Can Agentic Code Review Catch Vulnerabilities Before They Are Merged?

Yihan Yang[*], **Chenyuan Yang**[*], **Zijie Zhao**, **Anh Nguyen Hoang**[†], **Ziqi Zhang**, **Weidong Wang**, **Zhiheng Lyu**, **Lingming Zhang**
University of Illinois Urbana-Champaign
{yihany5, cy54, lingming}@illinois.edu

Abstract

LLM agents have been increasingly adopted for security-focused code review, but existing benchmarks mostly derive the reviewed changes from their later fixes instead of the real-world bug-introducing commits. In this paper, we present **SecPRBench**, the first comprehensive benchmark for evaluating agentic security review on validated real-world code changes. More specifically, we have built an automated agentic pipeline to autonomously harvest, build, and validate massive real-world bug-introducing commits for popular software systems. To ensure precise evaluation, we also adopt LLM-as-a-judge to credit the findings matching ground-truth vulnerabilities. To date, we have collected 265 tasks from nine critical systems, including Firefox, V8, and OpenSSL with millions of lines of code. Across eight frontier models, recall ranges from 32.5% to 78.1%, with Grok 4.6 and GPT-6 sol ranking at the top. Our study also shows that most misses are not simply oversights: in 68% of them, the agent reports on a file that holds the flaw but argues for a different mechanism. Security-focused harnesses also behave differently: OpenAI's Codex Security lowers recall by 8.3 points, mostly by abstaining, while Anthropic's Defending Code harness raises it by 7.7 points, at a higher cost.

github.com/secprbench/secprbench
secprbench.github.io

1 Introduction

Anthropic, OpenAI, and GitHub now offer agents that review code changes, such as pull requests (PRs), for vulnerabilities before merge (Anthropic, 2026b; OpenAI, 2025; GitHub, 2026). A vulnerability missed at review can persist for years (Yang et al., 2025). Consider a Firefox sandbox escape in which a compromised child process could make the privileged parent return an unintentionally powerful handle over inter-process communication (IPC) (CVE-2025-2857; Mozilla Foundation, 2025). Catching it at review requires connecting the IPC code to Windows handle semantics and the trust boundary between processes. We ask: *Can today's agents catch vulnerabilities before merge?*

A diff shows a reviewer where to start looking but not what is wrong, so a security review is only as useful as its diagnosis. Unfortunately, existing benchmarks do not measure diagnosis on real changes (Table 1). Earlier detection datasets score a binary label on single functions (Ding et al., 2025; Yildiz et al., 2025), which models often get right for the wrong reasons (Ullah et al., 2024). Reproduction and exploitation benchmarks start from a known vulnerability and score an executable proof-of-concept (PoC) or exploit (Wang et al., 2026b; Lee et al., 2026; Wang et al., 2026a; Lee and Brumley, 2026). Code-review benchmarks also do not target security vulnerabilities (Zeng et al., 2026; Zhang et al., 2026b). Security PR-review benchmarks come closest, and MalPR-Bench scores whether the reviewer identifies the target (Chen et al., 2026). However, most of their vulnerable PRs revert or partially apply a later fix, which centers each diff on the repaired code, instead of using the real vulnerability-introducing commit (Melo et al., 2026; Chen et al., 2026).

We introduce SecPRBench, a benchmark for *pre-merge security review* that aims to evaluate modern agentic code review on real-world bug-introducing commits. Each task gives an agent the diff of a real historical change that introduced a vulnerability, its commit message, and the repository, but no description of the flaw; the agent reports *findings*. A finding earns credit only if it explains the root cause of the task's *target* vulnerability in the reviewed code; in Figure 1, a bounds warning on the right call earns none. SecPRBench contains 265 buggy tasks from nine widely deployed C, C++, and Rust

[*]Equal contribution.
[†]Quantum AI & Cyber Security Institute, FPT Corporation; work done during an internship at UIUC.

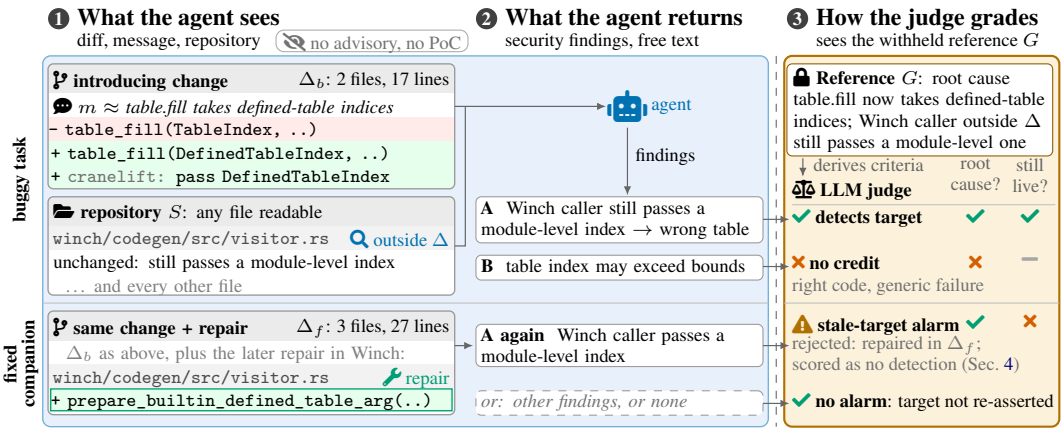


Figure 1: **One SecPRBench case** (Wasmtime CVE-2026-34946; diff and message paraphrased, findings illustrative). The flaw depends on unchanged code outside the diff; the LLM judge credits a finding only if it explains the target's root cause and that cause is still live in the reviewed code.

projects, including Firefox, V8, and OpenSSL with millions of lines of code. Each buggy task has a *fixed companion* that adds the later repair, on which the target should no longer be reported.

Building such pairs is challenging as advisories typically name the fix, not the introducing change, and automatic attribution can be unreliable (Lu et al., 2026). We have built an automated agentic pipeline to trace each bug's introducing change and compose it with the later fix. Moreover, our pipeline also validates the pairs automatically, mostly by PoC replay on the parent, introducing, and fixed versions. Most cases pass mechanical validation checks based on PoC execution; the other task instances require human adjudication. For task evaluation, since no oracle checks free text, a large language model (LLM) judge, Claude Opus 5.5 (high), derives target-specific criteria from a reference built from public security records and checks each finding against them and the reviewed code.

Across eight frontier models and two security-focused harnesses, three findings stand out. **Strong agents tie on easy targets but diverge on contested ones.** Grok 4.6, GPT-6 sol, GLM 5.3, and Opus 5.5 each find about three in four targets and are statistically tied overall. On easy targets, which nearly all other agents find, every agent but MiniMax M3 finds at least nine in ten. On contested targets, which few other agents find, the top four still find about half, while the other four find at most one in five. Even combined, the eight models leave 23 targets unfound. **Most misses are not oversights.** In 68% of misses, a finding lands in a file that holds the flaw, often in the very function, but argues a mechanism the judge rejects. Scoring by location or by a binary vulnerable/safe verdict would count these as detections; MalPR-Bench similarly finds that reviewers often block a PR without identifying its flaw (Chen et al., 2026). **Security-focused harnesses move recall in both directions.** With each backend model fixed, OpenAI's Codex Security plugin, run as a static diff scan (OpenAI, 2026f), lowers recall by 8.3 points relative to the same model in the Codex CLI, mostly by abstaining; Anthropic's Defending Code reference harness (Anthropic, 2026e) raises it by 7.7 points on the tasks it completed, at higher cost and report volume. On repaired changes, the judge never rules that the removed target is still present (§ 5.4).

The paper makes three main contributions:

- **Benchmark and the construction pipeline.** SecPRBench, the first comprehensive benchmark for evaluating agentic security review on validated real-world code changes, with 265 cases built by an automated, repeatable pipeline that turns fixes into validated pairs of introducing and repaired changes (§ 3).
- **Grading.** A criteria-first judge that credits only root-cause explanations of a target present in the reviewed code, separating correct diagnosis from findings that merely land in the right place, with each verdict's rationale recorded (§ 4).
- **Analysis.** Evaluation of eight frontier models and two security-focused harnesses showing where agents diverge, why they miss, and whether harnesses and training cutoffs change recall, with reports, verdicts, and analysis code released (§ 5).

Benchmark	Tasks	Task input			Validation		Scoring	
		Real change	No hint	Full repo	Paired control	PoC-validated	Scored by	Root cause
VULNERABILITY DETECTION: LABEL								
PrimeVul (Ding et al., 2025)	6,968	✗	✓	✗	✓	✗	Label	✗
JitVul (Yildiz et al., 2025)	1,758	◐	✓	◐	✓	✗	Label	✗
REPRODUCTION AND EXPLOITATION: POC, EXPLOIT, OR PATCH								
SEC-bench (Lee et al., 2025)	200	✗	✗	✓	✗	✓	Exec.	✗
CyberGym (Wang et al., 2026b)	1,507	✗	◐	✓	✗	✓	Exec.	✗
CyberGym-E2E (Shi et al., 2026)	920	✗	✓	✓	✗	✓	Exec.	✗
SEC-bench Pro (Lee et al., 2026)	344	✗	◐	✓	✗	✓	Exec. + LLM	✗
ExploitGym (Wang et al., 2026a)	898	✗	✗	✓	✗	✓	Exec. + LLM	✗
ExploitBench (Lee and Brumley, 2026)	41	✗	✗	✓	✗	✓	Exec.	✗
CODE REVIEW: COMMENTS								
c-CRAB (Zhang et al., 2026b)	184	✓	✓	✓	✗	✗	Tests	✗
SECURITY REVIEW OF PULL REQUESTS: VERDICT OR REPORT								
SEVRA-Bench (Melo et al., 2026)	∼1,000	◐	✓	✓	✗	✗	LLM	◐
MalPR-Bench (Chen et al., 2026)	89 + 50	◐	✓	✓	✓	◐	LLM, human	✓
SecPRBench (ours)	**265 + 265**	✓	✓	✓	✓	◐	LLM	✓

Symbols: ✓ yes, ◐ partial, ✗ no. *Tasks*: vulnerable + control where both are reported; JitVul counts functions from 879 CVEs. *Real change*: from project history; ◐ for fix-derived changes (reverted or partial patches) or functions from introducing commits. *No hint*: ◐ if hints vary by level (CyberGym) or target metadata is given (SEC-bench Pro). *Paired control*: fixed or benign controls, not always one per vulnerable input. *PoC-validated*: ◐ if only some cases are (MalPR-Bench: twelve discovery cases; SecPRBench: § 3.3).

