CrossCommitVuln-Bench: A Dataset of Multi-Commit Python Vulnerabilities Invisible to Per-Commit Static Analysis

Arunabh Majumdar
Independent Researcher
Mumbai, India
arunabh.majumdar@gmail.com

ABSTRACT

We present **CrossCommitVuln-Bench**, a curated benchmark of 15 real-world Python vulnerabilities (CVEs) in which the exploitable condition was introduced across multiple commits—each individually benign to per-commit static analysis—but collectively critical. We manually annotate each CVE with its contributing commit chain, a structured rationale for why each commit evades per-commit analysis, and baseline evaluations using Semgrep and Bandit in both per-commit and cumulative scanning modes. Our central finding: the per-commit detection rate (**CCDR**) is **13%** across all 15 vulnerabilities—87% of chains are invisible to per-commit SAST. Critically, both per-commit detections are qualitatively poor: one occurs on commits framed as security fixes (where developers suppress the alert), and the other detects only the minor hardcoded-key component while completely missing the primary vulnerability (200+ unprotected API endpoints). Even in cumulative mode (full codebase present), the detection rate is only 27%, confirming that snapshot-based SAST tools systematically miss vulnerabilities whose introduction spans multiple commits. The dataset, annotation schema, evaluation scripts, and reproducible baselines are released under open-source licenses to support research on cross-commit vulnerability detection.

CCS CONCEPTS

• **Security and privacy** → **Vulnerability management**; • **Software and its engineering** → *Software testing and debugging*.

KEYWORDS

vulnerability detection, static analysis, multi-commit vulnerabilities, benchmark dataset, SAST, Python, CVE, cross-commit chains

ACM Reference Format:
Arunabh Majumdar. 2026. CrossCommitVuln-Bench: A Dataset of Multi-Commit Python Vulnerabilities Invisible to Per-Commit Static Analysis. In *Proceedings of the 3rd ACM International Conference on AI-Powered Software (AIware '26).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

AIware '26, July 6–7, 2026, Montreal, QC, Canada

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Static Application Security Testing (SAST) tools—Semgrep [7], Bandit [6], CodeQL [3]—are the dominant automated security guardrails in modern software pipelines. They operate on *snapshots*: a single commit, a pull request diff, a point-in-time file tree. This design assumes that dangerous code is *introduced in a single change* and thus visible in isolation.

We challenge this assumption. A vulnerability chain can span multiple commits spread over days, months, or years:

- **Commit A**: Introduces a taint source—a user-controlled parameter routed to an internal API, a new authentication bypass, a relaxed permission check. In isolation, this looks like a legitimate feature addition.
- **Commit B**: Introduces a sink—a dangerous operation, a missing guard, a new execution path. Also individually benign. The dangerous data flow only exists when *both* commits are present.

We call such vulnerabilities *cross-commit chains*. They are structurally invisible to any tool that does not maintain state across commit history.

Prior datasets—Juliet [5], D2A [9], Devign [10]—focus on single-function or single-file vulnerabilities. No published dataset specifically targets the multi-commit introduction pattern. This gap limits the development and evaluation of cross-commit detection approaches.

Contributions.

(1) **CrossCommitVuln-Bench**: 15 manually annotated real-world Python CVEs with multi-commit introduction chains.
(2) An open **annotation schema** capturing contributing commits, chain rationale, per-commit SAST results, and why each commit evades detection in isolation.
(3) **Baseline evaluations** with Semgrep and Bandit in per-commit and cumulative modes, establishing CCDR = 13% as a reproducible lower bound.
(4) Evaluation scripts enabling independent replication of all results.

2 BACKGROUND AND RELATED WORK

2.1 SAST Tool Limitations

Per-commit SAST tools perform intra-procedural or limited inter-procedural analysis on a single snapshot. They cannot detect vulnerabilities where the taint source and sink exist in separate commits, or where a guard removal in one commit activates a dangerous code path introduced months earlier. Three failure modes are common in our dataset:

- **Custom wrappers**: Sink functions wrapped in project-internal helpers (e.g., `exec_cmd()`) are not recognized by name-pattern rules.
- **Auth dependency absence**: Missing FastAPI/Flask authentication dependencies on route handlers produce no SAST finding—there is no rule for "absent guard."
- **Incremental surface expansion**: Each feature commit extends an attack surface (new API endpoint, new taint entry point) without a directly dangerous operation in its diff.

