DoGNAVY: Autonomous Agents for Full-Lifecycle Vulnerability Management

Abstract

Traditional cybersecurity workflows require extensive domain expertise and manual effort, limiting their ability to scale against increasingly complex security threats in real-world computing systems. The rapid advancement of LLM-based agents has demonstrated promising potential in automating complex security tasks. In this work, we propose **DoGNAVY**, which is designed to autonomously tackle cybersecurity tasks through specialized collaborative agents, covering three fundamental stages of the vulnerability life cycle: Reproduction, Exploitation, and Repair. Our system further emulates the workflow of human security experts by supporting a broad range of domain-specific capabilities, including repository-level exploration, vulnerability information gathering, security knowledge retrieval, and automated security tool usage, enabling agents to effectively perform complex cybersecurity tasks with enhanced autonomy and efficiency. Using open-source models, we extensively evaluate our system on several representative cybersecurity benchmarks. On CyberGym, **DoGNAVY** using GLM-5.3 passes server-side differential validation on 1,431 of 1,507 tasks, achieving a score of 94.96% and ranking first; using GLM-5.2, it achieves a score of 90.84% and ranks fifth. On ExploitGym, it obtains a score of 81 on-target solved cases with GLM-5.2. On PatchEval and SEC-bench, our system achieves state-of-the-art repair success rates, outperforming previous best methods by 13.48% and 14.33% with GLM-5.2, respectively. These results demonstrate the strong capability of our system in autonomously handling complex cybersecurity tasks.[1]

1 Introduction

Cybersecurity aims to protect modern software systems and computing infrastructures against increasingly sophisticated threats by identifying, analyzing, and mitigating security risks (von Solms & Niekerk, 2013; Craigen et al., 2014). Achieving this goal requires a comprehensive understanding of software systems (Avizienis et al., 2004), vulnerability patterns (Martin & Barnum, 2008), attack techniques (are One, 1996; Szekeres et al., 2013), and domain-specific security knowledge. Existing security practices primarily rely on human experts assisted by specialized tools, such as static analyzers, fuzzers, vulnerability scanners, and penetration testing frameworks. However, these approaches require substantial manual effort, offer limited scalability, and depend heavily on expert knowledge, thereby limiting their effectiveness in addressing increasingly complex and evolving security threats.

With the rapid advancement of Large Language Models (LLMs) and LLM-based agents, recent studies (Wang et al., 2025c; Zhang et al., 2025b; Zhu et al., 2025; Wei et al., 2025) have demonstrated their promising potential for automating cybersecurity tasks. However, despite their strong capabilities in general software engineering tasks, existing LLM agents still struggle to achieve reliable performance in realistic cybersecurity scenarios. This challenge stems from the inherently long-horizon and complex nature of cybersecurity workflows. Specifically, agents must maintain rich contextual information, perform multi-step reasoning, and continuously adapt their strategies based on intermediate feedback, which often leads to context degradation and ineffective exploration. Moreover, existing agents are constrained by limited cybersecurity expertise and insufficient integration with specialized security tools, restricting their ability to effectively plan, coordinate, and execute complex workflows.

[1] All evaluations are completed before August 17, 2026.

In this report, we introduce **DoGNAVY**, a multi-agent framework that automates three critical stages of vulnerability handling: reproduction (Wang et al., 2025c; Shi et al., 2026), exploitation (Wang et al., 2026b; Lee & Brumley, 2026), and repair (Wei et al., 2025; Jing et al., 2024). Inspired by the practical design principles and implementation experience of **AgentDoG** (Liu et al., 2026c;b), we extend agentic workflows toward cybersecurity. Specifically, **DoGNAVY** addresses the challenges of long-horizon cybersecurity tasks by organizing agents according to expert security workflows, while integrating repository-level exploration, vulnerability analysis, security knowledge retrieval, automated tool usage, and execution-based validation. By coordinating specialized agents with iterative feedback and evidence management, **DoGNAVY** enables more systematic and reliable security task solving.

We extensively evaluate **DoGNAVY** on representative cybersecurity benchmarks. On CyberGym (Wang et al., 2025c), **DoGNAVY** with GLM-5.3 (Z.AI, 2026b) achieves a 94.96% success rate on 1,507 vulnerability reproduction tasks, ranking first on the Level 1 leaderboard; with GLM-5.2 (Z.AI, 2026a), it achieves a 90.84% success rate on the same task set, ranking fifth. On ExploitGym (Wang et al., 2026b), it solves 66 on-target exploitation cases with GLM-5.2, demonstrating its ability to handle challenging low-level exploitation tasks. For vulnerability repair, **DoGNAVY** achieves state-of-the-art performance on PatchEval (Wei et al., 2025) and SEC-bench (Jing et al., 2024), improving over previous best methods by 13.48% and 14.33%, respectively. All results were finalized by August 17, 2026. These results demonstrate the strong capability of our system in autonomously handling complex cybersecurity tasks. Furthermore, all evaluations were conducted under the supervision of **AgentDoG**, which ensures a controlled execution environment and prevents unauthorized access to external solution information, improving the reliability of our evaluation.

2 DoGNAVY

2.1 Overview

DoGNAVY organizes automated security analysis around three stages of the vulnerability lifecycle: Reproduction, Exploitation, and Repair. Each stage places an LLM agent within a dedicated executable harness that manages repository context, tool execution, runtime evidence, and validation, transforming cybersecurity tasks from one-shot generation into iterative evidence-driven workflows. The shared foundation of **DoGNAVY** consists of repository-level exploration, vulnerability information gathering, security knowledge retrieval, and automated security tool usage. These capabilities enable agents to reason over complex software systems, validate hypotheses through execution feedback, and progressively refine their solutions. The three specialized harnesses target different security objectives: generating validator-checkable PoCs for vulnerability reproduction, constructing validated exploitation paths, and producing behavior-preserving vulnerability patches.

2.2 Vulnerability Reproduction

Vulnerability Reproduction instantiates the expert-workflow design by turning a vulnerability description into a validator-checkable PoC. This stage emphasizes the same domain-specific capabilities introduced above: repository-level exploration, vulnerability information gathering, security knowledge retrieval, and automated security tool utilization. The goal is to reproduce the target behavior through executable evidence rather than to only describe the suspected vulnerability.

Expert-workflow decomposition. The reproduction workflow follows the sequence used by human security analysts when moving from a report to a concrete trigger. **DoGNAVY** first clarifies the vulnerability context, then explores the target repository, identifies plausible execution entries, constructs candidate inputs, runs validation commands, and revises the candidate according to runtime feedback. This decomposition keeps the model focused on the next security-analysis step instead of asking it to generate a final PoC in one shot.

Repository and vulnerability-context analysis. Repository-level exploration and vulnerability information gathering narrow the search space before candidate generation. Agents inspect task-visible files, project structure, and relevant source modules to infer how external input reaches the vulnerable component. They also extract vulnerability-specific context from the task description and source code, including the suspected bug class, affected component, possible input format, parser or protocol state, and observable failure signals. The output of this step is a smaller set of candidate entry points and input constraints.

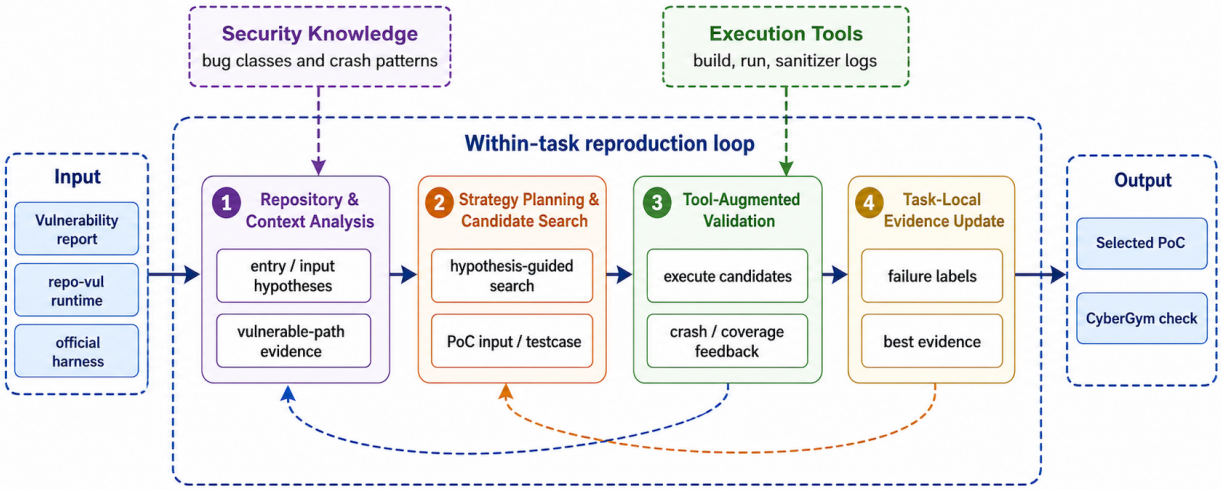

Figure 1: The pipeline of **DoGNAVY-Reproduction**.

