Daily Scholar Papers Report — 2026-05-02

2026-05-02

Daily Scholar Papers Report — 2026-05-02

Download PDF

Window covered: 2026-05-01 → 2026-05-02 (Google Scholar alerts + user-curated self-emails, last 24 h)

Executive Summary

A quiet alert window today: only one fresh paper makes it through to deep-read, and it is a **user-flagged self-email** rather than a Scholar alert. The day's Outstanding paper, **VulWeaver** (Cao, Chen, Hu, Bihuan Chen and collaborators at Fudan University and Huawei), targets the same Java vulnerability-detection terrain that **Phoenix** mapped yesterday but takes a different route: instead of synthesising Gherkin behavioural contracts, VulWeaver builds an **enhanced Unified Dependency Graph (UDG)** by neuro-symbolic graph repair, extracts a **holistic vulnerability context** (explicit slice plus implicit usage / definition / declaration context), and then drives a **meta-prompted, CWE-specific four-step deduction** with self-consistency majority voting. On their newly curated PrimeVul4J dataset (1,620 deduplicated samples spanning 41 CWEs, chronologically split into 80/10/10), VulWeaver attains *F1 = 0.75*, *Pair-Correct = 0.58*, and *VP-S = 0.58* with DeepSeek-V3.2, beating the strongest learning-based / LLM-based / agent-based baselines by **23 % / 15 % / 60 %** in F1 and pushing pairwise discriminative power **164 %** above the best baseline. The single most reusable engineering result is the **three-tier implicit-context recovery** — usage, definition, and declaration contexts each separately resolved beyond the explicit slice — which closes the false-positive gap that doomed the same authors' prior work on inter-procedural slicing.

Methodologically, VulWeaver pairs nicely with yesterday's **Phoenix**: both papers replace a single free-form LLM judgement with a structured intermediate representation and an explicit reasoning rubric, but at different layers — Phoenix at the *contract* layer (Gherkin Given-When-Then), VulWeaver at the *context* layer (UDG + holistic slice). Read together with **AnalysisAgent** (Bouzenia, Cadar, Pradel — yesterday's Outstanding) the picture for 2026-Q2 is consistent: structured rubrics beat free-form judgement by double-digit F1 / verified-success points, and *isomorphism between problem space and solution space* (Phoenix's phrase) is becoming the design principle. VulWeaver also delivers the most concrete real-world evidence in this trio: **26 true vulnerabilities** flagged across 9 real Java projects, **15 confirmed by developers**, and **5 CVE identifiers** assigned — plus **40 confirmed vulnerabilities** in an industrial Huawei deployment.

The four Scholar alerts that arrived inside today's 24 h window had already been processed in yesterday's report (AnalysisAgent, Phoenix, RealBench, HEAP LOCALIZATION); they are not re-deep-read here, see the 2026-05-01 report for the full treatment.

Outstanding: 1 · **Keep:** 0 · **Borderline High-Priority:** 0

The full analysis follows.

Highlighted Papers

#	Title	Authors	Venue	Link
4.1	VulWeaver: Weaving Broken Semantics for Grounded Vulnerability Detection	Yiheng Cao, Yihao Chen, Xin Hu, Bihuan Chen, Jiayi Deng, Zhuotong Zhou, Susheng Wu, Yiheng Huang, Xueying Du, Xingman Chen, Miaohua Li, Xin Peng	arXiv 2604.10767 [cs.SE] (preprint, ACM-formatted)	arXiv

Outstanding Papers (Deep-Read)

4.1 · VULN-AGENT · [USER-PICK] VulWeaver lifts PrimeVul4J F1 to 0.75 and VP-S to 0.58 (164% above the best baseline) by weaving a neuro-symbolic UDG, three-tier implicit context, and CWE-specific meta-prompting — 5 CVEs assigned in real Java projects□□□