Table 1: **How SecPRBench differs from related benchmarks.** Only SecPRBench uses complete vulnerability-introducing changes (no hint, full repo) and pairs each with its repaired version.

2 Related Work

Reviewing code changes. Code-review benchmarks range from diff hunks (Li et al., 2022) to whole PRs with repository context (Zeng et al., 2026; Zhang et al., 2026a; Kumar, 2026; Zhang et al., 2026b), but their ground truth is review comments across all concerns. Yu et al. (2024) compare LLMs with static analyzers on security defects from OpenStack and Qt code reviews, and Google's Big Sleep, seeded with a recent SQLite commit, found a loosely related vulnerability before release (Big Sleep team, 2024). SEVRA-Bench wraps reverted historical fixes in persuasive PR descriptions (Melo et al., 2026), and Alexopoulos et al. (2026) show that such framing biases LLM security review. MalPR-Bench, the closest design, pairs malicious PRs with benign controls and scores whether a reviewer identifies a rubric-specified target, but derives most PRs from later fixes (Chen et al., 2026). SecPRBench instead presents the original change that accidentally introduced each vulnerability and was merged before the flaw was reported (Table 1).

Vulnerability detection. PrimeVul pairs vulnerable and patched functions and shows that detection drops sharply under realistic evaluation (Ding et al., 2025). Ullah et al. (2024) find that models sometimes reach correct labels through incorrect reasoning, and Risse et al. (2025) argue that function-level labels are ill-posed because vulnerability usually depends on calling context. JitVul pairs functions from introducing and fixing commits (Yildiz et al., 2025), and VulnGym evaluates repository-wide audits (Ji et al., 2026); SecPRBench keeps paired controls and repository access but scores a root-cause explanation of a whole change.

Security agent benchmarks. Cybench and BountyBench cover capture-the-flag and bug-bounty tasks (Zhang et al., 2025b,a), and the reproduction and exploitation benchmarks in Table 1 require a PoC for a known vulnerability (Lee et al., 2025; Wang et al., 2026b; Lee et al., 2026), hint-free discovery and patching (Shi et al., 2026), or an exploit (Wang et al., 2026a; Lee and Brumley, 2026). All score an executable artifact on a whole system, not an explanation of a change.

LLM judges. PaperBench and HealthBench judge free-form outputs against per-instance rubrics (Starace et al., 2025; Arora et al., 2025), but judges of code are swayed by response order, identifier names, and misleading comments (Jiang et al., 2026), and c-CRAB and ExploitBench use execution oracles (Zhang et al., 2026b; Lee and Brumley, 2026). Execution cannot score an explanation, so our judge derives criteria from a reference, not a human-written rubric, and checks both the target's

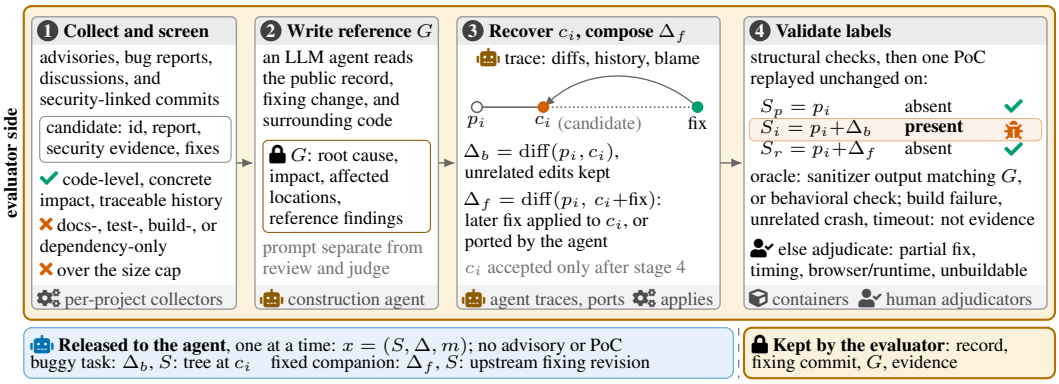


Figure 2: **Benchmark construction and validation.** From evidence withheld from evaluated agents, a construction agent writes G and recovers c_i; Δ_f adds the later fix (agent-ported if needed); labels are accepted by mechanical checks, including replay on S_p, S_i, S_r, or by human adjudication.

mechanism and its presence in the reviewed code.

3 Benchmark Design and Construction

Each case is built in four steps from evidence withheld from the evaluated agents (Figure 2).

3.1 Task Formulation

SecPRBench evaluates whether an agent can identify and explain a vulnerability that a proposed code change introduces. Each *case* centers on one *target* vulnerability and yields two review tasks: a *buggy task* built from the change that introduced the target (in *partial-fix* cases, a security-relevant change that failed to remove it; § 3.3), and a *fixed companion* in which the target is repaired.

Review input. The agent receives $x = (S, \Delta, m)$: a repository snapshot S, a diff Δ, and change text m from the commit message. For a buggy task with introducing commit c_i and parent p_i, S is the tree at c_i and $\Delta = \Delta_b = \mathrm{diff}(p_i, c_i)$. For a fixed companion, $\Delta = \Delta_f$ is the introducing change composed with the later repair, again relative to p_i; S is the upstream fixing revision, which also contains intervening upstream changes (see Limitations). The agent may read any code in S but sees no review discussion. Each task runs in an isolated container with no Git history and with network access limited to the model provider. The security record, the fixing commit (except code already in a companion's diff or snapshot), the reference, the validation evidence, and the task label stay with the evaluator (Appendix C).

Review output. The agent returns a possibly empty set of findings $P(x)$; each should locate the relevant code and explain the trigger, the unsafe operation, and the violated security condition. A finding detects the target if it explains the causal failure described in the reference G (§ 3.2) and that failure is present in the reviewed code (§ 4). A fixed companion controls only for its target: G need not cover every vulnerability in the change. This evaluation set has no training split.

3.2 Case Construction

Collection. Per-project collectors gather CVE-linked records and public project security reports, including issues without CVE identifiers, with their discussions and fixing commits (Appendix A). A candidate is eligible when the evidence establishes a concrete code-level security failure, a plausible trust boundary or untrusted-input path, and a causally connected fixing change. We keep fixes committed from January 2025 through June 2026, and both review diffs of a case must stay within 30 files and 2,000 added plus deleted lines.

Reference. An LLM construction agent, run mainly with Claude Sonnet 5, drafts the reference G from the public security record, the fixing change, and the surrounding code. G explains the root cause and impact and identifies the code locations needed to understand the causal failure. We do not audit every reference explanation by hand.

Introducing change. The same agent traces the affected code backward from the fix, using diffs, history search, and blame, to a candidate introducing commit c_i with parent p_i. Automatic attribution is unreliable: SZZ-style tools such as V-SZZ (Bao et al., 2022) blame the lines a fix touches but struggle to identify vulnerability-introducing commits (Lu et al., 2026), and agent-based tracing nar-

rows but does not close this gap on bug-introducing commits (Risse and Böhme, 2026). We therefore accept an attribution only after validation (as detailed in § 3.3), and the buggy task keeps the complete change Δ_b, including unrelated edits.

Fixed companion. We apply the later upstream fix to c_i and take the composed change Δ_f relative to p_i; when the fix does not apply cleanly, the construction agent ports it, instructed to keep both the introducing change's functionality and the fix's security-relevant behavior (Appendix C). Validation decides whether the result removes the target; patch application alone does not.

3.3 Task Label Validation

Validation checks that the introducing change makes the target reachable and that the repair in Δ_f removes it. After structural checks, 217 cases pass mechanical checks and 48 pass human adjudication; evidence scope varies by case (Appendix B).

PoC replay. We build three snapshots in project-specific containers: the parent S_p, the buggy tree $S_i = \text{apply}(p_i, \Delta_b)$, and the repaired tree $S_r = \text{apply}(p_i, \Delta_f)$, which differs from the companion snapshot that agents review. We reuse a public reproducer, adapted to the old revision or harness if needed, or generate one with AnyPoC (Zhao et al., 2026). We replay one PoC input and invocation unchanged on all three snapshots and require the target failure to be absent on S_p, present on S_i, and absent on S_r. The oracle uses sanitizer output matching the reference root cause for memory-safety failures and a behavioral check for supported logic bugs; build failures, unrelated crashes, and timeouts do not count as evidence.

Human adjudication. Adjudicators decide the cases that replay cannot settle. Seven partial-fix cases are a narrow exception to introducing-change tasks: the reviewed repair or hardening leaves the target reachable on both S_p and S_i, while replay records its removal on S_r. We retain them only when source evidence directly links the remaining target to the attempted repair in the changed code (Appendix B). Other cases need judgment beyond the default oracle, such as interpreting assertions, debugger traces, or races, or cannot be fully replayed on historical builds or platforms. Adjudication combines the security record, source comparison, fixing change, and available execution evidence; we retain the automated outcome alongside each decision.

3.4 Dataset Characteristics

Figure 3 summarizes the 265 cases by project, fixing date, and time from introduction to fix, and Appendix F.3 adds change sizes. Flaws reported and fixed separately remain separate cases even when they trace to the same change (Appendix A).

4 Protocol Grading

Agents answer a review task with free-text findings, which no execution oracle can check. Having validated the task labels (§ 3.3), we now describe how we decide whether an agent found the target.

4.1 What Counts as a Detection

We credit a finding when it explains the target's causal failure: what triggers it, which operation becomes unsafe, and which security condition is violated (§ 3.1). Wording and cited locations may differ from the reference; a sound account of the mechanism suffices, even without an exploit.

We do not credit findings that merely resemble the target. Some point at the right code but claim a different or generic failure; others describe a real bug next to the target. The hardest case is a finding that describes the target correctly but in code where it has already been fixed: agents often write nearly identical findings for a change and its fixed companion, naming the same function, operation, and vulnerability class. Text similarity, location overlap, and Common Weakness Enumeration (CWE) labels cannot separate these cases, so the judge must decide which failure a finding claims and whether it exists in the reviewed code.