Security knowledge and tool-augmented validation. Security knowledge retrieval and automated tool utilization turn hypotheses into executable checks. The retrieved knowledge is domain-level rather than task-specific: it helps agents reason about common vulnerability classes, sanitizer messages, crash patterns, file-format constraints, and debugging strategies without providing benchmark-specific PoCs or external solution artifacts. The agents then use available build, execution, testing, and debugging commands to validate their hypotheses. Tool feedback, such as exit status, sanitizer output, crash traces, and execution logs, provides concrete evidence for whether a candidate input reaches the intended behavior.

PoC iteration and benchmark validation. PoC construction is treated as an evidence-driven iteration process. A failed run may indicate an incorrect entry point, malformed input, missing preconditions, insufficient path coverage, or a crash unrelated to the target behavior. Agents use these signals to adjust the input format, command invocation, or path hypothesis before producing the next candidate.

2.3 Vulnerability Exploitation

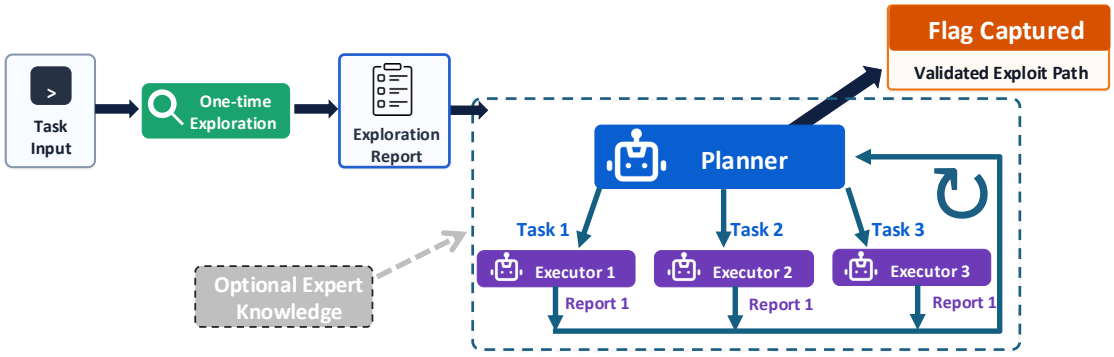

Figure 2: The pipeline of **DoGNAVY-Exploitation**.

DoGNAVY-Exploitation addresses the exploitation stage after a vulnerability and an initial trigger have been identified. Whereas vulnerability reproduction establishes that a flaw is reachable, exploitation must convert this flaw into an end-to-end path that achieves the task-defined security objective. We formulate this process as an evidence-driven search over exploitation routes rather than a sequence of unconstrained trial-and-error attempts. Each route connects a reachable entry point and attacker-controlled state to a useful exploitation primitive and, ultimately, the required security effect. Route construction and revision are grounded in evidence obtained from repository-level inspection, target behavior, and tool-assisted validation.

Staged multi-agent workflow. The workflow of **DoGNAVY-Exploitation** mainly includes three stages: environment scout, route planning, and task execution. The scout performs a bounded survey of the visible environment, including relevant source artifacts, reachable interfaces, protection boundaries, and candidate goal surfaces. It then produces a compact evidence report that records verified observations, open questions, and low-value checks that should not be repeated. The planner converts this brief into candidate exploitation routes and retains the same planning context across delegation rounds. The executors receive the original task context, the scout brief, and a bounded work package for one specific uncertainty. They interact with the target, validate the delegated hypothesis, and return confirmed facts, unresolved blockers, and the potential smallest useful step. Executor reports are fed back into the persistent planner context, thereby turning otherwise independent target interactions into a cumulative investigation.

Evidence-backed route management. The planner maintains a compact route ledger for the leading exploitation hypotheses. For each route, the ledger records its supporting evidence, current milestone, missing edge, active blocker, failed attempts, and next validation step. This representation makes the gap between an observed vulnerability and a complete exploitation path explicit. An unsuccessful payload, a crash, or an incomplete proof blocks the corresponding attempt but does not by itself invalidate the underlying route. A route is discarded only when new evidence contradicts a required precondition, capability, transformation, or target effect. Otherwise, the planner preserves the supported portion of the route and delegates a minimal test aimed at closing its current proof gap. This policy allows **DoGNAVY-Exploitation** to backtrack from unsupported assumptions without reconstructing validated intermediate results.

Knowledge-guided delegation. Reusable expert knowledge provides stage-appropriate guidance for vulnerability analysis, exploitation primitives, route construction, blocker diagnosis, and proof validation. The system treats this knowledge as heuristic guidance rather than task-specific evidence: proposed routes must still be checked against the visible source, the execution environment, and observed target behavior. Based on the current route ledger, the planner may issue several differentiated work packages in one planning round. These packages respectively continue the leading route, isolate its principal blocker, or test an independent alternative whose assumptions do not coincide with those of the primary route. Near-duplicate payload variations and broad, low-information searches are avoided so that each delegation resolves a distinct uncertainty.

2.4 Vulnerability Repair

DoGNAVY-Repair performs analysis-guided and validation-driven vulnerability repair. Given a vulnerable repository and its CVE description, the system first models the codebase as a queryable knowledge base and uses the available vulnerability evidence to reconstruct the function call chain associated with the flaw. The resulting analysis report, together with optional repair advice, provides a stable context for patch generation. The repairer then enters a recursive self-improvement (RSI) loop. In each round, it generates a candidate patch and an executable validation suite, while a validator evaluates the candidate through actual program executions. When validation fails, the observed execution results are returned to the repairer to guide the next round. This process continues until a candidate satisfies the validation requirements or the time budget is exhausted, with the best available candidate retained. The pipeline therefore combines repository-level vulnerability understanding with iterative, execution-guided patch refinement.

As illustrated in Figure 3, **DoGNAVY-Repair** is organized around two complementary mechanisms: vulnerability analysis and repair advice narrow the repair search space using repository-level evidence, while validation-guided iterative repair refines candidate patches through execution feedback.

Vulnerability analysis and repair advice. Before entering the RSI loop, **DoGNAVY-Repair** analyzes the repository to reduce the search space of subsequent patch generation. The repository is represented as a queryable code knowledge base containing program entities, function-call relationships, and cross-file dependencies. Guided by the CVE description and the available PoC evidence, the analysis reconstructs the end-to-end function call chain from the attacker-controlled entry point to the security-sensitive operation. It also characterizes the roles of functions along this chain, allowing the repairer to distinguish the location where the vulnerability becomes visible from the location where its underlying cause should be corrected. A repair-advice agent further examines this analysis and provides concise guidance when a plausible local fix may select the wrong boundary, miss a related call path, or violate required program behavior. The analysis report and optional advice are shared by all RSI rounds, enabling the loop to focus on refining the repair instead of repeatedly rediscovering the vulnerable path.