4.1 VulWeaver: Weaving Broken Semantics for Grounded Vulnerability Detection

arXiv:2604.10767

Title: VulWeaver: Weaving Broken Semantics for Grounded Vulnerability Detection **Authors:** Yiheng Cao, Yihao Chen, Xin Hu, Bihuan Chen* (corresponding), Jiayi Deng, Zhuotong Zhou, Susheng Wu, Yiheng Huang, Xueying Du, Xin Peng (Fudan University, College of Computer Science and Artificial Intelligence); Xingman Chen, Miaohua Li (Huawei Technologies Co., Ltd). **Venue:** arXiv:2604.10767v1 [cs.SE] — preprint, ACM-formatted ("Manuscript submitted to ACM"), submitted 12 Apr 2026. **Year:** 2026 **Link:** https://arxiv.org/abs/2604.10767 **License:** ACM (publication rights licensed to ACM, copyright held by the author(s)). Original figures not embedded; pipeline recreated in Mermaid below. **Source:** User-curated forward (`Paper-read`, 2026-05-01) — STEP 2b USER-PICK.

Objective Summary

- **Problem.** LLM-based source-level vulnerability detection has converged on a recipe of inter-procedural slicing followed by chain-of-thought prompting (LLMxCPG, IRIS, VulInstruct, etc.), but the recipe stalls on three concrete limitations: (1) the program representations on which slicing depends are *inaccurate* — Joern and similar tools deliberately favour scalability over precision, leaving over-approximations (spurious polymorphic edges) and under-approximations (missing reflection edges); (2) the extracted vulnerability context is *incomplete* because slicing follows explicit control- and data-dependency edges only, missing structurally disjoint but semantically essential context such as global variable definitions, top-level class declarations, and import statements; (3) LLM reasoning over the resulting context is *opaque and ungrounded*, captured by Limitation 3 of §1, where the model often relies on lexical cues over data-flow semantics.
- **Approach.** VulWeaver is a three-stage neuro-symbolic pipeline that addresses each limitation.
    - **Stage 1 — Unified Dependency Graph Construction.** Joern's CPG is decomposed and only its semantic representations (CFG, DDG, CG) are unified into an original

UDG `G_o = (V_o, E_o)` with statement-level nodes and edges typed `τ ∈ {control flow, data dependency, call}`. The AST is omitted to avoid graph blow-up. The original UDG is then enhanced into `G_e` by a *neuro-symbolic enhancement* combining deterministic syntactic rules with LLM-based semantic inference (e.g. recovering missing polymorphic/reflective call edges, marking globals as global nodes).

- **Stage 2 — Holistic Vulnerability Context Extraction.** For every sensitive invocation (predefined sensitive APIs from a 102-CWE-type knowledge base or user-supplied dangerous functions) VulWeaver extracts an **explicit context** `C_e` by backward-and-forward slicing along `G_e`'s explicit edges, and an **implicit context** `C_i = C_use ∪ C_def ∪ C_decl` recovered through three resolvers — Usage, Definition, and Declaration context resolution — that recover unresolved variable usages, missing definitions for unresolved variables (`V_unresolved = V_use \ V_def`), and structural global declarations (imports, package, top-level class) respectively. The final holistic context is `C = C_e ∪ C_i`. The motivating CVE-2020-26282 example (Fig. 2 of the paper) shows that without `C_decl` the LLM misclassifies the patched `MessageSanitizer.escape` because `ESCAPE_PATTERN`'s regex initialisation sits structurally outside the data-flow slice.
- **Stage 3 — Context-Aware LLM Reasoning.** Each `(API, CWE)` pair gets a meta-prompt with a CWE-specific four-step guideline: (i) Contextual Flow Understanding, (ii) Trigger Condition Verification against `vuln_patterns_cwe`, (iii) Defense Assessment against `defense_knowledge_cwe`, (iv) Evidence-Driven Verdict Synthesis. The LLM is queried `N` times independently and the verdict aggregated by **majority voting** for self-consistency; the paper finds `N = 5` saturates F1 at 0.754 (Fig. 8).

- **Knowledge base.** A 102-CWE-type knowledge base mapping sensitive APIs to vulnerability semantics and remediation heuristics. Initialised by an internal industrial tool in 137 s, refined by two senior security experts (5+ years experience). Released with the paper.
- **Implementation.** 7 K lines of Python; DeepSeek-V3.2 as the inference model; Java target language, generalised to C/C++ in §4 with minimal adaptation.