In the Wasmtime task of Figure 1 (Appendix C), the runtime now expects a defined-table index, but one caller still passes a module-level index. A finding that ties this mismatch to an access on the wrong table counts as a detection even without a working sandbox escape; a generic bounds warning on the same call does not.

4.2 Criteria-First Judge

The judge is Claude Opus 5.5 (high), with tool use and web access disabled. It sees the reference root cause, reference findings, and relevant locations, together with the agent's findings and an excerpt of the reviewed change. For a fixed companion, it receives the original target's reference. It works through three steps in one session:

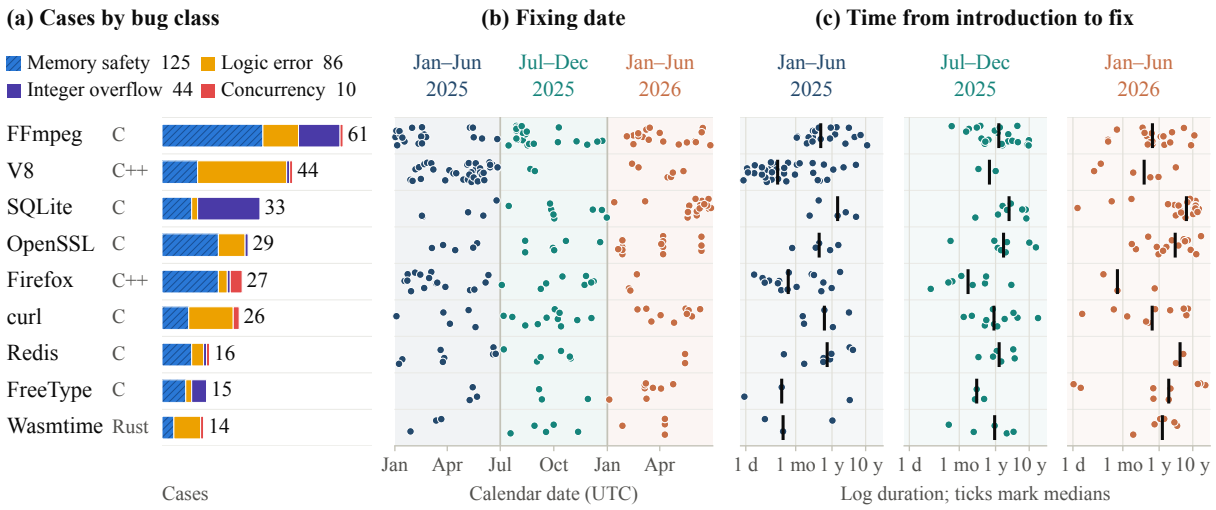


Figure 3: SecPRBench by project: (a) bug classes, (b) fixing dates, and (c) time from introduction to fix, grouped by fixing half-year. Each dot is a case; colors denote fixing half-years in (b) and (c). Duration panels share a log scale.

1. **Derive criteria.** It first writes down three to six conditions that separate the target from nearby bugs: the trigger, the unsafe operation, and the missing or incorrect guard, lifetime rule, bound, state transition, or trust boundary.
2. **Match the mechanism.** It then checks whether at least one finding meets these criteria, rejecting findings that only point at the same code or describe a neighboring bug.
3. **Check the reviewed code.** Finally, it confirms that the causal chain is still present in the reviewed change and that no repair, validation, bounds check, or synchronization breaks it; matching the historical reference alone does not show that the vulnerability is live.

The judge returns a single verdict per task, along with its criteria, confidence, and reasoning. Appendix D summarizes the prompt, and Appendix E gives implementation details, including how change excerpts are selected.

4.3 Judge Reliability

Across 2,143 nonempty buggy reports, Opus 5.5 and Sonnet 4.6 agree on 96.9% of verdicts. In a separate manual check, both judges matched our adjudications on all 25 randomly sampled agreement controls. Among 25 selected disagreements, our adjudications supported Opus in 21 cases and Sonnet in four. These checks demonstrate strong agreement between judges and support Opus on most audited disagreements. The diagnostic sample does not estimate overall judge accuracy. Appendix E gives the audit and grading examples.

4.4 Metrics

Target recall. Our main metric is the fraction of buggy tasks in which the judge credits a detection. With $d_i \in \{0, 1\}$ as the verdict for buggy task i, target recall is

$$R = \frac{1}{N} \sum_{i=1}^{N} d_i, \tag{1}$$

where $N = 265$ is the number of buggy tasks, the same for every agent. A task counts as missed if the agent returns nothing or its run ends in an error.

Hit@1. Hit@1 is the fraction of buggy tasks whose first finding detects the target; it models a developer who reads only the top finding. When the judge's rationale does not identify the credited finding, we report bounds.

Report volume and cost. We also count findings per task, since each one is something a developer must read. Findings that match no target are not necessarily false positives: the reference does not list every vulnerability in a change, and several findings may describe the same defect. Cost excludes construction and judging (Appendix F.1).

5 Evaluation

On the 265 buggy changes and their fixed companions, we ask how often agents find the introduced flaw (RQ1), why they miss (RQ2), whether a security-focused harness helps (RQ3), and how they perform on fixed versions (RQ4).

Agent	Detection			Misses		Fixed companion		Effort		
	Found	Recall	Hit@1	Rej.	Empty	Flagged	Δ Findings	Tok (k)	Est. cost ($)	Min.
Claude Code										
Grok 4.6	207	**78.1**	46.4–55.5	53	5	0	↓4%	43.3	3.20	12.6
GLM 5.3	198	74.7	**49.8–58.1**	66	1	0	↓6%	72.6	4.99	20.8
Claude Opus 5.5	196	74.0	47.2–55.1	63	6	0	↓13%	7.2	0.44	1.4
Claude Sonnet 5	157	59.2	41.1–47.9	100	8	0	↓6%	31.9	0.83	5.5
MiniMax M3	86	32.5	20.4–26.8	126	53	0	↑11%	38.5	3.33	11.5
Codex CLI										
GPT-6 sol	202	76.2	44.9–57.0	49	14	0	↓14%	15.7	0.60	3.3
GPT-5.6 sol	148	55.8	43.8–46.4	76	41	0	↓31%	9.3	0.74	2.7
GPT-6 luna	147	55.5	42.6–46.4	76	42	0	↓28%	14.8	0.02	2.7
GPT-5.6 sol + Codex Security	126	47.5	38.5–41.9	67	72	0	↓25%	31.3	3.74	7.3
Any agent (oracle)	242	91.3	—	—	—	—	—	—	—	—

Table 2: Results on 265 tasks (rates in %); darker is higher within a column. Hit@1: first finding detects the target; unresolved ranks count as none or all first. Rej./Empty: rejected/empty reports; Flagged: positive target verdicts. Δ Findings: change in mean findings per scan from the buggy change to its fixed companion (↓ fewer, ↑ more). Effort: mean output tokens, estimated cost, and minutes per scan.

Setup. We evaluate Claude Opus 5.5, Claude Sonnet 5, Grok 4.6 (xAI, 2026), GPT-5.6 sol (OpenAI, 2026c), GPT-6 sol, GPT-6 luna, GLM 5.3 (Z.ai, 2026; GLM-5 Team, 2026), and MiniMax M3 (MiniMax, 2026b) on the same inputs, withholding the reference and the later fix (Appendix D). Each model runs in an agent harness, the program that provides its tools and drives its loop. For direct review, the three GPT agents run in the Codex CLI (OpenAI, 2026a) at `xhigh` reasoning effort, and the others in Claude Code (Anthropic, 2026a), both general-purpose coding harnesses. For RQ3, we compare two security-focused harnesses: GPT-5.6 sol also runs as a static diff scan inside OpenAI's Codex Security plugin (OpenAI, 2026f), and Opus 5.5 runs inside Anthropic's Defending Code reference harness (Anthropic, 2026e). Web tools are disabled, and runs have no turn or token limit but time out after an hour (Appendix F.1).

5.1 RQ1: How Often Do Agents Find the Introduced Vulnerability?

Strong agents tie on easy targets and separate on contested ones. Grok 4.6, GPT-6 sol, GLM 5.3, and Opus 5.5 lead, each finding about three in four targets (Table 2). Grok is highest at 78.1%, but no pair among the four differs significantly (Appendix F.2); they differ more in speed, and Opus 5.5 is the fastest of all eight. To see where agents separate, we group targets by how many of the other seven agents find them. On easy targets, which nearly all other agents find, every agent but MiniMax M3 finds at least nine in ten; on contested ones, which only a few others find, the top four still find about half, while the other four find at most one in five (Appendix F.2). A newer model can close that gap: with the same harness and reasoning effort, GPT-6 sol finds 58 targets that GPT-5.6 sol misses and misses only four that it finds, with the largest gains on integer overflows and logic errors (Table 8). As in prior security benchmarks (Wang et al., 2026b; Lee et al., 2026; Wang et al., 2026a), combining agents improves coverage, but here the gains shrink quickly: the three leading agents together find about nine in ten targets, and all eight find only one more than the top four (Figure 4a). Even combined, the eight agents leave 23 targets unfound, far more than independent errors would predict (Appendix F.4). Training cutoffs make no clear difference: adjusted post-cutoff recall is several points lower for four of five testable models and slightly higher for GPT-6 sol, but every interval spans zero, and fixing dates cannot show training exposure (Appendix F.9).