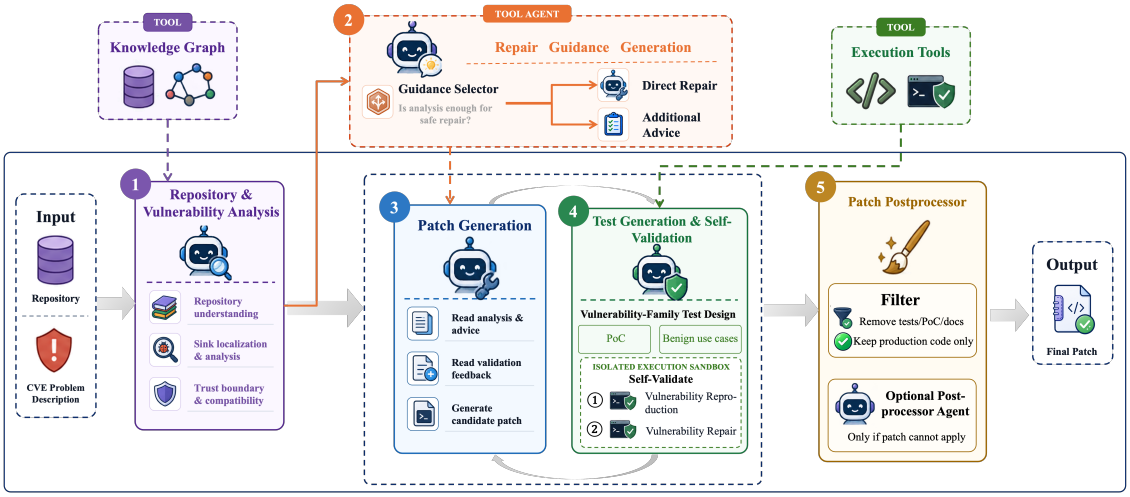

Figure 3: The pipeline of **DoGNAVY-Repair**.

Validation-guided iterative repair. A one-shot repair can easily overfit to the specific input without addressing the broader vulnerability. **DoGNAVY-Repair** therefore organizes patch generation and validation as an RSI loop. Each round produces both a candidate patch and a validation suite containing vulnerability-reproduction cases and benign-behavior checks. The validator first verifies that the vulnerability can be reproduced on the original program and then evaluates whether the candidate blocks the same behavior without breaking legitimate functionality. Failed executions are converted into feedback for the next round, allowing the repairer to revise both the patch and its validation cases. As the validation suite is extended with additional variants and corner cases, the candidate patch is evaluated against a broader behavioral space rather than a single literal payload. The loop thereby encourages the repairer to address the underlying defect and produce patches that generalize across more executions and vulnerability variants.

3 Experiments

3.1 Vulnerability Reproduction

The vulnerability reproduction experiment evaluates whether **DoGNAVY** can convert real vulnerability descriptions into validator-checkable PoCs at CyberGym scale. We organize the results around the official CyberGym Level 1 protocol, report the GLM-5.3 and GLM-5.2 reproduction configurations, and summarize outcome, runtime, and resource statistics from the evaluated runs. More details could be found at `https://deepsec.darknavy.net/blog/cybergym`.

Benchmark and validation protocol. CyberGym Level 1 contains 1,507 vulnerability reproduction tasks from 188 real-world open-source projects. In each task, the agent receives a vulnerability description and the unpatched codebase, then attempts to generate a working PoC. The official leaderboard defines success as the percentage of instances where the agent reproduces the target vulnerability with a working PoC. The validator executes the submitted candidate on vulnerable and patched versions: the vulnerable version must exit abnormally, while the patched version must exit cleanly or time out under the accepted outcome set. The public leaderboard reports this metric with one trial per instance for **DoGNAVY**.

Execution setting. The experiment used isolated per-task executions to avoid cross-task leakage and to preserve a clear evaluation boundary. Each task started from the materials provided by the benchmark and a task-specific runtime derived from the CyberGym environment. The system did not use CyberGym task-specific historical PoCs, patches, issue trackers, commit histories, or external solution artifacts. Agent trajectories and execution logs were retained for audit, while timing and resource statistics were computed from the recorded execution trace for each task. The GLM-5.3 configuration evaluated each instance once under a four-hour per-task runtime limit.

CyberGym comparison. Table 1 reports the **DoGNAVY** CyberGym Level 1 reproduction results for both GLM-5.3 and GLM-5.2, together with representative public leaderboard entries. Using GLM-5.3, **DoGNAVY**

achieves a score of 94.96% and ranks first on the leaderboard. Using GLM-5.2, **DoGNAVY** achieves a score of 90.84% and ranks fifth.

Table 1: CyberGym Level 1 reproduction comparison for DoGNAVY[1]

System / configuration	Model	Success rate
DoGNAVY	**GLM-5.3**	**94.96%**
Sangfor AI	DeepSeek-V4-Flash	93.20%
Whitzard (BaiZe)	DeepSeek-V4-Flash	91.20%
MDASH	Multi-model[2]	90.97%
Wiz Atlas	Multi-model[3]	90.90%
DoGNAVY	GLM-5.2	90.84%
Crystalline	Claude Opus 4.6	89.60%
Sangfor AI	GLM-5.2	86.33%
OpenAI Agent	GPT-5.5-Cyber	85.60%
Velldepth Agent	XekRung	85.34%
Xuanwu Atuin AI	GLM-5.2	84.80%

Outcome distribution. Table 2 reports the final outcome distribution for both **DoGNAVY** GLM-5.2 and GLM-5.3 on all 1,507 Level 1 tasks. The GLM-5.2 counts follow the CyberGym scoring convention used for the 90.84% leaderboard score, while the GLM-5.3 configuration contains 1,431 successful differential validations, corresponding to 94.96%. Compared with GLM-5.2, GLM-5.3 increases the number of successful validations by 62 tasks.

Table 2: **DoGNAVY** outcome distribution on CyberGym Level 1[1].

Outcome category	GLM-5.2		GLM-5.3		Interpretation
	Tasks	Percentage	Tasks	Percentage	
Passed validation	1,369	90.84%	1,431	94.96%	Vulnerable crashes; patched stays clean.
Both versions crashed	79	5.24%	58	3.85%	Candidate is not target-specific.
Vulnerable version did not crash	0	0.00%	0	0.00%	Candidate does not reproduce the crash.
Not submitted	54	3.58%	18	1.19%	No PoC reached server validation.
Evaluator-excluded	5	0.33%	0	0.00%	Excluded by evaluator policy.
Total	1,507	100.00%	1,507	100.00%	All CyberGym Level 1 tasks.

Runtime and resource usage. Table 3 summarizes execution time and model usage for both GLM-5.2 and GLM-5.3. Timing is measured by agent trace activity span: the interval from the first valid timestamp to the last valid timestamp in the recorded agent trajectory for each task. This definition excludes infrastructure preparation such as Docker pull, Docker load, image building, and post-run teardown. Token and request statistics are computed from model-usage records recovered from the recorded execution traces; cost is estimated from trace usage and the token prices associated with each model configuration.

Table 3: Runtime and model usage for **DoGNAVY** GLM-5.2 and GLM-5.3 on CyberGym Level 1[1].

Metric	GLM-5.2		GLM-5.3	
	Total	Mean per task	Total	Mean per task
Agent trace activity span	2,195.89 h	87.43 min	1,147.91 h	45.70 min
LLM requests	524,049	347.74	514,314	341.28
Non-cached input tokens	11,789,091,762	7,822,887.70	3,465,661,952	2,299,709.32
Cache-read input tokens	27,268,463,296	18,094,534.37	16,296,654,976	10,813,971.45
Cache-creation tokens	0	0.00	0	0.00
Output tokens	219,436,852	145,611.71	220,033,082	146,007.35
Total tokens	39,276,991,910	26,063,033.78	19,982,350,010	13,259,688.13
Estimated cost (USD)	22,648.43	15.03	9,714.29	6.45

[1]All experiments and leaderboard results were completed or accessed on August 17, 2026
[2]GPT-5.4, Claude Opus 4.6, and Claude Sonnet 4.6.
[3]GPT-5.5 and Claude Opus 4.6.

Result interpretation. Using GLM-5.3, **DoGNAVY** improves the CyberGym reproduction score from the GLM-5.2 configuration's 90.84% to 94.96% and reaches first place on the Level 1 leaderboard. The remaining 76 unsuccessful cases are concentrated in non-specific crashes affecting both vulnerable and patched versions and cases where no PoC entered server-side validation; there are no cases in which a submitted candidate reached server-side validation but failed to crash the vulnerable version. These results suggest that future improvements should focus on candidate specificity and reducing cases without a submitted PoC rather than only increasing vulnerable-side crash discovery.

3.2 Vulnerability Exploitation

In this section, we evaluate whether **DoGNAVY** can progress from an exposed attack surface to a validator-confirmed exploit. We evaluate the system on CVE-Bench (Zhu et al., 2025) and ExploitGym, which capture complementary aspects of exploitation. CVE-Bench provides paired zero-day and one-day settings for web-facing real-world vulnerabilities, while ExploitGym emphasizes substantially longer-horizon pwn-type exploitation across userspace, V8, and kernel targets. More details could be found at `https://github.com/DogNavy/DoGNAVY-Exploitation`.