Headline Numbers (verbatim where possible from §4 / Table 2)

- **PrimeVul4J test set (DeepSeek-V3.2 inference):** Precision 0.81, Recall 0.70, **F1 = 0.75**, Pair-Correct **P-C = 0.58**, P-R = 0.00, **VP-S = 0.58**.
- **Whole PrimeVul4J:** F1 = 0.77, VP-S = 0.58 — virtually identical to the test set, indicating no data-scale bias.
- **Improvements over baselines (test set, F1):** +23 % over learning-based DeepDFA (0.61), +60 % over agent-based VulTrial (0.47), +15 % over the strongest LLM-based VulInstruct (0.65); +164 % VP-S over the strongest pairwise baseline VulRAG (VP-S 0.22 → 0.58); P-R = 0.00 vs P-R 0.20 for VulInstruct.
- **Cross-language generalisation.** On C/C++ PrimeVul, F1 = 0.78 with minimal adaptation.
- **CWE-type breakdown (Table 3).** Highest F1 across all six CWE-664 / 707 / 284 / 693 / 691 / 703 categories; tied for best on the two categories with `#S = 2`.
- **Token-tax robustness.** F1 stays within ±0.02 across context-length buckets `[0, 5k)`, `[5k, 10k)`, `[10k, 20k)`, `[20k, ∞)` — VulWeaver's structured context extraction largely defuses the lost-in-the-middle phenomenon.

- **Parameter sensitivity.** F1 saturates at `N = 5` self-consistency rounds (F1 = 0.754, VP-S = 0.58); further rounds give negligible gains.
- **Practical usefulness. 26 true vulnerabilities** detected across 9 real-world Java projects, **15 confirmed by developers**, **5 CVE identifiers assigned**. In industrial Huawei deployment, **40 confirmed vulnerabilities** in an internal repository.
- **Engineering scale.** 7 K lines of Python; 256 GB RAM + Nvidia A100 (80 GB) host; knowledge-base initialisation 137 s.

Ablations (Table 2, lower block)

- **w/o `G_e` (skip neuro-symbolic UDG enhancement, use Joern's raw graph):** F1 0.68 (-0.07), VP-S 0.42 (-0.16) on the test set. Confirms Limitation 1 quantitatively — neuro-symbolic graph repair is worth ~7 F1 points.
- **w/o `C_i` (skip implicit context, use explicit slice only):** F1 0.62 (-0.13), VP-S 0.34 (-0.24). The implicit-context recovery is the single largest contributor to performance — bigger than UDG enhancement.
- **w/o `C` (skip holistic context entirely, function-level only):** F1 0.60 (-0.15), VP-S 0.37 (-0.21). Confirms Limitation 2.
- **w/ CoT (replace meta-prompting by chain-of-thought):** F1 0.61 (-0.14), VP-S 0.26 (-0.32). Confirms Limitation 3 — meta-prompting carries +0.32 VP-S over CoT given identical context.
- **w/ Qwen2.5-32B (smaller open-source backbone):** F1 0.74, VP-S 0.51 — only -0.01 F1 vs DeepSeek-V3.2, confirming that the structured pipeline, not model capacity, drives the gain.

Robustness Pilot (Table 1)

Five vanilla LLMs (GPT-5-mini, Claude-Haiku-4.5, DeepSeek-V3.2, Gemini-2.5-flash, Grok-Code-Fast) tested under identifier-perturbation adversarial prompting (`non_vulnerable` / `vulnerable` prefixes injected). CoT prompting collapses: F1 falls 11–39 %, VP-S falls 44–1,050 %, even turning negative for Claude-Haiku-4.5 (-0.09) and Gemini-2.5-flash (-0.38). Meta-prompting is robust: F1 deltas 0–6 %, VP-S deltas within 16 %. *Empirical evidence that structured guideline-driven decomposition shields against superficial-lexical-cue overfitting* — the same conclusion AnalysisAgent reaches at the orchestration layer.