5.2 RQ2: Why Do Agents Miss?

Most misses are misdiagnoses, not oversights. In 68% of scanned direct-review misses, the agent files a finding in a root-cause file, but the judge rejects it as describing a neighboring mechanism (Figure 4b); these reports mostly argue a competing diagnosis rather than declaring the site safe, and we did not re-adjudicate them. In curl CVE-2026-3805, a nine-line change to `lib/smb.c` leaves a request's path pointing into a buffer that dangles once its connection is freed; all eight agents report on that file, but four call the buffer leaked and the GPT agents call the line a logic bug, and

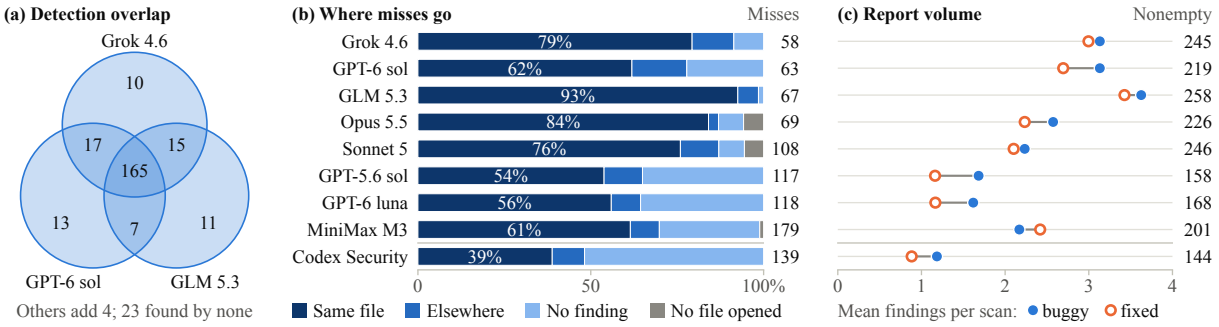


Figure 4: Results on the 265 buggy tasks and fixed companions. (a) Targets found by the three leading agents. (b) Misses: a finding in a root-cause file for another mechanism (%), findings only elsewhere, no finding, or no root-cause file opened. (c) Mean findings per buggy and fixed scan; Nonempty: fixed companions with any finding.

our audit upholds one such rejection (Appendix E). Agents rarely fail to reach the right file, but only because the diff almost always includes it. Nor is effort lacking: within difficulty strata, Opus's token spend does not differ significantly between misses and hits (Appendix F.4). Logic errors are harder than memory-safety flaws: averaged over agents, recall is about half against nearly three in four, and logic errors make up most targets that every agent misses (Table 8); bug class is confounded with project, but the gap persists outside V8. How far the mechanism spreads also matters: pooled recall falls from about seven in ten with one root-cause file to under four in ten with five or more, even among diffs of similar size, whereas change size and age show weak or no association; these splits are observational (Appendices F.4 and F.3).

5.3 RQ3: Does a Security-Focused Harness Help?

With GPT-5.6 sol at the same reasoning effort, Codex Security lowers recall by 8.3 points (Table 2), mostly by abstaining: on most tasks found only by direct review, it returns no finding (Appendix F.7). It also files fewer findings (Figure 4c), despite about five times the mean estimated cost. Anthropic's Defending Code reference harness (Anthropic, 2026e) raises Opus 5.5's recall by 7.7 points on the pairs it completed, all but five, with more findings at about ten times the median nominal cost (Appendix F.7). It alone finds all ten concurrency targets, where no direct-review agent finds more than six, and it finds six of the 22 unfound targets it completed, through breadth rather than ranking: each credited report there carries over a dozen findings. Both comparisons fix the backend model and reasoning effort; they remain descriptive because prompts, budgets, and operating points differ.

5.4 RQ4: How Do Agents Perform on Fixed Versions?

No fixed companion receives a positive target verdict, in direct review or either harness (Table 2; Appendix F.7). This is a narrow check: a negative verdict does not show whether a report reasserts the removed flaw, so these verdicts measure neither false-positive rates nor report precision. Location overlap alone would credit about three in four companion scans: over nine in ten nonempty reports still name a root-cause file. Still, every direct-review agent but MiniMax M3 files fewer findings after the repair (Δ Findings in Table 2; Figure 4c), and the drop follows correct diagnoses: when an agent's only finding on a buggy change is credited, its companion report is empty almost half the time, against about one in six when that finding is rejected. A fixed companion is thus a control only for its target, not a claim that the whole change is safe (Appendix F.6).

6 Conclusion

SecPRBench evaluates vulnerability diagnosis on real code changes, pairing 265 buggy tasks with repaired companions. Recall across eight direct-review agents ranges from 32.5% to 78.1%, with four agents statistically tied at the top. Strong agents tie on easy targets but separate on contested ones. Even combined, the eight agents leave 23 targets unfound. Most misses are misdiagnoses despite findings in the flawed file. Security-focused harnesses move recall in both directions: Codex Security abstains more, while Anthropic's Defending Code harness finds more at higher cost. On repaired changes, the judge never rules the removed flaw still present, though this

does not establish report precision. The automated pipeline supports refreshes from new disclosures; released tasks, reports, and verdicts let others inspect and re-grade the findings. Better diagnosis is the key remaining challenge.

Limitations

Scope. SecPRBench covers publicly disclosed vulnerabilities in nine widely deployed C, C++, and Rust projects, with changes of up to 30 files and 2,000 changed lines (§ 3.2), so our conclusions concern security review of systems code. Its targets were later disclosed and fixed, so they may differ from flaws that remain undiscovered. Because every buggy task contains a validated target, SecPRBench measures whether agents find and diagnose known vulnerabilities, not how often ordinary pull requests are vulnerable. Fixed companions are reviewed at the upstream fixing revision, which can include intervening changes, and control only for their targets, so SecPRBench does not measure report precision (§ 5.4).

Labels and judging. Of the 265 cases, 217 pass mechanical checks, mainly PoC replay on the parent, introducing, and repaired trees (§ 3.3). The other 48, such as partial fixes and races, rest on human adjudication of source and execution evidence, and we keep the automated outcome beside each decision. Our judge, Claude Opus 5.5 (high), is also an evaluated model, as is Sonnet 5, which drafted most references (§ 3.2). Switching the judge from Sonnet 4.6 to Opus 5.5 changes Opus 5.5's own verdicts on only seven tasks, four up and three down, and the two judges agree on 96.9% of 2,143 nonempty buggy reports (Appendix F.8). Our manual audit supports the judge on all 25 sampled agreements and 21 of 25 disagreements (Appendix E), and we release reports and verdicts so that other judges can re-grade them.

Runs and comparisons. Each agent has one run per task, without matched token budgets, so we report tokens, cost, and time beside recall (Table 2); each result describes a model in its harness, not the model alone. We do not repeat runs; paired tests over all 265 tasks separate each of the four leading agents from each of the other four (Appendix F.2). Harness comparisons fix the backend model and reasoning effort but not prompts or budgets, so they compare each harness as a whole, in the configuration we ran (Appendix F.7).

Training data. Public security records and fixes may appear in model training data. Agents review buggy changes without the security record, the fixing commit, Git history, or web tools (§ 3.1; Appendix F.1), and harness comparisons hold the backend model, and so its training data, fixed. For the five models with fixes on both sides of their documented cutoffs, adjusted recall after the cutoff is lower for four of them, but no difference is statistically clear (Appendix F.9). Because collection, tracing, and PoC replay are automated, SecPRBench can be refreshed with vulnerabilities disclosed after future training cutoffs.

Ethics Statement

The tasks are based on public security records and linked fixes. We reuse, adapt, or generate PoCs for these historical vulnerabilities only to validate task labels; the benchmark itself evaluates defensive source review. All nine projects are open source, and their code stays under its upstream licenses, such as MPL-2.0 for Firefox, BSD-3-Clause for V8, and Apache-2.0 for OpenSSL and Wasmtime. Tasks contain only public code, commit messages, and security records; we collect no personal data beyond developer names that the projects already publish. Benchmark labels do not establish a new vulnerability in a current release; any new finding requires independent validation and responsible disclosure to the affected project.

An AI coding assistant helped inspect artifacts, compute descriptive statistics, and edit and proofread the manuscript. Language model agents were also used for task construction, PoC preparation, evidence interpretation, and scoring. Agent validation reports do not count as independent human judgments. The authors are responsible for the data, adjudication decisions, and claims.

References

Georgios Alexopoulos, Nikolaos Alexopoulos, Diomidis Spinellis, and Dimitris Mitropoulos. 2026. Measuring and exploiting contextual bias in LLM-assisted security code review. *arXiv preprint*.

Anthropic. 2026a. Claude Code. Accessed September 25, 2026.

Anthropic. 2026b. Claude Code Security Review. Accessed September 21, 2026.

Anthropic. 2026c. Claude Opus 5.5 overview. Accessed October 12, 2026.

Anthropic. 2026d. Claude Sonnet 5 overview. Accessed October 12, 2026.

Anthropic. 2026e. Defending Code Reference Harness.

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. 2025. HealthBench: Evaluating large language models towards improved human health. *arXiv preprint*.

Lingfeng Bao, Xin Xia, Ahmed E. Hassan, and Xiaohu Yang. 2022. V-SZZ: Automatic identification of version ranges affected by CVE vulnerabilities. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*.

Big Sleep team. 2024. From Naptime to Big Sleep: Using large language models to catch vulnerabilities in real-world code. Google Project Zero blog.

Zhuo Chen, Boyang Wang, Xiyue Zhang, Xiaoyun Xu, Ahmad-Reza Sadeghi, Stjepan Picek, and Lichao Wu. 2026. From verdict to diagnosis: Attributable security review of pull requests. *arXiv preprint*.

Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David Wagner, Baishakhi Ray, and Yizheng Chen. 2025. Vulnerability detection with code language models: How far are we? In *IEEE/ACM International Conference on Software Engineering*.

GitHub. 2026. Security reviews now available in the GitHub Copilot app. Published July 14, 2026. Accessed September 22, 2026.

GLM-5 Team. 2026. GLM-5: from vibe coding to agentic engineering. *arXiv preprint*.

Kexing Ji, Jiachen Liu, Enze Hu, Cuiyun Gao, Keke Lian, Yongheng Liu, Lei Zhang, Tian Dong, Hao Chen, and Wang Bin. 2026. VulnGym: Benchmarking coding agents for repository-level vulnerability detection. *arXiv preprint*.

Hongchao Jiang, Yiming Chen, Yushi Cao, Hung-yi Lee, and Robby T. Tan. 2026. CodeJudgeBench: Benchmarking LLM-as-a-judge for coding tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19416–19448. Association for Computational Linguistics.

Deepak Kumar. 2026. SWE-PRBench: Benchmarking AI code review quality against pull request feedback. *arXiv preprint*.