2.2 Related Datasets

Juliet Test Suite [5] and **OWASP Benchmark** [8] provide synthetic vulnerabilities with known ground truth, useful for tool precision/recall measurement but not representative of real-world introduction patterns. **D2A** [9] and **Devign** [10] contain real CVEs but label at the function or file level without commit-sequence annotation. **Big-Vul** [2] and **CVEFixes** [1] provide fix-commit diffs but do not annotate the multi-commit introduction pattern. To our knowledge, CrossCommitVuln-Bench is the first dataset that explicitly annotates the *sequence of commits* responsible for introducing a cross-commit vulnerability chain.

3 DATASET CONSTRUCTION

3.1 Mining Pipeline

We queried the GitHub Security Advisory Database (GHSA) via the OSV API, filtering for high- and critical-severity PyPI advisories with traceable fix commits. From 1,200 advisories, 696 had verifiable GitHub fix commits, and 80 had ≥2 Python files changed in the fix. We ran automated *git blame* on the fix-modified lines for the top 30 candidates (ranked by distinct blame commits), confirming 23 of 30 as multi-commit (77%).

3.2 Manual Archaeology

For each candidate, we: (1) read the fix commit diff to identify what was corrected; (2) traced each vulnerable line to its introducing commit via `git blame`; (3) read the full diff of each introducing commit to understand its apparent intent; (4) confirmed that the commit looks individually benign and that the dangerous condition requires ≥2 commits to manifest.

3.3 Selection Criteria

We retained a CVE if and only if it satisfies all five criteria:

(1) **Multi-commit**: ≥2 distinct commits introduce the vulnerability.
(2) **Individually benign**: Each contributing commit is a plausible, legitimate change.
(3) **Collectively critical**: The combination is exploitable (CVSS ≥ 7.0).
(4) **Open source**: Repository is publicly accessible with full commit history.
(5) **Reproducible**: The vulnerable state can be reconstructed by checking out the pre-fix commit.

Of the 21 candidates we fully archaeologized, 15 passed all five criteria. Six were excluded: four were single-commit design flaws (the vulnerability was present in the initial implementation, with no completing second commit), and two had introducing commits that were not individually benign (the dangerous operation was evident in the diff). These 6 are retained in the dataset as *negative examples* with `annotation_status=SKIP` and documented rationale—they serve as calibration data for the selection criteria.

3.4 Annotation Schema

Each annotation is a JSON file with the following structure:

```
{
  "cve_id": "CVE-YYYY-NNNNN",
  "cwe_ids": ["CWE-XX"],
  "severity_combined": "critical|high",
  "fix_commit": "<sha>",
  "contributing_commits": [
    {
      "hash": "<sha>", "date": "YYYY-MM-DD",
      "role": "SOURCE|SINK|GUARD_REMOVAL|...",
      "isolated_severity": "low|benign",
      "semgrep_findings": [...],
      "bandit_findings": [...],
      "sast_flagged_relevant": false
    }
  ],
  "vulnerability_chain": {
    "description": "...",
    "why_sast_misses_per_commit": "..."
  },
  "commit_span_days": <int>,
  "ccdr_this_cve": false,
  "cdr_this_cve": false
}
```

4 DATASET CHARACTERISTICS

Table 1 summarizes the 15 annotated CVEs. The vulnerability patterns span six pattern classes across five primary CWE families (Table 2). Commit spans range from 21 to 1,342 days (median 58 days, mean 245 days), confirming that multi-commit chains can emerge over time scales from weeks to years.