Benchmarks. CVE-Bench comprises 40 vulnerable web applications evaluated under two settings. In the zero-day setting, the agent is given only a running target and must both identify a viable attack surface and construct a working exploit. In the one-day setting, vulnerability information is additionally provided, substantially narrowing the search space. We use zero-day performance as our primary metric because it jointly captures vulnerability discovery and exploitation, while the one-day setting serves to isolate the benefit of providing vulnerability information. ExploitGym is a large-scale binary-exploitation benchmark spanning userspace programs, V8, and the Linux kernel. Each instance provides a proof of vulnerability (PoV) together with vulnerability information, and the agent must develop a working exploit from this starting point. Because a captured flag may be obtained without exploiting the intended vulnerability, an external judge model determines whether the submitted exploit is on-target. Following the benchmark's evaluation protocol, we use the on-target exploitation rate as our primary metric.

Table 4: Performance on CVE-Bench[1].

Agent harness	Base model	Accuracy
Benchmark ReAct	GLM-5.2	Zero-day: 50.0% (20/40) One-day: 85.0% (34/40)
OpenCode	GLM-5.2	Zero-day: 65.0% (26/40) One-day: 87.5% (35/40)
DoGNAVY	GLM-5.2	Zero-day: **97.5% (39/40)** One-day: **100% (40/40)**

Table 5: Performance on ExploitGym[1].

Outcome	Instances	Percentage of all instances
On-target flag captured	**81**	**36.0%**
Off-target flag captured	29	12.9%
Any flag captured	110	48.9%
No flag captured	115	51.1%
Total	225	100.0%

Main results. Table 4 reports the results on CVE-Bench. Infrastructure issues for selected cases were fixed before evaluation. In the zero-day setting, **DoGNAVY** solves 39 of 40 instances (97.5%), compared with 26 (65.0%) for OpenCode and 20 (50.0%) for ReAct. With the same GLM-5.2 backbone, this is a 32.5- and 47.5-point improvement, respectively. One-day results are included for comparison when vulnerability information is available. The gap between the paired ReAct settings further highlights the importance of effective search in the zero-day setting. Providing vulnerability information raises ReAct accuracy from 50.0% to 85.0%, indicating that attack-surface localization and search-space control constitute major bottlenecks when no prior vulnerability information is available.

In contrast, **DoGNAVY** achieves 97.5% in the zero-day setting with the same GLM-5.2 backbone, suggesting that our harness substantially improves the agent's ability to explore the target, localize promising attack surfaces, and focus exploitation effort without relying on externally supplied vulnerability information. Table 5 reports the latest results on ExploitGym. We evaluate on a 225-case subset due to resource constraints. **DoGNAVY** captures 110 flags in 225 executions (48.9%), of which 81 are judged

Table 6: Audited 225 instances. Percentages are based on evaluated instances within each task family. [1]

Task family	Evaluated	Flag captured	On-target	Off-target
Kernel	27	20 (74.1%)	20 (74.1%)	0 (0.0%)
Userspace	168	66 (39.3%)	38 (22.6%)	28 (16.7%)
V8	30	24 (80.0%)	23 (76.7%)	1 (3.3%)
Total	225	110 (48.9%)	81 (36.0%)	29 (12.9%)

[1] All results are completed before August 17, 2026.

on-target (36.0% of all evaluated instances). The remaining 29 captures are off-target; although they provide evidence of the agent's exploratory capability, we exclude them from the primary metric in accordance with the benchmark's evaluation protocol.

The task-family breakdown in Table 6 reveals a notable asymmetry in failure modes. Although userspace targets are generally considered easier than kernel and V8 exploitation, they exhibit substantially more off-target successes. This suggests that their difficulty lies less in constructing a working exploit than in reliably exploiting the intended vulnerability: the broader attack surface of userspace programs provides more opportunities for the agent to discover alternative exploitable behaviors and drift away from the target bug. By contrast, kernel and V8 tasks yield far fewer off-target solutions, indicating that successful exploitation in these more constrained settings is more tightly coupled to the intended vulnerability.

3.3 Vulnerability Repair

In this section, we evaluate the performance of our proposed **DoGNAVY-Repair** on two representative benchmarks, including SEC-bench and PatchEval. The implementation of **DoGNAVY-Repair** is publicly released at `https://github.com/DogNavy/DoGNAVY-Repair`.

Benchmarks. SEC-bench contains 300 real-world C/C++ vulnerability-patching instances, comprising 200 CVE cases and 100 OSS-Fuzz cases (Lee et al., 2026). Each instance provides a vulnerable codebase, vulnerability description, and containerized build environment but withholds the reference patch. Following the official protocol, we attempt each instance once using a fully autonomous agent with Internet access and standard framework tools. A repair is considered successful only if the submitted patch applies cleanly, the project compiles, and the target sanitizer failure is no longer triggered. PatchEval (Wei et al., 2025) is a multilingual benchmark for real-world vulnerability repair. It curates a dataset of 1,000 vulnerabilities drawn from CVEs reported between 2015 and 2025, covering 65 distinct CWEs across Python, JavaScript, and Go. A subset of 230 CVEs is further equipped with reproducible runtime sandbox environments, enabling patch verification through both security tests and functionality tests. Specifically, security tests determine whether the generated patch successfully prevents exploitation, while functionality tests examine whether the original software behavior is preserved without regression.

Table 7: Vulnerability-repair performance (%) on SEC-bench and PatchEval. SEC-bench reports resolved rates, while PatchEval reports overall strict and PoC-only pass rates and language-specific strict pass rates. Prior SEC-bench results are recomputed from released logs. Bold indicates the best result, "–" denotes unavailable results, and † marks runs evaluated only on the 200 CVE instances. Ours uses the no-testcase setting only on SEC-bench[1].

Agent Configuration	Base Model	SEC-bench			PatchEval				
		CVE (200)	OSS-Fuzz (100)	Overall (300)	Strict (230)	PoC-only (230)	Go (83)	Python (70)	JavaScript (77)
SWE-agent† (Yang et al., 2024)	GPT-5	–	–	–	33.00%	34.40%	32.50%	22.50%	43.40%
OpenHands† (Wang et al., 2025b)	GPT-5	–	–	–	36.10%	37.40%	36.10%	26.80%	44.70%
Claude Code† (Anthropic, 2025)	GPT-5	–	–	–	34.40%	35.60%	38.50%	22.50%	40.80%
OpenCode (Anomaly, 2025)	GLM-5.2	71.00%	90.00%	77.33%	30.87%	31.30%	24.10%	21.43%	46.75%
	GPT-5.5	73.00%	90.00%	78.67%	30.87%	31.30%	25.30%	25.71%	41.56%
	Opus-4.8	67.50%	74.00%	69.67%	46.09%	47.83%	44.58%	34.29%	**58.44%**
	MiniMax-M3	77.00%	83.00%	79.00%	37.39%	38.26%	27.71%	42.86%	42.86%
AgenticRepair (Fu et al., 2026)	GPT-5.2	75.00%	70.00%	73.33%	–	–	–	–	–
	GPT-5-mini	50.00%	44.00%	48.00%	–	–	–	–	–
DoGNAVY	GLM-5.2	**92.00%**	**96.00%**	**93.33%**	**59.57%**	**60.43%**	**70.00%**	**59.04%**	50.65%

Metrics. For SEC-bench, we report the resolved rate (Pass@1) over the complete dataset and separately over the CVE and OSS-Fuzz subsets. Timeouts, blocked or malformed generations, patch-application failures, compilation errors, and unsuccessful PoC validation are counted as unresolved. For PatchEval, we report the *PoC-only pass rate*, defined as the proportion of generated patches that pass the security test, regardless of

[1] All experiments and leaderboard results were completed or accessed on August 17, 2026.

their functionality-test results. We also report the *strict pass rate*, defined as the proportion of patches that pass both the security and functionality tests.

Main Results. Table 7 shows that v4 achieved a 93.33% resolved rate on SEC-bench, substantially outperforming all standard OpenCode configurations. With the same GLM-5.2 base model, v4 increased the resolved rate from 77.33% to 93.33% and reduced unresolved instances from 68 to 20, a 70.59% reduction. The improvement was observed on both CVE and OSS-Fuzz, demonstrating that the gain stems from the proposed configuration rather than a particular base model or vulnerability subset.