Hwiwon Lee, Jiawei Liu, Dongjun Kim, Wubing Xia, Ziqi Zhang, Chunqiu Steven Xia, and Lingming Zhang. 2026. SEC-bench Pro: Can language models solve long-horizon software security tasks? *arXiv preprint arXiv:2605.26548*.

Hwiwon Lee, Ziqi Zhang, Hanxiao Lu, and Lingming Zhang. 2025. SEC-bench: Automated benchmarking of LLM agents on real-world software security tasks. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, pages 128940–128976.

Seunghyun Lee and David Brumley. 2026. ExploitBench: A capability ladder benchmark for LLM cybersecurity agents. *arXiv preprint*.

Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, and Neel Sundaresan. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

Jin Lu, Xuening Han, Yang Zhong, Lin Tan, Kevin Luo, Andrew Gacek, and Neha Rungta. 2026. VICBench: A multi-language benchmark for code vulnerability detection. *arXiv preprint*.

Rui Melo, Riccardo Fogliato, Sean Zhou, Pratiksha Thaker, and Zhiwei Steven Wu. 2026. SEVRA-BENCH: Social engineering of vulnerabilities in review agents. *arXiv preprint*.

MiniMax. 2026a. MiniMax-M3 chat template. Provider checkpoint template; accessed October 12, 2026.

MiniMax. 2026b. MiniMax M3: Frontier coding, 1M context, native multimodality—all in one model. Published June 1, 2026. Accessed September 25, 2026.

Mozilla Foundation. 2025. Mozilla Foundation Security Advisory 2025-19. CVE-2025-2857: Incorrect handle could lead to sandbox escapes. Announced March 27, 2025.

OpenAI. 2025. Introducing upgrades to Codex. Published September 15, 2025. Accessed September 22, 2026.

OpenAI. 2026a. Codex CLI. Accessed September 25, 2026.

OpenAI. 2026b. GPT-5.6 sol model documentation. Accessed October 12, 2026.

OpenAI. 2026c. GPT-5.6 system card. Published July 9, 2026. Accessed September 25, 2026.

OpenAI. 2026d. GPT-6 luna model documentation. Accessed October 12, 2026.

OpenAI. 2026e. GPT-6 sol model documentation. Accessed October 12, 2026.

OpenAI. 2026f. Scan code changes for security (Codex Security plugin documentation). Accessed September 25, 2026.

Niklas Risse and Marcel Böhme. 2026. LLM-based agents for identifying bug-introducing commits. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 81–92.

Niklas Risse, Jing Liu, and Marcel Böhme. 2025. Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*.

Tianneng Shi, Robin Rheem, Dongwei Jiang, Mona Wang, Francisco De La Riega, Zhun Wang, Jingzhi Jiang, Alexander Cheung, Sean Tai, Jonah Cha, Jianhong Tu, Gabriel Han, Chenguang Wang, Jingxuan He, Wenbo Guo, and Dawn Song. 2026. CyberGym-E2E: Scalable real-world benchmark for AI agents' end-to-end cybersecurity capabilities. In *International Conference on Machine Learning (ICML)*.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. 2025. PaperBench: Evaluating AI's ability to replicate AI research. In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 56843–56873.

Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse Coskun, and Gianluca Stringhini. 2024. LLMs cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks. In *IEEE Symposium on Security and Privacy (S&P)*.

Zhun Wang, Nico Schiller, Hongwei Li, Srijiith Sesha Narayana, Milad Nasr, Nicholas Carlini, Xiangyu Qi, Eric Wallace, Elie Bursztein, Luca Invernizzi, Kurt Thomas, Yan Shoshitaishvili, Wenbo Guo, Jingxuan He, Thorsten Holz, and Dawn Song. 2026a. ExploitGym: Can AI agents turn security vulnerabilities into real attacks? *arXiv preprint*.

Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. 2026b. CyberGym: Evaluating AI agents' real-world cybersecurity capabilities at scale. In *International Conference on Learning Representations*.

xAI. 2026. Model card: Grok 4.6. Published August 12, 2026; revision August 17, 2026. Accessed September 25, 2026.

Chenyuan Yang, Zijie Zhao, Zichen Xie, Haoyu Li, and Lingming Zhang. 2025. KNighter: Transforming static analysis with LLM-synthesized checkers. In *ACM Symposium on Operating Systems Principles (SOSP)*.

Alperen Yildiz, Sin G. Teo, Yiling Lou, Yebo Feng, Chong Wang, and Dinil Mon Divakaran. 2025. Benchmarking LLMs and LLM-based agents in practical vulnerability detection for code repositories. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30848–30865.

Jiaxin Yu, Peng Liang, Yujia Fu, Amjed Tahir, Mojtaba Shahin, Chong Wang, and Yangxiao Cai. 2024. An insight into security code review with LLMs: Capabilities, obstacles, and influential factors. *arXiv preprint*.

Z.ai. 2026. GLM-5.3: Frontier coding with emergent cyber capabilities. Published August 14, 2026. Accessed September 25, 2026.

Zhengran Zeng, Ruikai Shi, Keke Han, Yixin Li, Kaicheng Sun, Yidong Wang, Zhuohao Yu, Rui Xie, Wei Ye, and Shikun Zhang. 2026. SWR-Bench: Assessing LLM performance in real-world code review comment generation. *Proceedings of the ACM on Software Engineering*, 3(FSE):3093–3115.

Andy K. Zhang, Joey Ji, Celeste Menders, Riya Dulepet, Thomas Qin, Ron Y. Wang, Junrong Wu, Kyleen Liao, Jiliang Li, Jinghan Hu, Sara Hong, Nardos Demilew, Shivatmica Murgai, Jason Tran, Nishka Kacheria, Ethan Ho, Denis Liu, Lauren McLane, Olivia Bruvik, and 15 others. 2025a. BountyBench: Dollar impact of AI agent attackers and defenders on real-world cybersecurity systems. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*.

Andy K. Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W. Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, Pura Peetathawatchai, Ari Glenn, Vikram Sivashankar, Daniel Zamoshchin, Leo Glikbarg, Derek Askaryar, Mike Yang, Teddy Zhang, Rishi Alluri, and 8 others. 2025b. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. In *International Conference on Learning Representations*.

Lei Zhang, Yongda Yu, Minghui Yu, Xinxin Guo, Zhengqi Zhuang, Guoping Rong, Dong Shao, Haifeng Shen, Hongyu Kuang, Zhengfeng Li, Boge Wang, Guoan Zhang, Bangyu Xiang, and Xiaobin Xu. 2026a. AACR-Bench: Evaluating automatic code review with holistic repository-level context. *arXiv preprint*.

Yuntong Zhang, Zhiyuan Pan, Imam Nur Bani Yusuf, Haifeng Ruan, Ridwan Shariffdeen, and Abhik Roychoudhury. 2026b. Code review agent benchmark. *arXiv preprint*.

Zijie Zhao, Chenyuan Yang, Weidong Wang, Yihan Yang, Ziqi Zhang, and Lingming Zhang. 2026. AnyPoC: Universal proof-of-concept test generation for scalable LLM-based bug detection. *arXiv preprint arXiv:2604.11950*.

A Collector Inputs and Inclusion Evidence

Construction-stage counts. Table 3 summarizes collection, selection, and validation. Counts before selection pool successive collection rounds; the selected cases form the evaluation population (Appendix B). Each selected case has a buggy task and a fixed companion. Targets follow the collected security records: flaws reported and fixed separately are retained as separate cases, each with its own reference and fixed companion, even when they trace to the same introducing change.

Collection and eligibility. We collect candidates from two complementary sources: CVE-linked vulnerability records and public project security reports, including issues without assigned CVE identifiers. Project-specific collectors gather the associated advisories, bug reports, discussions, and fixing commits, and normalize them into candidate records containing an issue identifier, available report text, evidence of security relevance, and candidate fixing revisions. Beyond the eligibility criteria in § 3.2, we exclude documentation-only, test-only, dependency-update, and build-metadata changes, as well as unsupported security claims. Introducing and fixing revisions must lie on the project's mainline, and fixing revisions have committer timestamps on or after January 1, 2025, and before July 1, 2026. Table 4 records the source-specific inputs used by the project collectors. A security label is an inclusion signal, not a common severity scale across projects. Cases without CVE identifiers use project issue identifiers or publicly documented security-linked fixing commits.

The shared size thresholds are 30 changed files and 2,000 added plus deleted lines, with no minimum-line threshold. All 265 buggy changes and their fixed companions satisfy these caps. We retain the introducing and fixing revisions and the source evidence so that eligibility can be checked independently of agent results.

B Validation Evidence and Adjudication

Recorded outcomes and population. Of the 265 cases, identified by project and bug identifier, 217 are accepted through the mechanical route and 48 through human adjudication (Table 5). For the human-adjudicated cases, the available execution evidence and control coverage vary by case. Adjudication also distinguishes target failures from safe input rejection, feature absence, and unrelated execution failures in control runs.

Evidence scope also varies within a route. The FFmpeg case fixed by commit 4eca335d94ca passes mechanical validation with a source-based check of the repair instead of a replay on the repaired tree, and some Firefox cases replay their PoCs in component test harnesses. Per-case validation records, which we release, document the evidence behind each route decision.

Structural and mechanical checks. Before replay, structural checks cover commit hashes, task metadata, diff structure, and patch applicability. For mechanical validation, the shared validator reconstructs three snapshots from the introducing revision's first parent: the parent S_p, the buggy tree S_i, and the repaired tree S_r. When no public reproducer exists, AnyPoC generates one from the vulnerability report, the fix, and the repository. The validator records build results, patch-application status, exit codes, sanitizer output, and execution logs. For introducing changes, validation requires the same PoC and invocation to show the target failure on S_i and no target failure on S_p or S_r. The observed failure must match the reference root cause; unrelated crashes, build failures, and timeouts do not establish a positive verdict. We inspect the recorded evidence to confirm the observed behavior, not just build or crash status.

Partial fixes. We distinguish partial fixes in the reviewed change from those in its companion. A reviewed partial fix is a narrow exception to the requirement that a buggy task introduce the target: the same flaw may remain reachable before and after the change. Real repair commits can leave vulnerabilities unresolved; reviewers should identify why the changed logic fails to remove the target. We retain such cases only when source evidence establishes this causal link and the companion removes the target; an unrelated pre-existing flaw does not qualify. A companion that still contains the target is not a valid repaired control; one that removes it may still contain another defect.