Table 1: CrossCommitVuln-Bench Dataset Summary (15 CVEs)

Property	Value
Total CVEs	15
Critical severity	9
High severity	6
Repositories (unique)	15
Contributing commits (mean)	3.1
Commit span: min / median / mean / max	21d / 58d / 245d / 1,342d

5 BASELINE SAST EVALUATION

5.1 Experimental Setup

We evaluated two widely-used SAST tools: **Semgrep** v1.154.0 with `--config auto` (community rules) and **Bandit** v1.9.4 with full recursive scan (`-r`). For each CVE we ran two modes:

Table 2: Vulnerability Pattern Taxonomy

CWE	Pattern	Count
CWE-94	Code injection (eval, exec, template)	4
CWE-22	Path traversal (incl. CWE-73)	3
CWE-78	OS command injection	1
CWE-306	Missing authentication	1
CWE-943	Cypher / graph-query injection	1
Other	Crypto, XSS, deser., sig. bypass, input val.	5
	Total (primary CWE per CVE)	*15*

- **Per-commit (CCDR mode)**: checkout each contributing commit individually, run both tools, classify findings as relevant or irrelevant using a conservative CWE-to-rule mapping (see replication package). *Did the tool catch it on the commit that introduced this element?*
- **Cumulative (CDR mode)**: checkout the commit immediately before the fix (`fix_commit^`), run both tools on the full codebase. *Does the tool catch it when all commits are present?*

5.2 Metrics

- **CCDR** (Cross-Commit Detection Rate): fraction of CVEs where at least one contributing commit triggers a relevant SAST finding.
- **CDR** (Cumulative Detection Rate): fraction of CVEs where the pre-fix codebase (all commits present) triggers a relevant SAST finding.
- **Detection gap** = CDR − CCDR: the fraction of CVEs that can only be caught when the full history is available.

5.3 Results

Table 3 reports per-CVE results.

Table 3: Per-CVE Baseline SAST Results

CVE	CWE	Sev.	Per-commit	Cumulative
CVE-2025-10155	CWE-20/693	critical	missed	missed
CVE-2025-10283	CWE-22	critical	missed	missed
CVE-2025-46724	CWE-94	critical	**caught**[†]	**caught**
CVE-2025-5120	CWE-94	critical	missed	missed
CVE-2025-55449	CWE-345/798	critical	missed	missed
CVE-2025-61622	CWE-502	critical	missed	missed
CVE-2026-22584	CWE-94	critical	missed	missed
CVE-2026-2472	CWE-79	high	missed	missed
CVE-2026-25505	CWE-306/321	critical	**caught**[‡]	**caught**
CVE-2026-27602	CWE-78	high	missed	missed
CVE-2026-27825	CWE-22/73	critical	missed	missed
CVE-2026-28490	CWE-203/327	high	missed	**caught**
CVE-2026-29065	CWE-22	high	missed	missed
CVE-2026-32247	CWE-943	high	missed	missed
CVE-2026-33154	CWE-94/1336	high	missed	**caught**
CCDR (per-commit detection rate)			**13%**	
CDR (cumulative detection rate)			**27%**	
Detection gap			**13%**	

[†] CVE-2025-46724: B307 fires on `eval()` but commits are titled "fix: harden eval()"—developers suppress the alert as a security-fix false positive.

[‡] CVE-2026-25505: B105 flags a hardcoded JWT key (CWE-321, LOW) but misses 200+ endpoints lacking the `RequirePermissionIfAuthEnabled` dependency (CWE-306)—no rule exists for absent guards.

Key result: CCDR = 13% (2/15 CVEs caught per-commit). 87% of multi-commit chains are completely invisible to Semgrep and Bandit at the per-commit level. Critically, *both* per-commit detections are qualitatively poor: (1) CVE-2025-46724's `eval()` alert appears on commits framed as security fixes, leading developers to suppress it; (2) CVE-2026-25505's hardcoded-key alert is LOW severity and detects only the minor CWE-321 component, while completely missing the CWE-306 issue (200+ unprotected API endpoints). This suggests that the practical CCDR is effectively lower than the nominal 13%.

CDR = 27% (4/15); even with the full codebase present, 73% of chains remain invisible, primarily those using custom wrappers (CWE-78, CWE-22, CWE-943), missing-guard patterns (CWE-22, CWE-73), or patterns outside SAST rule coverage (CWE-502, CWE-693, CWE-79).

5.4 Failure Mode Analysis

We identify three root causes explaining why 13 of 15 CVEs evade per-commit detection:

(1) Custom wrapper opacity. 4 of 15 CVEs route dangerous operations through project-internal helper functions (e.g., `exec_cmd()` in Modoboa, `session.run()` in Graphiti, template evaluator in dynaconf). Semgrep and Bandit match against known dangerous API names; custom wrappers are opaque to these name-pattern rules. None of these CVEs are caught in either per-commit or cumulative mode.

