program.

CVE fix → Slicer → LLM → Compiler → **ESBMC** → Sanitizer → Dataset

ESBMC finds the input that breaks the program

```
State 10, line 25, read_uint32
  entry_count = 4278190081 (0xFF000001)
State 14, line 39, init_log
  capacity = 1
State 15, line 40, init_log
  total = 4278190081
State 102, line 48, allocate_entries
  entries = buffer of 1 LogEntry
Violated property
  line 56, read_entry
  dereference failure: array bounds violated
  CWE-787 (and 5 related CWEs)
VERIFICATION FAILED, k = 41
```

ESBMC checks every execution of the program and returns one of three verdicts.

VUL Counterexample: line ℓ and input π. Kept.

SAFE Proof over all inputs. Kept.

Inconclusive Timeout or memory limit. Dropped.

The cast keeps the low 16 bits, so capacity is 1. The loop writes a second record past the one-record buffer.

ESBMC 8.4, k-induction. Counterexample abridged.

CVE fix → Slicer → LLM → Compiler → ESBMC → Sanitizer → Dataset

The counterexample crashes the real binary

```
Input from the counterexample, 84 bytes
00 00 00 00 | 01 00 00 FF | 76 zero bytes
magic | entry_count = 0xFF000001 | records
```

```
AddressSanitizer: heap-buffer-overflow
WRITE of size 4
  #0 read_entry source.c:56
  #1 process_log source.c:67
  #2 main source.c:87
0 bytes after 40-byte region, allocated at
  #1 allocate_entries source.c:48
```

Line 56

ESBMC and ASan report the same line and defect class.

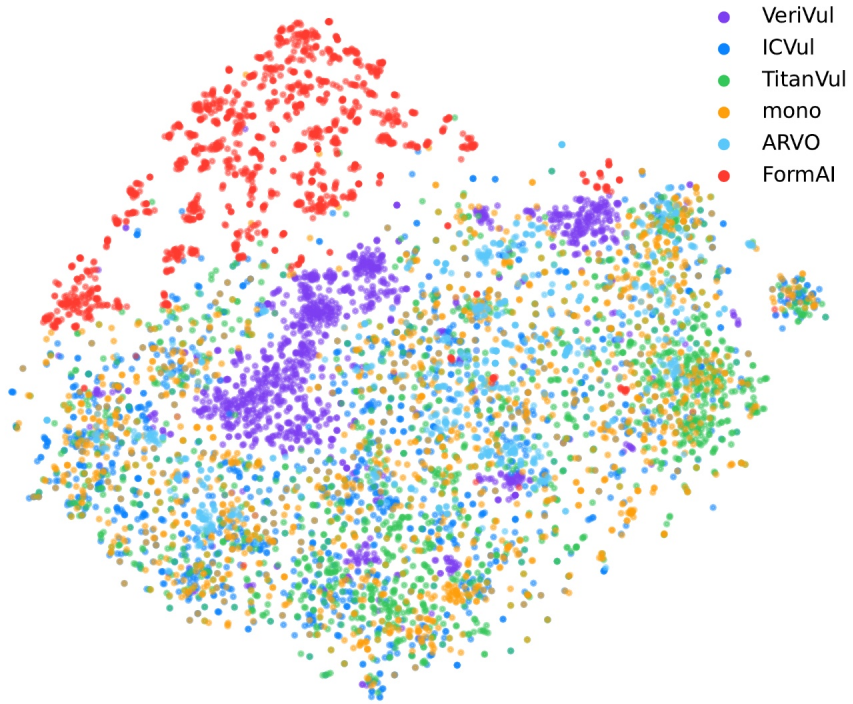
Each sample stores its label, the trigger line ℓ, the input π and the sanitizer report.

All 10,225 VUL programs in the snapshot fail under a sanitizer on a stored input.

ASan report abridged. Re-run with the pipeline's sanitizer flags.

Realism

Closer to real-world code than FormAI



Embedding view

VeriVul is closer than FormAI to ICVul, TitanVul, Mono and ARVO in 11 of 12 distances (centroid L2, per-dimension KS, MMD).

k-means with k = 6: ARI 0.261, silhouette ≈ 0.01.

Code-metric view: mean KS against ARVO, lower is closer

Category	VeriVul	FormAI	Wins VV/FA
Complexity (6)	0.536	0.699	5/1
Memory safety (5)	0.343	0.298	3/2
Control flow (4)	0.223	0.579	4/0
Naming (2)	0.536	0.655	2/0
All 17 metrics	0.405	0.548	14/3

t-SNE of jina-embeddings-v2-base-code embeddings, n = 31,917

Evaluation

Four frontier LLMs on VeriVul

Detection accuracy

53.8–71.9%

Models flag most programs as vulnerable: recall 83.1–97.1%, precision 52.2–66.4%.

Root-cause localization

7.9–14.2%

exact match with the ESBMC trigger line ℓ, on programs the model flagged correctly.

Memorization probe

46.0% vs 6.3%

of the target that Claude-Opus-4.7 reproduces as one span: public ReposVul function vs derived VeriVul program.

Models: GPT-5.4, Claude-Opus-4.7, DeepSeek-V4-Pro, GPT-OSS-120B, zero-shot. 400 vulnerable/safe pairs over eight CWEs, compared on CWEs shared with each partner dataset.

Limitations

What a VeriVul label covers

ESBMC models the environment

Library and environment models can differ from a real run. 26.8% of VUL programs depend on the environment. 1.2% of C and C++ SAFE programs fail under a sanitizer.

Property and language scope

Memory-safety and arithmetic CWEs in C and C++. Other classes need another property checker.

Generator and seed bias

Five expansion models and two seed sources (ReposVul, ICVul) shape the programs and their closeness to real code.

Labels describe the program

A label certifies the generated program. It does not certify the defect of the source CVE. 102 programs have a verified counterpart.

Code and data

Verified vulnerability programs from fresh CVEs, on demand

Framework

github.com/qcri/verivul-framework

Dataset

huggingface.co/datasets/qcri/verivul-dataset

Archive

doi.org/10.5281/zenodo.24021165