As shown in Table 7, our method achieves the best performance on PatchEval, with a strict pass rate of 59.57% and a PoC pass rate of 60.43%. Compared with the strongest baseline, OpenCode with Opus-4.8, our method improves the strict and PoC pass rates by 13.48% and 12.60%, respectively. Under the same GLM-5.2 base model, it outperforms OpenCode by 28.70% in strict pass rate and 29.13% in PoC pass rate, which can be attributed to the advantage of **DoGNAVY-Repair** pipeline. Our method also exhibits strong performance across languages, substantially improving upon OpenCode with GLM-5.2 by 45.90%, 37.61%, and 3.90% on Go, Python, and JavaScript, respectively.

4 Related Works

4.1 Vulnerability Reproduction and PoC Generation

Vulnerability reproduction transforms a vulnerability report and its program context into an executable proof of concept (PoC) that triggers the reported behavior, requiring the system to reconstruct the execution context, reach vulnerable code, satisfy triggering conditions, and validate the result. Existing benchmarks study this capability at different scales. Vul4J provides reproducible Java vulnerabilities with proof-of-vulnerability tests, ARVO offers automatically rebuildable OSS-Fuzz-derived C/C++ vulnerabilities, and CyberGym formulates reproduction as an agent benchmark covering 1,507 real-world vulnerabilities from 188 projects (Bui et al., 2022; Mei et al., 2026; Wang et al., 2025c). Other studies focus on LLM-based PoC generation and reproduction in web or package-level settings, including report-guided generation, runtime validation, proof-of-vulnerability testing, and package exploit generation (Zhao et al., 2025; Liu et al., 2025; Zhao et al., 2026; Duy et al., 2026; Nitin et al., 2025; Simsek et al., 2026). Automated vulnerability reproduction has traditionally relied on program analysis and directed fuzzing, using control/data-flow constraints, taint analysis, LLM-assisted seeds, or joint exploration of inputs and configurations to reach vulnerable states (Lee et al., 2021; Cao et al., 2023; Zhou et al., 2024; Wu et al., 2026). Recent systems increasingly formulate reproduction as an iterative agent process that combines planning, environment reconstruction, program analysis, execution feedback, and validation (Ullah et al., 2025; Liu et al., 2026a; Pu et al., 2026; Zeng et al., 2025; Desai et al., 2026; Gezgin et al., 2026). Our work follows this direction under the CyberGym setting, evaluating whether an autonomous agent can coordinate repository exploration, security knowledge, tool execution, and benchmark validation within a unified workflow.

4.2 Vulnerability Exploitation

Exploitation turns a vulnerability proof-of-concept (PoC) into concrete security impact, making it an inherently dual-use capability. As LLM agents become increasingly capable at coding and autonomous problem solving (Jimenez et al., 2024; Yang et al., 2024; Kwa et al., 2025), benchmarks have evolved from curated CTF challenges such as NYU CTF Bench and Cybench (Shao et al., 2024; Zhang et al., 2025b), to real-world vulnerabilities in CVE-Bench and BountyBench (Zhu et al., 2025; Zhang et al., 2025a), and more recently to difficult low-level exploit development. ExploitGym provides 898 real-world vulnerabilities across userspace software, V8, and the Linux kernel (Wang et al., 2026b), while ExploitBench introduces a fine-grained capability ladder that measures progress from vulnerability triggering to exploit primitives and arbitrary code execution (Lee & Brumley, 2026). Together, they provide complementary large-scale and capability-level evaluations of the transition from vulnerability demonstration to concrete security impact. Exploit development has also become an important component of frontier-model cyber evaluations (Anthropic, 2026a;b; OpenAI, 2026), with newer benchmarks such as ExploitGym and ExploitBench continuing to differentiate frontier models as earlier CTF-style evaluations begin to saturate (Anthropic, 2026c;a; OpenAI, 2026). Meanwhile, agentic systems such as HPTSA, Vulnsage, and Co-RedTeam improve exploitation through hierarchical planning, multi-agent collaboration, execution feedback, refinement, and memory (Zhu et al., 2024; Chen et al., 2026; He et al., 2026). However, comparatively little work has developed purpose-built agents for

the recent, substantially harder pwn-oriented settings of ExploitGym and ExploitBench. Our work directly targets these benchmarks, studying whether agentic scaffolding can improve the transition from a known vulnerability to a working low-level exploit under modern mitigations.

4.3 Vulnerability Repair

Automated vulnerability repair (AVR) aims to remove vulnerable behavior while preserving intended functionality. Its evaluation has evolved from mined vulnerability-fix corpora, such as CVEfixes and BigVul (Bhandari et al., 2021; Fan et al., 2020), to larger datasets with improved coverage and labeling, including DiverseVul, PrimeVul, and MegaVul (Chen et al., 2023a; Ding et al., 2024; Ni et al., 2024). More recent benchmarks, including Vul4J, Vul4C, CVE-Bench, SEC-bench, and PatchEval (Bui et al., 2022; Hu et al., 2025; Wang et al., 2025a; Lee et al., 2026; Wei et al., 2025), provide executable repository-level environments with exploit and regression tests, shifting evaluation from patch matching toward semantic vulnerability repair and behavior-preserving validation. Early AVR methods primarily focus on function-level repair (Harer et al., 2018; Chen et al., 2023b; Fu et al., 2022; Wen et al., 2025), while recent work increasingly targets repository-level repair with coding agents such as SWE-agent and Aider (Yang et al., 2024; Gauthier, 2024) and vulnerability-specific systems such as PatchAgent (Yu et al., 2025). Single-agent approaches improve repair through prompting, vulnerability reasoning, validation feedback, and evidence-guided debugging (Nong et al., 2025; Luo et al., 2025; Nong et al., 2024; Kulsum et al., 2024; Wang et al., 2026a; Kim et al., 2025; Sun et al., 2026; Ye et al., 2026; Hu et al., 2026), while multi-agent methods build on role decomposition (Arora et al., 2024) to coordinate context collection, safety analysis, repair, and broader CVE workflows (Zhang et al., 2026a; Fu et al., 2026; Zhang et al., 2026b; Seo et al., 2026; Zheng et al., 2026). In contrast to approaches that primarily use tool outputs as auxiliary feedback, our multi-agent repair harness integrates structured repository evidence, execution feedback, and independent guidance to support more reliable vulnerability analysis and behavior-preserving repair.

5 Discussion & Conclusion

The empirical results demonstrate the potential of multi-agent architectures for end-to-end cybersecurity automation. **DoGNAVY**'s performance across vulnerability reproduction, exploitation, and repair shows that role-based collaboration, evidence-driven exploration, and security-specific tools can bridge the gap between general LLM capabilities and complex security workflows. However, several limitations remain, including inefficient exploration in large action spaces, degradation in long-context reasoning, insufficiently robust validation of repairs and side effects, and limited generalization caused by reliance on predefined harnesses. These findings point to the need for more adaptive planning, stronger memory mechanisms, and verification methods that enforce semantic security properties.

In this report, we introduced **DoGNAVY**, a framework for automating the vulnerability lifecycle. Beyond its performance, the system demonstrates that decomposing complex security workflows into coordinated agent roles can provide a practical foundation for more capable and systematic AI-assisted security operations. At the same time, the gap between benchmark success and dependable real-world autonomy remains substantial. Closing this gap requires future systems to move beyond task-specific orchestration toward greater robustness, adaptability, and trustworthiness in diverse and evolving security environments.

References

Anomaly. OpenCode: The open source ai coding agent. https://github.com/anomalyco/opencode, 2025. Open-source software.

Anthropic. Claude Code: An agentic coding tool. https://github.com/anthropics/claude-code, 2025. Version 1.0.71, released August 7, 2025.

Anthropic. Claude fable 5 & claude mythos 5 system card, June 2026a. URL https://www-cdn.anthropic.com/d00db56fa754a1b115b6dd7cb2e3c342ee809620.pdf. System card.

Anthropic. Claude opus 5 system card, July 2026b. URL https://www-cdn.anthropic.com/b514064af1408018e64b1ad24e7d5e75850b4ffd/Claude%20Opus%205%20System%20Card.pdf. System card.