Methodological Reusable Ideas

1. **Three-tier implicit context recovery (Usage / Definition / Declaration).** The cleanest single contribution. The Definition resolver explicitly handles the `V_unresolved = V_use \ V_def` case by checking incoming DDG edges and falling back to global-definition slicing; the Declaration resolver pulls structural top-level statements (imports, package, top-level class) that are unreachable via standard slicing but essential for type resolution. Worth lifting as a standard slicer plug-in for any LLM-based detector that consumes function-body slices.
2. **Neuro-symbolic UDG enhancement.** Stage 1's recipe — start from Joern's CPG, drop the AST, unify the semantic graphs, then repair under-approximations (reflection / polymorphism) and over-approximations using LLM-based edge classification gated by deterministic syntactic constraints — is a transferable pattern for any analysis pipeline that depends on Joern / CodeQL / Soot.
3. **CWE-specific meta-prompt template.** The four-step guideline (Contextual Flow → Trigger Condition vs `vuln_patterns_cwe` → Defense Assessment vs `defense_knowledge_cwe` →

Evidence-Driven Verdict Synthesis), parameterised per CWE, enforces taint-style reachability + sanitiser-vs-bypass logic explicitly. Reusable as a prompt template for any CWE-typed detection system.

4. **Self-consistency saturation analysis.** Reporting F1 vs `N` (Fig. 8) makes the cost-vs-quality tradeoff explicit; `N = 5` is the recommended setting.
5. **Pairwise VP-S as the discriminative metric.** P-C minus P-R penalises models that flip predictions across vulnerable / patched pairs. With P-R = 0.00 VulWeaver demonstrates that structured prompting can drive *zero* semantic-reversal failures — a signal we should adopt in our own evaluation rubrics.

Pipeline Recreation (Mermaid)

```
flowchart LR
  Repo[Target repo] --> Joern[Joern CPG]
  Joern --> Go[Original UDG G_o<br/>CFG ∪ DDG ∪ CG]
  Go --> Enh[Neuro-Symbolic<br/>UDG Enhancement]
  Enh --> Ge[Enhanced UDG G_e]
  Ge --> SI[Sensitive invocation<br/>(API, CWE)]
  SI --> Ce[Explicit context C_e<br/>backward / forward slice]
  Ce --> CR[Three-tier resolver]
  CR --> Cu[C_use]
  CR --> Cd[C_def]
  CR --> Cl[C_decl]
  Cu --> Ci[Implicit context C_i]
  Cd --> Ci
  Cl --> Ci
  Ce --> CTX[C = C_e ∪ C_i]
  Ci --> CTX
  CTX --> MP[Meta-prompt<br/>4-step guideline + CWE knowledge]
  MP --> LLM[LLM × N rounds]
  LLM --> MV[Majority voting<br/>self-consistency]
  MV --> V[Verdict<br/>(vulnerable / non-vulnerable)]
```

Critique / Limitations

- The PrimeVul4J construction (CrossVul + ReposVul + CVEFixes, denoised under PrimeVul's labelling, normalised+deduplicated, chronologically split) is itself a contribution — but it borrows the labelling pipeline that Phoenix critiques in §2 of the same week's preprint. The "Double Standard Problem" Phoenix surfaces on the C/C++ PrimeVul (gen_assignment, FractionalAvgPoolGrad) is not addressed here for Java; whether PrimeVul4J carries analogous near-duplicate cross-CVE label inversions is not reported.
- Source-level static analysis caveat: the paper's own in-depth analysis (§4) attributes residual false negatives to opaque Spring-Boot dependency-injection annotations that source-only Joern cannot resolve. This is a fundamental limit of the substrate, not the pipeline.
- The 7 K-LOC + DeepSeek-V3.2 + 102-CWE knowledge base, while released, makes a clean baseline reproduction expensive. The Qwen2.5-32B ablation result (-0.01 F1) is the easiest entry point.

- Industrial deployment numbers (40 confirmed vulnerabilities) are reported without precision/recall, only count.