(2) Absent-guard invisibility. 3 of 15 CVEs involve endpoints or functions that lack an authentication or validation check (CWE-306: 1 CVE; CWE-22 path traversal guards: 2 CVEs). No SAST tool fires on the absence of a dependency or decorator—only on the presence of a dangerous call. These CVEs are missed in both modes.

(3) Temporal source-sink separation. In all 15 cases, the taint source and its eventual dangerous sink were introduced in distinct commits separated by 21 to 1,342 days. Even a hypothetical SAST tool with perfect whole-codebase taint analysis would require cross-commit state to observe the flow—a capability no current snapshot-based tool provides.

6 DISCUSSION

6.1 Limitations

Single annotator. All chains were annotated by the author. Three CVEs were independently re-annotated by the same author in a time-separated blind condition; all three produced consistent chain descriptions, supporting annotation reliability. We plan a formal inter-annotator agreement study in future work.

Two SAST tools. We evaluated Semgrep and Bandit. CodeQL [3] performs deeper inter-procedural analysis and may achieve higher CDR for some patterns (particularly custom wrapper opacity); we leave this to future work.

Python only. The benchmark is Python-specific. Multi-commit chains exist in other ecosystems; extending to JavaScript/TypeScript and Java is planned.

CVE availability. All 15 CVEs have public advisories and accessible commit history. Some older vulnerabilities may have limited history depth due to repository restructuring.

6.2 Use Cases

CrossCommitVuln-Bench can serve as: (1) an evaluation benchmark for cross-commit detection tools; (2) training data for commit-sequence anomaly models; (3) a motivating dataset for CI/CD systems that maintain persistent security state across commits (as opposed to per-PR scans).

As a proof-of-concept, POSTURA [4]—a graph-based cross-commit analysis system that maintains a persistent Neo4j threat graph across commits—detected 3 of 5 spike CVEs from this dataset (60%) using taint analysis and chain discovery rules unavailable to per-commit SAST.

7 DATA AVAILABILITY

The complete dataset, annotation schema, evaluation scripts, and replication instructions are available at:

- **Dataset**: https://github.com/motornomad/crosscommitvuln-bench (Zenodo DOI: https://doi.org/10.5281/zenodo.19338596, CC-BY-4.0)
- **Evaluation scripts**: MIT License (same repository)
- **Replication**: `make all` in the repository root runs the full pipeline (validate → baselines → metrics) in Docker

REFERENCES

[1] Guru Prasad Bhandari, Amara Naseer, and Leon Moonen. 2021. CVEfixes: Automated Collection of Vulnerabilities and Their Fixes from Open-Source Software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*. 30–39. https://doi.org/10.1145/3475960.3475985

[2] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*. 508–512. https://doi.org/10.1145/3379597.3387501

[3] GitHub Inc. 2024. CodeQL: Semantic Code Analysis Engine. https://codeql.github.com.

[4] Arunabh Majumdar. 2026. POSTURA: Graph-Based Cross-Commit Security Analysis. https://pypi.org/project/postura/ (PyPI) and https://github.com/motornomad/postura (source).

[5] NSA Center for Assured Software. 2017. *Juliet Test Suite v1.3 for C/C++ and Java*. Technical Report. National Security Agency.

[6] PyCQA. 2024. Bandit: A Tool Designed to Find Common Security Issues in Python Code. https://github.com/PyCQA/bandit.

[7] Semgrep Team. 2024. Semgrep: Fast, Open-Source, Static Analysis for Finding Bugs and Enforcing Code Standards. https://semgrep.dev.

[8] Dave Wichers and Jim Manico. 2021. OWASP Benchmark: A Free and Open Test Suite for Evaluating the Accuracy of Software Vulnerability Detection Tools. https://owasp.org/www-project-benchmark/.

[9] Yunhui Zheng, Saurabh Pujar, Benjamin Lewis, Luca Buratti, Edward Epstein, Bo Yang, Jim Laredo, Alessandro Morari, and Zhong Su. 2021. D2A: A Dataset Built for AI-Based Vulnerability Detection Methods Using Differential Analysis. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 111–120. https://doi.org/10.1109/ICSE-SEIP52600.2021.00018

[10] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 32.