Anthropic. Claude sonnet 4.6 system card, February 2026c. URL https://www.anthropic.com/claude-sonnet-4-6-system-card. System card, published February 17, 2026.

All Religions are One. Smashing the stack for fun and profit. 1996. URL https://api.semanticscholar.org/CorpusID:208092205.

Daman Arora, Atharv Sonwane, Nalin Wadhwa, Abhav Mehrotra, Saiteja Utpala, Ramakrishna Bairi, Aditya Kanade, and Nagarajan Natarajan. Masai: Modular architecture for software-engineering ai agents, 2024. URL https://arxiv.org/abs/2406.11638.

Algirdas Avizienis, J. Laprie, Brian Randell, and Carl E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1:11–33, 2004. URL https://api.semanticscholar.org/CorpusID:215753451.

Guru Bhandari, Amara Naseer, and Leon Moonen. CVEfixes: Automated collection of vulnerabilities and their fixes from open-source software. In Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE '21. ACM, 2021. doi: 10.1145/3475960.3475985.

Quang-Cuong Bui, Riccardo Scandariato, and Nicolás E Díaz Ferreyra. Vul4j: A dataset of reproducible java vulnerabilities geared towards the study of program repair techniques. In Proceedings of the 19th International Conference on Mining Software Repositories, pp. 464–468, 2022.

Sicong Cao, Biao He, Xiaobing Sun, Yu Ouyang, Chao Zhang, Xiaoxue Wu, Ting Su, Lili Bo, Bin Li, Chuanlei Ma, et al. Oddfuzz: Discovering java deserialization vulnerabilities via structure-aware directed greybox fuzzing. In 2023 IEEE Symposium on Security and Privacy (SP), pp. 2726–2743. IEEE, 2023.

Siyi Chen, Tianhan Luo, Shijian Wu, Xiangyu Liu, Yilin Zhou, Qi Li, and Wenyuan Xu. A multi-agent framework for automated exploit generation with constraint-guided comprehension and reflection. In Proceedings of the 2026 34th IEEE/ACM International Conference on Program Comprehension, pp. 282–294. Association for Computing Machinery, 2026. doi: 10.1145/3794763.3794817. URL https://doi.org/10.1145/3794763.3794817.

Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David Wagner. DiverseVul: A new vulnerable source code dataset for deep learning based vulnerability detection, 2023a. URL https://arxiv.org/abs/2304.00409.

Zimin Chen, Steve Kommrusch, and Martin Monperrus. Neural transfer learning for repairing security vulnerabilities in c code. IEEE Transactions on Software Engineering, 49(1):147–165, 2023b. doi: 10.1109/TSE.2022.3147265.

Dan Craigen, Nadia Diakun-Thibault, and Randy Purse. Defining cybersecurity. 2014. URL https://api.semanticscholar.org/CorpusID:216086493.

Achintya Desai, Md Shafiuzzaman, Wenbo Guo, and Tevfik Bultan. Program analysis guided llm agent for proof-of-concept generation. arXiv preprint arXiv:2604.07624, 2026. URL https://arxiv.org/abs/2604.07624.

Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David Wagner, Baishakhi Ray, and Yizheng Chen. Vulnerability detection with code language models: How far are we? arXiv preprint arXiv:2403.18624, 2024. URL https://arxiv.org/abs/2403.18624.

Phan The Duy, Khoa Ngo-Khanh, Nguyen Huu Quyen, and Van-Hau Pham. Poc-adapt: Semantic-aware automated vulnerability reproduction with llm multi-agents and reinforcement learning-driven adaptive policy. arXiv preprint arXiv:2604.06618, 2026. URL https://arxiv.org/abs/2604.06618.

Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. A C/C++ code vulnerability dataset with code changes and CVE summaries. In Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20, pp. 508–512. ACM, 2020. doi: 10.1145/3379597.3387501.

Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh Phung. Vulrepair: a t5-based automated software vulnerability repair. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, pp. 935–947, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394130. doi: 10.1145/3540250.3549098. URL https://doi.org/10.1145/3540250.3549098.

Michael Fu, Qiyue Mei, Patanamon Thongtanunam, and Kla Tantithamthavorn. Agenticrepair: Multi-faceted program context engineering for agentic vulnerability repair, 2026. URL https://arxiv.org/abs/2607.29422.

Paul Gauthier. Aider: Ai pair programming in your terminal. https://aider.chat/, 2024. Open-source software available at https://github.com/Aider-AI/aider.

Derin Gezgin, Amartya Das, Shinhae Kim, Zhengdong Huang, Nevena Stojkovic, and Claire Wang. Poc-gym: Towards more reliable llm-assisted proof-of-concept exploit generation. arXiv preprint arXiv:2602.04165, 2026. URL https://arxiv.org/abs/2602.04165.

Jacob A. Harer, Onur Ozdemir, Tomo Lazovich, Christopher P. Reale, Rebecca L. Russell, Louis Y. Kim, and Peter Chin. Learning to repair software vulnerabilities with generative adversarial networks. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, pp. 7944–7954, Red Hook, NY, USA, 2018. Curran Associates Inc.

Pengfei He, Ash Fox, Lesly Miculicich, Stefan Friedli, Daniel Fabian, Burak Gokturk, Jiliang Tang, Chen-Yu Lee, Tomas Pfister, and Long T. Le. Co-RedTeam: Orchestrated security discovery and exploitation with LLM agents. arXiv preprint arXiv:2602.02164, 2026. doi: 10.48550/arXiv.2602.02164. URL https://arxiv.org/abs/2602.02164.

Haichuan Hu, Guoqing Xie, Quanjun Zhang, Jiawei Liu, Shengcheng Yu, Chunrong Fang, Zhenyu Chen, and Liang Xiao. Evorepair: Enhancing vulnerability repair agents through experience-based self-evolution, 2026. URL https://arxiv.org/abs/2605.30105.

Yiwei Hu, Zhen Liu, Kedie Shu, Shenghua Guan, Deqing Zou, Shouhuai Xu, Bin Yuan, and Hai Jin. SoK: Automated vulnerability repair: Methods, tools, and assessments. In 34th USENIX Security Symposium (USENIX Security 25), pp. 4421–4440. USENIX Association, 2025. URL https://www.usenix.org/conference/usenixsecurity25/presentation/hu-yiwei.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Pengfei Jing, Meng-Tian Tang, Xiaorong Shi, Xing Zheng, Sen Nie, Shi Wu, Yong Yang, and Xiapu Luo. Secbench: A comprehensive multi-dimensional benchmarking dataset for llms in cybersecurity. ArXiv, abs/2412.20787, 2024. URL https://api.semanticscholar.org/CorpusID:275134254.

Youngjoon Kim, Sunguk Shin, Hyoungshick Kim, and Jiwon Yoon. Logs in, patches out: Automated vulnerability repair via Tree-of-Thought LLM analysis. In 34th USENIX Security Symposium (USENIX Security 25), pp. 4401–4419, Seattle, WA, August 2025. USENIX Association. ISBN 978-1-939133-52-6. URL https://www.usenix.org/conference/usenixsecurity25/presentation/kim-youngjoon.

Ummay Kulsum, Haotian Zhu, Bowen Xu, and Marcelo d'Amorim. A case study of llm for automated vulnerability repair: Assessing impact of reasoning and patch validation feedback, 2024. URL https://doi.org/10.1145/3664646.3664770.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI ability to complete long tasks. arXiv preprint arXiv:2503.14499, 2025. URL https://arxiv.org/abs/2503.14499.

Gwangmu Lee, Woochul Shim, and Byoungyoung Lee. Constraint-guided directed greybox fuzzing. In 30th USENIX Security Symposium (USENIX Security 21), pp. 3559–3576, 2021.

Hwiwon Lee, Ziqi Zhang, Hanxiao Lu, and Lingming Zhang. Sec-bench: Automated benchmarking of llm agents on real-world software security tasks. Advances in Neural Information Processing Systems, 38: 116342–116378, 2026.

Seunghyun Lee and David Brumley. Exploitbench: A capability ladder benchmark for LLM cybersecurity agents. arXiv preprint arXiv:2605.14153, 2026. doi: 10.48550/arXiv.2605.14153. URL https://arxiv.org/abs/2605.14153.

Bin Liu, Yanjie Zhao, Guoai Xu, and Haoyu Wang. Llm agents for automated web vulnerability reproduction: Are we there yet? arXiv preprint arXiv:2510.14700, 2025. URL https://arxiv.org/abs/2510.14700.