In the SQLite case fixed by commit 03380a55a521, an attempted zipfile repair uses a 30-byte header constant in a bounds check that requires 46 bytes. The same PoC triggers the target overread in both the parent and reviewed versions. The companion corrects the constant and safely rejects the malformed archive.

Stage	Count
Raw source records, all rounds	>7,000
Triaged candidates	768
Selected cases	265
Validated cases: mechanical	217
Validated cases: human adjudication	48

Table 3: Collection and validation counts. Raw records include overlapping reports and repeated records of the same issue. Selected cases are accepted through executable checks (mechanical) or additional human adjudication.

Project	Collector input	Security signal
curl	Advisory index and machine-readable records	Project-maintained advisory information
FFmpeg	Trac, advisories, CVEs, and Git history	Advisory or commit evidence of a concrete security failure
Firefox	Bugzilla records and comments	Security-bug flags and critical/high/moderate security labels
FreeType	OSS-Fuzz-linked commits and GitLab issues	Source-based evidence of memory errors or malicious-font crashes
OpenSSL	OpenSSL advisory data and GitHub issues/PRs	Upstream ratings and source-based security screening
Redis	GitHub Security Advisories and OSV	Published advisories with security-fix references
SQLite	Project CVE page, Fossil forum, and tracker	Input-reachable memory-safety or corruption evidence
V8	Chromium security issues fixed in V8	Chromium security labels, restricted to fixes in V8
Wasmtime	GitHub Security Advisories and OSV	Published advisories linked to fixes

Table 4: Sources and security-relevance signals used by each project's collectors. All projects additionally require a traceable code change suitable for benchmark task construction.

Failures not observable with AddressSanitizer (ASan). In Firefox issue 1948111, the introducing change adds repeated locale-string writes that can race with worker reads. A recorded browser run exercises the relevant paths without an ASan error; source comparison with the fix identifies the missing synchronization. This motivates race-aware or source-based adjudication, rather than interpreting sanitizer silence as absence of the bug. FreeType OSS-Fuzz issue 499455833 offers a complementary example: a debugger shows a point index outside the glyph's logical zone, while historical over-allocation masks the access from ASan. Human adjudication reads this evidence alongside the source and repair.

Dataset and control identity. The released snapshot records the selected cases and the hashes, commit identifiers, and diff statistics of all 530 review inputs. Per-case validation records identify the source trees and controls that each check exercised. Each validation verdict applies to the recorded companion patch, not to other patches for the same bug.

C Task Schema and Snapshot Contract

Construction implementation. Collectors produce normalized source records from public security evidence. The construction agent writes reference findings and identifies fixing and introducing revisions. A mechanical wrapper then extracts the introducing change and packages the buggy task. The companion generator first attempts mechanical fix composition and uses a construction agent when adaptation is needed. Relevant-code annotations enrich the evaluation reference. The benchmark loader removes the reference and other evaluator-only metadata before review; the review prompt presents the commit message and file diffs, with repository context available through tools.

Stored fields. Task records retain repository and revision identities, the review change, and the evaluation reference. Table 6 distinguishes agent-visible inputs from constructor and evaluator metadata; the released schema documents individual fields.

Task materialization. The setup tool exports each task's recorded source snapshot using `git archive` and provides the review diff separately. Snapshots use the introducing revision for buggy tasks and the recorded fixing revision for companions. The recorded parent identifies the base of the review diff, rather than the exported snapshot. The exported workspace does not contain Git history. Each task runs in its own isolated Docker container; evaluator-only artifacts remain outside it, and outbound network access is restricted to configured

Project	Cases	Mechanical	Human
curl	26	23	3
FFmpeg	61	53	8
Firefox	27	18	9
FreeType	15	12	3
OpenSSL	29	28	1
Redis	16	12	4
SQLite	33	29	4
V8	44	32	12
Wasmtime	14	10	4
Total	265	217	48

Table 5: Validation routes for the 265 evaluated cases. Mechanical: cases accepted through recorded executable checks; Human: additional cases accepted by adjudication. Each case contributes two review inputs.

Information	Agent	Constructor/evaluator
Review diff, commit message	Yes	Yes
Task source snapshot	Yes	Yes, with exact revision recorded
Target security record	No	Yes
Later fixing commit/diff	No, except in a fixed companion	Yes
Reference findings and annotated root-cause locations	No	Yes
Buggy/fixed label and case identifier (e.g., CVE or bug ID)	No	Yes
Future Git history	No	Constructor only

Table 6: Information boundary. Reference information is withheld from the reviewing agent.

provider endpoints. Removing evaluator-only labels, references, and validation metadata preserves the upstream change text, diff, and source files.

Paired materialization for validation. Let $\Delta_b = \mathrm{diff}(p_i, c_i)$ and let Δ_f be the composed fixed patch. The shared validator materializes $S_i = \mathrm{apply}(p_i, \Delta_b)$ and $S_r = \mathrm{apply}(p_i, \Delta_f)$; the first is the buggy-task snapshot. Validation reconstructs both changes from their common parent, whereas agents review the recorded fixing snapshot for companions. Per-case records identify whether controls use an upstream fixing revision or an adapted source tree (Appendix B). A structural check confirms patch applicability, which alone does not establish target removal.

A concrete target. Wasmtime CVE-2026-34946 illustrates the need for repository context: the changed `table.fill` runtime expects a defined-table index, but the unchanged Winch caller still supplies a module-level index and caller context. For imported tables, these can identify different tables. A useful report must connect this interface mismatch to the unchanged caller, rather than merely warn about bounds.

D Prompt Design and Summaries

We summarize the substantive instructions and input fields below. These are condensed descriptions of the prompts, not verbatim copies of the original templates. Runtime paths, tool schemas, and repeated formatting instructions are omitted.

Direct-review prompt. For the Claude Code agents, the system message sets the security-review role, repository-investigation workflow, and reporting policy. The task message supplies the commit message and each changed file's path, status, and diff. Source code is available through file-reading, search, and shell tools. The core instructions are:

> Review the proposed change for security failures. Begin with changed functions and follow relevant callers, callees, and data structures. Check whether the change breaks assumptions about types, bounds, ownership, error handling, or control flow, including exceptional paths. Read the code; do not infer safety from stated intent.
>
> Report each plausible failure with a concrete location, a path to it, and its security impact. State uncertainty; a complete exploit is not required. Exclude unsupported speculation, style issues, and failures demonstrably prevented by the code. Continue reviewing after each finding.

In Claude Code, each finding is submitted through our reporting tool as JSON with a title,

explanation, location, bug type, severity, and confidence; related locations and suggested fixes can supplement it. This reporting policy favors candidate discovery. The benchmark's detection criterion is applied afterward by the judge.

The three GPT agents' Codex CLI prompt supplies the same commit message and diff, requests security issues introduced or made reachable by the change, limits findings to changed files, and requires a final JSON report. Codex Security and Anthropic's Defending Code reference harness follow their own review workflows (Appendix F.7).

Criteria-first judge prompt. The judge receives the task type, the reference root cause, reference findings, relevant locations, the agent's findings, and a selected change excerpt. These reference fields are withheld from the reviewing agent. Fixed companions use the original target reference together with the evaluated companion's change. The core instructions are:

> Derive three to six necessary criteria for the reference vulnerability before assessing the reports. Make the criteria specific enough to distinguish neighboring bugs. Check whether at least one finding identifies the same trigger, unsafe operation, and violated guard, ownership rule, bound, state transition, or trust boundary.
>
> Credit detection only if the change evidence supports that causal chain and no repair or mitigation breaks it. Reject adjacent bugs, location-only matches, hypothetical future risks, and historical or already-fixed failures. Examine all findings and return one task-level verdict.

The JSON response records detection, criteria, a match score, confidence, current-code support, mitigation flags, and a short explanation. Appendix E covers excerpt selection and why this verdict does not measure target alarms on fixed companions.

Construction instructions. Construction agents use public security evidence and fixing history to identify the root cause, affected code, and introducing and fixing revisions. When mechanical repair composition fails, a construction agent adapts the repair to the older source while preserving both the intended functionality of the introducing change and the fix's security-relevant behavior. The validation pipeline then checks these artifacts.

E Scoring Details and Their Limits

Criteria-first evaluation. Our evaluation runner prompts the judge to derive task-specific criteria from the reference root cause and reference findings. The judge evaluates the agent's final report against these criteria, using relevant diff sections selected by affected files and falling back to the full diff when necessary. Excerpts longer than 30,000 characters are truncated. The judge returns a structured verdict indicating detection, confidence, and supporting reasoning. Tasks with empty reports are counted as non-detections without a judge call, and all tasks remain in the recall denominator.

Audited group	Opus 5.5	Sonnet 4.6
Judge disagreements	21/25	4/25
Agreement controls	25/25	25/25

Table 7: Agreement with our manual adjudications.

Manual audit. We adjudicated 50 evaluation records with LLM assistance and checked each adjudication manually. The sample contains 25 selected Sonnet–Opus disagreements and 25 randomly selected agreement controls with nonempty reports. We compared the reference, findings, verdicts, and code, including the supplied change excerpt where available.

Disagreements were deliberately oversampled, so Table 7 reports each group separately; pooled percentages do not estimate overall judge accuracy. In the four cases where Opus departs from our adjudication, it broadened a target trigger or sink, supplied a missing causal step, or combined partial findings across mechanisms. Agreement concerns the binary verdict, not every supporting claim: one correct negative included a repair rationale whose decisive code lay outside the judge's excerpt.

Matching the mechanism despite different wording. In curl CVE-2026-9546, clearing `CURLOPT_REFERER` with automatic Referer handling disabled leaves `state.referer` unchanged: the new conditional update has no clearing branch. The audited report identifies this transition and subsequent header use, but describes a dangling pointer and cites the setter. Opus credits the shared causal chain, consistent with our adjudication; Sonnet rejects it over classification and location differences. The accepted match concerns the missing reset; we did not independently validate the report's stronger claim about memory lifetime.