Cross-Paper Synthesis

This week's three vulnerability-detection-and-analysis papers — **AnalysisAgent** (Mon), **Phoenix** (Mon), and **VulWeaver** (today) — converge on a single architectural thesis: *replace free-form LLM judgement with a structured intermediate representation that the LLM verifies against an explicit rubric*, but they place the structure at three different layers.

Layer	Paper	Structured intermediate	Rubric / verification
Orchestration	AnalysisAgent	Reproducible Docker env + tool-specific artefacts (KLEE `klee-out/`, AFL++ `queue/`, ...)	Three-tier evidence rubric (structural + project-reference + semantic)
Context	VulWeaver	Enhanced UDG `G_e` + holistic context `C = C_e u C_use u C_def u C_decl`	Four-step CWE guideline (Contextual Flow → Trigger → Defense → Verdict) with self-consistency `N = 5`
Contract	Phoenix	Gherkin Given-When-Then specification synthesised from the vulnerable / patched pair	Strict compliance check by Contract Judge

The three papers also agree on a **failure mode of unstructured LLM-as-judge**: AnalysisAgent reports the verified-vs-self-validated gap (RAG-Agent: 9 % verified at 73 % self-reported); Phoenix's Table 1 shows that CoT prompting collapses 0.32 VP-S under identifier-perturbation adversarial prompting; VulWeaver's `w/ CoT` ablation shows the same collapse (0.58 → 0.26 VP-S) under identical context. **Structured rubrics + explicit intermediate representations are now the cheapest way to claw back ~30 VP-S / verified-success points without scaling the model.**

A second thread is **smaller-models-with-structure-beat-larger-models-without**: Phoenix's 7-14 B Qwen variants beat DeepSeek-V3 671 B by 0.16 F1; VulWeaver's `w/ Qwen2.5-32B` ablation drops only 0.01 F1 vs DeepSeek-V3.2; AnalysisAgent's verified success on Gemini-3-Flash exceeds GPT-5-mini under all three baseline architectures. The implication for our own work: invest the next iteration cycle in *intermediate-representation engineering* over *backend-model upgrade*.

A third thread, less universal but worth flagging, is **the increasing use of paired pre-/post-fix evaluation** as the discriminative ground truth: VulWeaver's VP-S, Phoenix's Pair-Correct, RealBench's Architecture@k against ground-truth UML. Evaluating a single positive label is becoming a poor signal once the absolute F1 ceiling exceeds 0.7; pairwise / rubric-graded metrics expose the residual semantic-reversal failures that single-instance F1 hides.

Writing & Rationale Insights

VulWeaver's prose is unusually disciplined for a 2026 LLM-detection preprint: the introduction's three-Limitation framing is mirrored by the three-stage Approach and the three-axis Ablation, so the reader can trace each problem statement to a specific architectural choice and a specific ablation row. The **Example 3.2** worked through CVE-2020-26282 — showing exactly which lines `C_e`, `C_use`, `C_def`, `C_decl` recover and how the LLM's verdict flips between the slice-only and the holistic-context inputs — does the heaviest lifting; it converts an otherwise abstract three-tier resolver into something the reader can reason about line-by-line. We should adopt this *worked-example-anchors-the-method* convention in our own writing whenever a method has more than two stages.

The robustness pilot in §2 (Table 1) is a particularly cheap and convincing experimental design: identifier-perturbation (`non_vulnerable` / `vulnerable` prefix injection) is a one-line adversarial transformation that surfaces lexical-cue dependence dramatically. We can lift this exact perturbation as a default robustness check in any code-classification eval — it costs nothing and exposes failure modes that standard test-set F1 hides.

Finally, on rationale framing, both VulWeaver and Phoenix this week argue that *the structure outside the LLM is doing the work*, but they make the argument in opposite directions: Phoenix argues by ablation (remove the Gherkin contract, F1 drops 0.09–0.35), VulWeaver argues by ablation + adversarial perturbation (remove the meta-prompt, VP-S drops 0.32). The combination is rhetorically stronger than either alone, and is a useful pattern when defending a structured-prompting contribution against the "the LLM did it all" critique.