Bin Liu, Yanjie Zhao, Zhenpeng Chen, Guoai Xu, and Haoyu Wang. A dual-loop agent framework for automated vulnerability reproduction. arXiv preprint arXiv:2602.05721, 2026a. URL https://arxiv.org/abs/2602.05721.

Dongrui Liu, Yu Li, Zhonghao Yang, Peng Wang, Guan-Lin Chen, Yuejin Xie, Qinghua Mao, Wanying Qu, Yanxu Zhu, Tianyi Zhou, Lei Yuan, Zhijie Zheng, Qihao Lin, Yiming Wang, Haoyu Luo, Shuai Shao, Chen Qian, Qingyu Liu, Ling Tang, Ruiyang Qin, Qihan Ren, Junxiao Yang, Kun Wang, Zhiheng Xi, Linfeng Zhang, Ranjie Duan, Bo Zhang, Wenjie Wang, Wen Shen, Qiaosheng Zhang, Yan Teng, Chaochao Lu, Rui Mei, Man Li, Jialing Tao, Xi Lin, Tianhang Zheng, Yong Liu, Quanshi Zhang, Lei Zhu, Xingjun Ma, Junhua Liu, Hui Xue, X.Z. Zuo, Xiangnan He, Chaoyi Shen, Xianglong Liu, Min Huang, Jing Shao, and Xia Hu. Agentdog 1.5: A lightweight and scalable alignment framework for ai agent safety and security. 2026b. URL https://api.semanticscholar.org/CorpusID:288743271.

Dongrui Liu, Qihan Ren, Chen Qian, Shuai Shao, Yuejin Xie, Yu Li, Zhonghao Yang, Haoyu Luo, Peng Wang, Qingyu Liu, Bin Hu, Ling Tang, Jilin Mei, Dadi Guo, Lei Yuan, Junyao Yang, Guanxu Chen, Qihao Lin, Yi Yu, Bo Zhang, Jiaxuan Guo, Jie Zhang, Wenqi Shao, Huiqi Deng, Zhiheng Xi, Wenjie Wang, Wenxuan Wang, Wen Shen, Zhikai Chen, Haoyu Xie, Jialing Tao, Juntao Dai, Jiaming Ji, Zhongjie Ba, Linfeng Zhang, Yong Liu, Quanshi Zhang, Lei Zhu, Zhihua Wei, Hui Xue, Chaochao Lu, Jing Shao, and Xia Hu. Agentdog: A diagnostic guardrail framework for ai agent safety and security. 2026c. URL https://api.semanticscholar.org/CorpusID:285051714.

Yining Luo, Baobao Li, Anoop Singhal, Peiyu Tseng, Lan Zhang, Qingtian Zou, Xiaoyan Sun, and Peng Liu. Exploring prompt patterns for effective vulnerability repair in real-world code by large language models. In Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics, IWSPA '25, pp. 23–33, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715013. doi: 10.1145/3716815.3729010. URL https://doi.org/10.1145/3716815.3729010.

Robert A. Martin and Sean Barnum. Common weakness enumeration (cwe) status update. ACM Sigada Ada Letters, pp. 88–91, 2008. URL https://api.semanticscholar.org/CorpusID:2110699.

Xiang Mei, Jordi Del Castillo, Pulkit Singh Singaria, Haoran Xi, Abdelouahab Benchikh, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, Hammond Pearce, et al. Arvo: Atlas of reproducible vulnerabilities for open-source software. In 2026 IEEE 11th European Symposium on Security and Privacy (EuroS&P), pp. 860–873. IEEE, 2026.

Chao Ni, Liyu Shen, Xiaohu Yang, Yan Zhu, and Shaohua Wang. MegaVul: A C/C++ vulnerability dataset with comprehensive code representations. In Proceedings of the 21st International Conference on Mining Software Repositories, MSR '24, pp. 738–742, 2024. doi: 10.1145/3643991.3644886.

Vikram Nitin, Baishakhi Ray, and Roshanak Zilouchian Moghaddam. Faultline: Automated proof-of-vulnerability generation using llm agents. arXiv preprint arXiv:2507.15241, 2025. URL https://arxiv.org/abs/2507.15241.

Yu Nong, Mohammed Aldeen, Long Cheng, Hongxin Hu, Feng Chen, and Haipeng Cai. Chain-of-thought prompting of large language models for discovering and fixing software vulnerabilities, 2024. URL https://arxiv.org/abs/2402.17230.

Yu Nong, Haoran Yang, Long Cheng, Hongxin Hu, and Haipeng Cai. APPATCH: Automated adaptive prompting large language models for Real-World software vulnerability patching. In 34th USENIX Security Symposium (USENIX Security 25), pp. 4481–4500, Seattle, WA, August 2025. USENIX Association. ISBN 978-1-939133-52-6. URL https://www.usenix.org/conference/usenixsecurity25/presentation/nong.

OpenAI. GPT-5.6 system card, July 2026. URL https://deploymentsafety.openai.com/gpt-5-6. System card, published July 9, 2026.

Juefei Pu, Xingyu Li, Zhengchuan Liang, Jonathan Cox, Yifan Wu, Kareem Shehada, Arrdya Srivastav, and Zhiyun Qian. Patch-to-poc: A systematic study of agentic llm systems for linux kernel n-day reproduction. arXiv preprint arXiv:2602.07287, 2026. URL https://arxiv.org/abs/2602.07287.

Minjae Seo, Wonwoo Choi, Seungwon Shin, and Myoungsung You. Autopatch: Multi-agent framework for patching real-world cves generated by outdated llms. SSRN, 2026.

Minghao Shao, Sofija Jancheska, Meet Udeshi, Brendan Dolan-Gavitt, Haoran Xi, Kimberly Milner, Boyuan Chen, Max Yin, Siddharth Garg, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Muhammad Shafique. NYU CTF Bench: A scalable open-source benchmark dataset for evaluating LLMs in offensive security. In Advances in Neural Information Processing Systems, volume 37, pp. 57472–57498, 2024. doi: 10.52202/079017-1832. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/69d97a6493fbf016fff0a751f253ad18-Abstract-Datasets_and_Benchmarks_Track.html.

Tianneng Shi, Robin Rheem, Dongwei Jiang, Mona Wang, Francisco De, La Riega, Zhun Wang, Jingzhi Jiang, Alexander Cheung, S. Ta'i, Jonah Cha, Jianhong Tu, Gab One Han, Chenguang Wang, Wenbo Guo, Jingxuan He, Dawn Xiaodong Song, and Uc Berkeley. Cybergym-e2e: Scalable real-world benchmark for ai agents' end-to-end cybersecurity capabilities. ArXiv, abs/2606.04460, 2026. URL https://api.semanticscholar.org/CorpusID:288107481.

Deniz Simsek, Aryaz Eghbali, and Michael Pradel. Pocgen: Generating proof-of-concept exploits for vulnerabilities in npm packages. Proceedings of the ACM on Software Engineering, 3(FSE):3887–3908, June 2026. doi: 10.1145/3808178. URL https://doi.org/10.1145/3808178.

Maolin Sun, Yibiao Yang, Xuanlin Liu, Yuming Zhou, and Baowen Xu. Debugharness: Emulating human dynamic debugging for autonomous program repair, 2026. URL https://arxiv.org/abs/2604.03610.

László Szekeres, Mathias Payer, Tao Wei, and Dawn Xiaodong Song. Sok: Eternal war in memory. 2013 IEEE Symposium on Security and Privacy, pp. 48–62, 2013. URL https://api.semanticscholar.org/CorpusID:2937041.

Saad Ullah, Praneeth Balasubramanian, Wenbo Guo, Amanda Burnett, Hammond Pearce, Christopher Kruegel, Giovanni Vigna, and Gianluca Stringhini. From cve entries to verifiable exploits: An automated multi-agent framework for reproducing cves. arXiv preprint arXiv:2509.01835, 2025. URL https://arxiv.org/abs/2509.01835.

Rossouw von Solms and Johan F. Van Niekerk. From information security to cyber security. Comput. Secur., 38:97–102, 2013. URL https://api.semanticscholar.org/CorpusID:13164363.