Separating neighboring mechanisms. In the OpenSSL CVE-2025-9232 companion, the reference concerns an unterminated `host` buffer reaching `strstr`; the fix adds `host[sl] = '\0';`. The audited report instead alleges that the brack-

eted host [] yields an empty, terminated search string whose repeated matches advance past the end of no_proxy. Both judges reject target detection, consistent with our adjudication: the function and operation overlap, but the trigger and violated invariant differ. The audit does not validate neighboring findings. In curl CVE-2026-3805 (§ 5), GLM's report notes that the request path points into connection-owned memory but concludes that this is a leak and a latent hazard, not a live use-after-free. Sonnet 4.6 credits the report, whereas Opus 5.5 rejects it, consistent with our adjudication.

Why target-alarm measurement needs two questions. Let $M(P, G)$ mean that findings P assert the failure in reference G, and let $V(S, G)$ mean that snapshot S supports that failure; subscript f marks a fixed companion. Crediting a buggy-task detection requires both: $M(P, G) \wedge V(S, G)$. On a correctly fixed companion, $V(S_f, G)$ is false. The conjunction is therefore false even when the agent incorrectly asserts the removed bug. A target-alarm test instead asks $M(P_f, G)$ after independently establishing that the fix removed the target. The task-level verdict does not expose $M(P_f, G)$, so it gives no target-alarm rate or pair accuracy.

F Additional Results

These results use the frozen 265-case roster and Claude Opus 5.5 (high) verdicts. A positive saved verdict denotes detection. Missing or incomplete runs, errors, and empty reports count as misses. All eight direct-review agents and Codex Security completed every pair; Anthropic's Defending Code harness completed 260 of the 265 pairs. We release per-case records and analysis code.

F.1 Run Accounting and Configuration

Every final report has a verdict; empty reports receive a negative verdict without a judge call.

Empty structured reports are not distinguished by cause; only filed findings are evaluated. Usage comes from each run's session artifacts; output tokens are comparable only within a backend. Dollar figures are nominal estimates and exclude judging, task construction, and human adjudication.

Agent configuration. The five Claude Code agents run headlessly with built-in tools and our reporting tool; the three GPT agents use the Codex CLI's non-interactive mode. Opus uses high reasoning effort, the other Claude Code agents use their default setting, and the GPT agents use xhigh. The non-Anthropic models in Claude Code are served through Anthropic-compatible endpoints. Claude Code 2.1.283 is used for the Opus 5.5 reviewer, judge, and Anthropic harness; Codex-based runs use releases in the 0.149–0.160 series. The released artifacts record exact versions and container image pins per arm. Runs have no turn or token budget and time out after 60 minutes. Each run reviews an isolated snapshot with Git history removed, using file-reading, search, and shell tools, with built-in web tools disabled.

F.2 Paired Comparisons

We compare direct-review agents on all 265 tasks using exact two-sided McNemar tests, counting missing or errored runs as misses. No pair among the top four agents (Grok, GPT-6 sol, GLM, and Opus) differs significantly even before correction for multiple comparisons. After Holm adjustment across all 28 direct-review pairs, no difference is significant within the top four or among Sonnet, GPT-5.6 sol, and GPT-6 luna; all remaining pairs have $p < 0.001$. Collapsing tasks that share an introducing commit into 254 clusters, each credited when any (or every) member is detected, leaves Grok's comparisons with the other top agents nonsignificant even without adjustment. The other splits in this appendix use exploratory sign, Fisher, Mann–Whitney, or Spearman tests without multiple-comparison correction.

To see where agents separate, we group targets by how many of the other seven direct-review agents find them. On easy targets, found by six or seven others, every agent except MiniMax M3 finds at least 90%. On contested targets, found by one to three others, the top four find 48–57%, while Sonnet, GPT-5.6 sol, and GPT-6 luna find 10–19% and MiniMax M3 finds 5%.

F.3 Results by Task Characteristics

Project. V8 has the lowest average direct-review recall (45.2%) and nine of the 23 unfound targets. Wasmtime, FreeType, and Redis each have fewer than 20 cases, limiting project-level conclusions.

Bug class. Table 8 groups the reference bug types into memory safety (use-after-free, double free, out-of-bounds access, buffer overflow, null dereference, type confusion, uninitialized memory, wild pointer), logic error, integer overflow, and concurrency (data race or race condition). Averaged over

the eight direct-review agents, recall is 51.5% on logic errors and 73.1% on memory-safety flaws. Logic errors account for 13 of the 23 unfound targets. Class and project are entangled: about a third of the logic errors are in V8. Outside V8, logic-error recall remains lower than memory-safety recall for every agent, by 4–22 percentage points, although several differences are not statistically clear. The concurrency group is too small for class-level conclusions. With the same harness and reasoning effort, GPT-6 sol finds 58 targets that GPT-5.6 sol misses and misses four that it finds. GPT-6 sol's largest class gains are 34.1 points on integer overflows and 24.4 points on logic errors.

Change size. The median buggy change touches 4 files and 173 changed lines; 106 of the 265 buggy changes touch more than five files. Every agent has lower recall on changes touching more than five files, although the difference is not statistically clear for GPT-5.6 sol or Codex Security. The number of agents that find a target has a weak negative association with changed lines and files, which also vary by project.

Vulnerability age. The introduction-to-fix interval has no clear marginal association with detection for any agent. This analysis does not control for project or bug class, both of which vary with age; the within-V8 comparison is also inconclusive.

F.4 Detection Failures

Table 9 classifies each scanned miss from its trajectory and report. A task's *root-cause files* comprise the reference finding's location, affected files, and additional files annotated as necessary to explain the mechanism. For three tasks whose annotated paths do not match changed files at the introducing commit, we use the diff files instead. A file is *opened* when a read, search, or shell command names it; Codex Security is classified from its shell log. Each miss takes the first matching category: (A) never opened a root-cause file; (B) opened one but filed no finding; (C) filed findings only elsewhere; or (D) filed a finding in a root-cause file but the judge rejected its mechanism. Because the diff exposes root-cause files for nearly all tasks, category A is rare by construction and says little about search behavior. About 68% of pooled direct-review misses fall in category D. Recall is also lower when a mechanism spans more root-cause files: pooled over the eight agents, it is 69.6% for targets with one root-cause file and 38.5% for targets with five or more. The drop persists within diff size: from 76.5% to 55.1% (one vs. two or more root-cause files) among diffs of two to five files, and from 59.0% to 38.6% (one vs. five or more) among diffs of six or more files. Outside V8 the pattern is similar (70.3% to 51.6%), but only eight such tasks span five or more root-cause files. Root-cause files include annotated supporting files, and this split is observational.

In one category-D miss, Opus 5.5's report on SQLite CVE-2025-29088 overlaps the target's allocation logic, but the judge rejects it for describing pointer underflow in a different loop rather than the target mechanism. Category D rests on judge verdicts we did not re-adjudicate; most such reports offer competing diagnoses rather than explicitly claiming the target site is safe. About half name a root-cause function, although function names are taken at fix time and can differ at introduction. Trajectories do not always record subagent reads, so category A is an upper bound.

Median output-token use is higher on missed tasks for every direct-review agent, but also rises with diff size and, for Opus and Grok, with the number of other agents that miss the task. Within difficulty strata, the Opus difference is no longer statistically clear; Grok's differences vary by stratum and include small miss samples. These results are observational, not causal.

Unfound targets. Grok, GPT-6 sol, and GLM together find 238 targets (89.8%); adding Opus brings the total to 241, and all eight find 242 (91.3%). Codex Security also misses all 23 remaining targets. Nearly all nevertheless draw findings from at least four direct-review agents, and none draws only empty reports. Independent detections at each agent's observed recall would yield fewer than one universally missed target in expectation. The observed excess could reflect task difficulty, hard-to-satisfy reference criteria, shared judge errors, or shared training data; this comparison cannot distinguish these explanations.

F.5 Report Order

Among multi-finding direct-review detections with a resolved rank, the credited finding is first in 59.2% (33.1% under random ordering) and among the first two in 83.4%. Including single-finding reports, reading only the first finding reaches the target in 74.9% of detections with a resolved rank. Unresolved ranks are excluded from these

Class	n	Opus	Sonnet	Grok	GLM	MiniMax	GPT-6s	GPT-5.6	GPT-6l	Codex
Memory safety	125	85.6	68.8	84.0	85.6	40.0	82.4	69.6	68.8	60.8
Logic error	86	61.6	44.2	69.8	62.8	18.6	67.4	43.0	44.2	33.7
Integer overflow	44	68.2	63.6	84.1	70.5	45.5	79.5	45.5	45.5	43.2
Concurrency	10	60.0	50.0	50.0	60.0	0.0	60.0	40.0	30.0	20.0

Table 8: Recall (%) by reference bug class; n is the row denominator. Abbreviations: Opus 5.5, Sonnet 5, Grok 4.6, GLM 5.3, MiniMax M3, GPT-6 sol (GPT-6s), GPT-5.6 sol, GPT-6 luna (GPT-6l), and Codex Security.

Agent	Misses	(A) Never opened	(B) No finding	(C) Only elsewhere	(D) Same file, other mechanism
Claude Opus 5.5	69	5.8	7.2	2.9	84.1
Claude Sonnet 5	108	5.6	7.4	11.1	75.9
Grok 4.6	58	0.0	8.6	12.1	79.3
GLM 5.3	67	0.0	1.5	6.0	92.5
MiniMax M3	179	1.1	29.1	8.4	61.5
GPT-6 sol	63	0.0	22.2	15.9	61.9
GPT-5.6 sol	117	0.0	35.0	11.1	53.8
GPT-6 luna	118	0.0	35.6	8.5	55.9
Codex Security	139	0.0	51.8	9.4	38.8
Eight direct-review agents	779	1.5	21.6	9.4	67.5

Table 9: Scanned misses by category (% of each row's misses). Categories A–D are mutually exclusive; totals may differ slightly from 100% due to rounding. The pooled row includes the eight direct-review agents.

percentages; the Hit@1 bounds in Table 2 count none or all of them as first. The judge sees findings in the agent's reported order, so a judge position effect cannot be excluded.