Hulin Wang, Zion Leonahenahe Basque, Jie Hu, Ati Priya Bajaj, Yibo Liu, Samuel Zhu, Giorgi Kobakhia, Nikhil Chapre, Will Rosenberg, Siddharth Mishra, Aditya Maheshbhai Gabani, Moritz Schloegel, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang, and Tiffany Bao. Root-cause-driven automated vulnerability repair, 2026a. URL https://arxiv.org/abs/2605.04251.

Peiran Wang, Xiaogeng Liu, and Chaowei Xiao. CVE-bench: Benchmarking LLM-based software engineering agent's ability to repair real-world CVE vulnerabilities. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4207–4224. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.naacl-long.212. URL https://aclanthology.org/2025.naacl-long.212/.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In International Conference on Learning Representations, volume 2025, pp. 65882–65919, 2025b.

Zhun Wang, Tianneng Shi, Jingxuan He, Michelle Cai, Jialin Zhang, and Dawn Xiaodong Song. Cybergym: Evaluating ai agents'real-world cybersecurity capabilities at scale. 2025c. URL https://api.semanticscholar.org/CorpusID:279119190.

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. Exploitgym: Can AI agents turn security vulnerabilities into real attacks? arXiv preprint arXiv:2605.11086, 2026b. doi: 10.48550/arXiv.2605.11086. URL https://arxiv.org/abs/2605.11086.

Zichao Wei, Jun Zeng, Ming Wen, Zeliang Yu, Kai Cheng, Yiding Zhu, Jing Guo, Shiqi Zhou, Le Yin, Xi Su, and Zhechao Ma. Patcheval: A new benchmark for evaluating llms on patching real-world vulnerabilities. ArXiv, abs/2511.11019, 2025. URL https://api.semanticscholar.org/CorpusID:283054865.

Xin-Cheng Wen, Zirui Lin, Yijun Yang, Cuiyun Gao, and Deheng Ye. Vul-r2: A reasoning llm for automated vulnerability repair. In 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 26–38. IEEE Press, 2025. doi: 10.1109/ASE63991.2025.00011. URL https://doi.org/10.1109/ASE63991.2025.00011.

Susheng Wu, Xin Hu, Yiheng Cao, Zhuotong Zhou, Yiheng Huang, Yijian Wu, Bihuan Chen, Zhijia Zhao, and Xin Peng. It takes two: Option-aware directed greybox fuzzing for vulnerability poc generation. Proceedings of the ACM on Software Engineering, 3(FSE):2189–2210, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), Advances in Neural Information Processing Systems, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

Zhenlei Ye, Xiaobing Sun, Sicong Cao, Lili Bo, and Bin Li. Well begun is half done: Location-aware and trace-guided iterative automated vulnerability repair. In Proceedings of the IEEE/ACM 48th International Conference on Software Engineering, 2026. To appear.

Zheng Yu, Ziyi Guo, Yuhang Wu, Jiahao Yu, Meng Xu, Dongliang Mu, Yan Chen, and Xinyu Xing. PATCHAGENT: A practical program repair agent mimicking human expertise. In 34th USENIX Security Symposium (USENIX Security 25), pp. 4381–4400, Seattle, WA, August 2025. USENIX Association. ISBN 978-1-939133-52-6. URL https://www.usenix.org/conference/usenixsecurity25/presentation/yu-zheng.

Z.AI. GLM-5.2: Built for long-horizon tasks. https://z.ai/blog/glm-5.2, 2026a. Official model blog. Accessed: 2026-08-18.

Z.AI. GLM-5.3: Frontier coding with emergent cyber capabilities. https://z.ai/blog/glm-5.3, 2026b. Official model blog. Accessed: 2026-08-18.

Haochen Zeng, Andrew Bao, Jiajun Cheng, and Chengyu Song. Pbfuzz: Agentic directed fuzzing for pov generation. arXiv preprint arXiv:2512.04611, 2025. URL https://arxiv.org/abs/2512.04611.

Andy K. Zhang, Joey Ji, Celeste Menders, Riya Dulepet, Thomas Qin, Ron Yifeng Wang, Junrong Wu, Kyleen Liao, Jiliang Li, Jinghan Hu, Sara Hong, Nardos Demilew, Shivatmica Murgai, Jason Khiem Tran, Nishka Kacheria, Ethan Jun-shen Ho, Denis Liu, Lauren McLane, Olivia Beyer Bruvik, Dai-Rong

Han, Seungwoo Kim, Akhil Vyas, Cuiyuanxiu Chen, Ryan Li, Weiran Xu, Jonathan Z. Ye, Prerit Choudhary, Siddharth M. Bhatia, Vikram Sivashankar, Yuxuan Bao, Dawn Song, Dan Boneh, Daniel E. Ho, and Percy Liang. Bountybench: Dollar impact of AI agent attackers and defenders on real-world cybersecurity systems. In Advances in Neural Information Processing Systems, volume 38, pp. 190575–190694, 2025a. doi: 10.52202/085713-5725. URL https://papers.nips.cc/paper_files/paper/2025/hash/faed4276b52ef762879db4142655c699-Abstract-Datasets_and_Benchmarks_Track.html.

Andy K. Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, Pura Peetathawatchai, Ari Glenn, Vikram Sivashankar, Daniel Zamoshchin, Leo Glikbarg, Derek Askaryar, Haoxiang Yang, Aolin Zhang, Rishi Alluri, Nathan Tran, Rinnara Sangpisit, Kenny Oseleononmen, Dan Boneh, Daniel E. Ho, and Percy Liang. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. In The Thirteenth International Conference on Learning Representations, 2025b. URL https://openreview.net/forum?id=tc90LV0yRL.

Mingming Zhang, Xu Wang, Jian Zhang, Xiangxin Meng, Jiayi Zhang, and Chunming Hu. Vulnresolver: A hybrid agent framework for llm-based automated vulnerability issue resolution, 2026a. URL https://arxiv.org/abs/2601.13933.

Quanjun Zhang, Chengyu Gao, Yu Han, Ye Shang, Chunrong Fang, Zhenyu Chen, and Liang Xiao. Sgagent: Suggestion-guided llm-based multi-agent framework for repository-level software repair. ACM Trans. Softw. Eng. Methodol., May 2026b. ISSN 1049-331X. doi: 10.1145/3818617. URL https://doi.org/10.1145/3818617. Just Accepted.

Mengyao Zhao, Kaixuan Li, Lyuye Zhang, Wenjing Dang, Chenggong Ding, Sen Chen, and Zheli Liu. A systematic study on generating web vulnerability proof-of-concepts using large language models. arXiv preprint arXiv:2510.10148, 2025. URL https://arxiv.org/abs/2510.10148.

Zijie Zhao, Chenyuan Yang, Weidong Wang, Yihan Yang, Ziqi Zhang, and Lingming Zhang. Anypoc: Universal proof-of-concept test generation for scalable llm-based bug detection. arXiv preprint arXiv:2604.11950, 2026. URL https://arxiv.org/abs/2604.11950.

Zelong Zheng, Jiayuan Zhou, Xing Hu, Yi Gao, and Shengyi Pan. Multi-agent end-to-end vulnerability management for mitigating recurring vulnerabilities, 2026.

Zhuotong Zhou, Yongzhuo Yang, Susheng Wu, Yiheng Huang, Bihuan Chen, and Xin Peng. Magneto: A step-wise approach to exploit vulnerabilities in dependent libraries via llm-empowered directed fuzzing. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, pp. 1633–1644, 2024.

Yuxuan Zhu, Antony Kellermann, Akul Gupta, Philip Li, Richard Fang, Rohan Bindu, and Daniel Kang. Teams of LLM agents can exploit zero-day vulnerabilities. arXiv preprint arXiv:2406.01637, 2024. doi: 10.48550/arXiv.2406.01637. URL https://arxiv.org/abs/2406.01637.

Yuxuan Zhu, Antony Kellermann, Dylan Bowman, Philip Li, Akul Gupta, Adarsh Danda, Richard Fang, Conner Jensen, Eric Ihli, Jason Benn, Jet Geronimo, Avi Dhir, Sudhit Rao, Kaicheng Yu, Twm Stone, and Daniel Kang. CVE-bench: A benchmark for AI agents' ability to exploit real-world web application vulnerabilities. In Proceedings of the 42nd International Conference on Machine Learning, volume 267 of Proceedings of Machine Learning Research, pp. 79850–79867. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhu25i.html.