F.6 Fixed Companions

Volume. Mean findings per scan fall for every direct-review agent except MiniMax (Table 10). In paired sign tests, Sonnet's decrease and MiniMax's increase are not statistically clear. Companion and buggy diffs have similar sizes, and median output tokens rise for every agent, so fewer findings do not reflect lower token use. Pooled over the eight direct-review agents, when a buggy report's only finding is credited, the companion report is empty 44.0% of the time, against 16.0% when that finding is rejected. After a category-D miss (Table 9), 88.6% of companion reports keep a finding in a root-cause file.

Location-only credit. Of the nonempty direct-review companion reports, 92.3% place a finding in a root-cause file. Crediting any such finding would accept 74.9% of direct-review companion scans, more than the 63.3% of buggy scans the judge credits (mean recall in Table 2).

What the judge records. No direct-review or Codex Security companion verdict is positive under Claude Opus 5.5 (high). A positive verdict requires the target mechanism to remain in the reviewed code; a negative verdict does not establish whether a report reasserts the removed target or raises a different issue. A fixed companion therefore controls only for its designated target; these verdicts do not measure report precision.

Concrete companion examples. Two cases from our audit illustrate this distinction (Appendix E). In OpenSSL CVE-2025-15467, the companion bounds the ASN.1 IV copy and rejects lengths above `EVP_MAX_IV_LENGTH`. GLM instead alleges that a shorter IV leaves bytes uninitialized. Both judges and our adjudication reject target detection: this allegation differs from the repaired oversized-IV stack overflow.

In curl pull request #21652, the companion checks the hexadecimal parser's return value and returns `HOST_NAME` on failure, rejecting the target's empty terminal `0x` component. Grok instead reports ordinary trailing-dot normalization and percent-encoded dots, calling the latter an incomplete fix. Both judges and our adjudication reject the target match; the audit does not assess whether these neighboring issues are real.

F.7 Review Harnesses

Codex Security. Codex Security uses the plugin's diff-scan skill (OpenAI, 2026f) through the Codex CLI (OpenAI, 2026a), with GPT-5.6

Agent	Mean findings per scan		Fixed nonempty (%)
	Buggy	Fixed	
Claude Opus 5.5	2.6	2.2	85.3
Claude Sonnet 5	2.2	2.1	92.8
Grok 4.6	3.1	3.0	92.5
GLM 5.3	3.6	3.4	97.4
MiniMax M3	2.2	2.4	75.8
GPT-6 sol	3.1	2.7	82.6
GPT-5.6 sol	1.7	1.2	59.6
GPT-6 luna	1.6	1.2	63.4
Codex Security	1.2	0.9	54.3

Table 10: Report volume across 265 completed pairs per agent. Fixed nonempty: companions with at least one finding. No fixed companion receives a positive target verdict.

sol at `xhigh` effort. Multi-agent delegation, web search, and sandbox network access are disabled. Scope rules restrict review to the last commit's diff and directly supporting code and forbid long build or test commands. Every buggy run completed, with no finding marked as dropped out of scope. Codex Security files no finding on 21 of the 31 targets found only by direct-review GPT-5.6 sol. Conversely, it finds nine targets that direct-review GPT-5.6 sol misses (exact McNemar $p < 0.001$). Its mean estimated cost per scan, \$3.74, is 5.1 times that of direct-review GPT-5.6 sol (Table 2). Our benchmark measures recall against one reference per task, but not precision on ordinary changes, which diff-scoped scanners are designed to review.

Defending Code reference harness. Anthropic's Defending Code reference harness (Anthropic, 2026e), using Opus 5.5 at high effort, completed 260 of the 265 pairs. On these 260 buggy tasks, its recall is 81.5%, versus 73.8% for direct-review Opus; it alone detects 31 targets, and direct review alone detects 11 (exact McNemar $p = 0.003$). It files more findings at higher cost (Table 11). Of the 23 targets that no direct-review agent finds, it completed 22 and detects six, three of them concurrency flaws; its credited reports on these six carry 14–32 findings each. On the 260 completed tasks, its Hit@1 upper bound is 54.6%, the same as direct-review Opus's. Counting the five missing pairs as misses gives 80.0% recall over all assigned tasks. Because the two arms differ in prompts and budgets, this comparison is descriptive. On companions, the harness averages 6.2 findings per scan and receives no positive target verdict; these verdicts do not establish report precision.

F.8 Judge Output Fields

All saved Claude Opus 5.5 (high) verdicts for the direct-review agents and Codex Security are complete and match the frozen task and report hashes. Empty reports receive negative verdicts without a judge model call. The judge receives the task type (buggy task or fixed companion). Its self-reported confidence is never low, and its auxiliary all-criteria-met flag almost always matches the verdict. Although credited findings usually receive high match scores, some have low scores; we count detections by verdict, not a score threshold. On identical reports, Opus 5.5 and Sonnet 4.6 differ on 67 buggy verdicts (35 gains and 32 losses with Opus) and one companion verdict (positive only under Sonnet 4.6). Opus 5.5's own reports change on seven buggy tasks (four gains and three losses), as do Sonnet 5's, while GPT-6 sol and GPT-6 luna gain the most (net eight and six). Excluding deterministic empty-report negatives, agreement is 96.9% across 2,143 nonempty buggy reports from the eight direct-review agents and Codex Security. This sensitivity comparison excludes the Anthropic harness because the two judge evaluations cover different task populations.

F.9 Temporal Comparisons

Raw recall gains on newer fixes are sensitive to case mix (Figure 5). Adjusted comparisons retain project–bug-class groups present in every period and weight their recalls by pooled case shares, fixed across periods; the half-year comparison retains 201 cases. Bootstrap intervals resample introducing-commit clusters; sparse groups limit precision, and some trend directions change when retained groups need two or more cases per period.

We compare fixing dates against declared knowledge cutoffs for the GPT models and Mini-

Arm	Recall (%)	Mean findings	Median $	Median min.
Anthropic harness, Opus 5.5	81.5	8.4	3.65	7.6
Direct Opus 5.5, same tasks	73.8	2.5	0.35	1.1

Table 11: Anthropic's reference harness and direct-review Opus 5.5 on the same 260 completed buggy tasks. Findings, nominal cost, and time are per scan; five assigned pairs are missing.

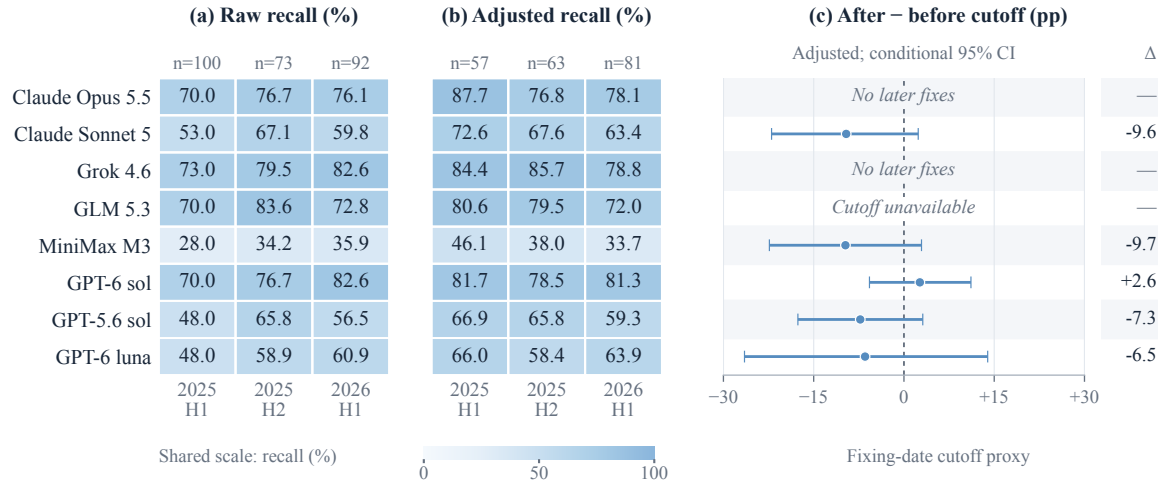


Figure 5: Direct-review recall by time: (a) all cases, (b) shared project–bug-class groups with fixed weights, and (c) within-model cutoff differences with conditional 95% cluster-bootstrap intervals. Headers give sample sizes.

		Cases: before/after		Adjusted recall (%)		After − before (pp)	
Model	Cutoff	All	Retained	Before	After	Δ	95% CI
Claude Sonnet 5	Jan. 2026	173/82	121/75	71.3	61.7	−9.6	[−22.0, +2.4]
MiniMax M3	Jan. 2026	173/82	121/75	42.7	33.0	−9.7	[−22.4, +2.9]
GPT-6 sol	Apr. 20, 2026	222/42	136/41	79.9	82.6	+2.6	[−5.7, +11.1]
GPT-5.6 sol	Feb. 16, 2026	189/76	136/74	66.0	58.7	−7.3	[−17.6, +3.1]
GPT-6 luna	May 18, 2026	235/28	104/28	63.6	57.1	−6.5	[−26.5, +13.9]

Table 12: Within-model cutoff comparisons. Counts exclude the cutoff day or month; retained cases share project–bug-class groups across periods. Recall uses fixed pooled weights.

Max (OpenAI, 2026b,e,d; MiniMax, 2026a) and against Sonnet's training-data cutoff (Anthropic, 2026d). Opus and Grok include training data through June 2026, leaving no later fixes (Anthropic, 2026c; xAI, 2026); GLM has no documented cutoff. All five within-model differences are inconclusive, with intervals that include zero (Table 12). Fixing dates do not establish training exposure or absence of contamination.

G Artifacts and Reproducibility

Code and data: `https://github.com/secprbench/secprbench`; results explorer with saved trajectories: `https://secprbench.github.io`. The released materials contain task metadata, evaluation results, a validation ledger with source hashes, and code to rebuild the tables without model calls. Scoring and report-volume accounting use the same final reports, matched to task inputs and saved verdicts by hashes. The release also includes the complete pairwise comparisons, task-level breakdowns, and finding-order records summarized above. Replaying validation requires the referenced source trees, builds, and PoCs; reconstructing model outputs also requires model access, the original assembled prompts, and harness artifacts. Rebuilding the tables does not re-check the judge's verdicts.