# AI Slop and Hallucinations in Vulnerability Assessment: A Survey on Reasoning Failures and Trustworthy Mitigation

Junchen Ding[1,2⋆], Jialiang Dong[2*], Yichen Zhu[3], Yi Liu[4], Gelei Deng[5], Willy Susilo[2], Siqi Ma[2], and Yuekang Li[1⋆⋆]

[1] University of New South Wales, High St, Kensington NSW 2052, Australia
{junchen.ding,yuekang.li}@unsw.edu.au
[2] University of Wollongong, Northfields Ave, Wollongong NSW 2500 Australia
[3] Nanjing University of Information Science & Technology, 219 Ningliu Road, Nanjing, Jiangsu, China
[4] Griffith University, Parklands Drive, Southport QLD 4222 Australia
[5] Nanyang Technological University, 50 Nanyang Ave, Singapore 639798, Singapore

**Abstract.** The integration of Large Language Models (LLMs) into cybersecurity has transformed vulnerability assessment, but it has also produced a trustworthiness crisis driven by the unchecked proliferation of "AI slop." These artifacts, hallucinated vulnerabilities, plausible but incorrect patches, and semantically repackaged bug reports, impose a cognitive burden on human triage pipelines that mirrors a denial-of-service attack. This paper surveys the empirical evidence, identifies a unifying mechanism, and traces a path toward trustworthy triage. We formalize a taxonomy of AI slop grounded in a structured literature review and dissect its root cause: the gap between the causal deductive reasoning of security experts and the autoregressive probabilistic generation of current LLMs. We operationalize this gap through a measurable proxy, the Deductive Coverage Score, and show that chain-of-thought prompting and tool-using agents narrow but do not close it. We review mitigation strategies and argue that passive detection and watermarking target provenance rather than correctness, facing fundamental entropy constraints. We instead advocate for active neuro-symbolic verification, mapping each pipeline component to prior systems with documented limits on security inputs. Finally, we specify two evaluation instruments, CVE-Bench and Slop-Score, including dataset construction, metric formulas, and anti-gaming provisions. By shifting evaluation from linguistic fluency to mathematical verifiability, this survey provides a roadmap for securing emerging AI-driven triage systems.

**Keywords:** AI Slop · Vulnerability Assessment · Trustworthy AI · Large Language Models · Neuro-Symbolic Verification · Content Provenance

⋆ Equal contribution.
⋆⋆ Corresponding author.

## 1 Introduction

The integration of Large Language Models (LLMs) into cybersecurity is reshaping vulnerability assessment, from automated detection to patch generation and threat intelligence [13]. Software auditing was historically grounded in deterministic methodologies. Security analysts relied on rule-based Static Application Security Testing (SAST) tools and manual code inspection to identify memory corruption, logic flaws, and architectural weaknesses. These approaches offered formal assurance but were labor-intensive and often insufficient for capturing the context-sensitive semantics of modern software [5]. With the emergence of foundation models exhibiting strong natural language understanding, the field has shifted toward AI-assisted auditing [22]. This transition reflects expectations that Artificial General Intelligence (AGI) may accelerate vulnerability discovery while reducing human effort [28].

This transition has introduced a structural challenge. As LLM-based security agents become embedded in analysis pipelines, the ecosystem is experiencing a growing influx of what we term "AI Slop." We define AI slop as security-relevant artifacts that exhibit high linguistic fluency and structural plausibility while lacking semantic validity or executable grounding [11]. Unlike traditional false positives, which arise from conservative rule-based analysis, these artifacts are difficult to invalidate through superficial inspection. They imitate the form of rigorous reasoning while remaining detached from the constraints that would make such reasoning correct. In practice, this includes fabricated CWEs, hallucinated inter-procedural execution paths, and patches that appear reasonable but fail to resolve the underlying vulnerability [12].

The impact is no longer theoretical. Open-source maintainers, bug bounty platforms, and CVE assignment authorities are processing large volumes of syntactically well-formed but semantically unreliable submissions. A representative case is the *cURL* project. Daniel Stenberg, the project's founder and maintainer since 1996 [35, 34], reported in early 2024 that the project's security workflow was increasingly disrupted by AI-generated vulnerability submissions [27]. By mid-2025, roughly 20% of reports submitted through the HackerOne Bug Bounty Program were identified as low-quality AI slop, while valid reports declined to around 5%. In January 2026, Stenberg announced the discontinuation of the program, citing the need to remove incentives for poorly substantiated submissions [29]. This case illustrates how plausible but unverifiable artifacts can degrade the sustainability of human-centered triage.

At the core of this phenomenon lies a divergence between how humans and current AI systems produce and validate knowledge. Human security experts rely on causal, multi-step deductive reasoning. They construct internal models of system behavior, trace data flow across procedural boundaries, and verify exploitability under explicit constraints [33]. Autoregressive LLMs operate through probabilistic token generation. Their outputs are shaped by statistical correlations rather than execution-grounded verification. When required information exceeds the model's effective reasoning horizon, the generation process defaults to statistically plausible continuation. AI slop is a systematic byproduct of this

mismatch, manifesting as hallucinated vulnerability detection, incorrect patch synthesis, and semantic repackaging of existing knowledge.

Prior research has largely focused on improving model accuracy or applying post hoc detection techniques. Such approaches remain limited when the generation process itself is not grounded in verification. There is a need for architectures that treat generated outputs as hypotheses subject to deterministic validation rather than authoritative conclusions.

This paper sits at a deliberate intersection. It is a survey in that it systematizes existing empirical evidence. It is a conceptual analysis in that it identifies a unifying mechanism behind seemingly disparate failures. It is a roadmap in that it traces the path toward trustworthy triage. The contributions are fourfold:

- We define and formalize AI slop in the context of vulnerability assessment, and propose a taxonomy that captures its major manifestations, including hallucinated vulnerabilities, incorrect patch synthesis, and semantic repackaging.
- We analyze the root cause of these phenomena through the lens of the reasoning gap between human deductive processes and probabilistic generation in LLMs.
- We review existing mitigation strategies and examine their limitations, with particular attention to the failure of passive statistical detection and the constraints of watermarking in low-entropy security domains.
- We argue for the necessity of active verification and neuro-symbolic architectures, and outline key open challenges for building trustworthy, human-aligned triage systems.

The remainder of this paper is organized as follows. Section 2 introduces the background of LLM-assisted vulnerability assessment and formalizes the concept of AI slop. Section 3 presents the taxonomy and supporting literature. Section 4 analyzes the divergence between human reasoning and model generation. Section 5 discusses mitigation strategies with a focus on verification. Section 6 outlines open challenges and future directions. Section 7 concludes the paper.

## 2 Background

To situate the emergence of AI slop and the resulting trustworthiness concerns, it is necessary to examine three interconnected components: the role of LLMs in contemporary vulnerability assessment, the operational dynamics of vulnerability reporting ecosystems, and the underlying differences in reasoning that shape the behavior of these systems.

### 2.1 LLMs in Vulnerability Assessment

The traditional vulnerability assessment lifecycle, spanning code auditing, static and dynamic analysis, and patch generation, has relied on expert-driven workflows demanding familiarity with programming languages, system architectures,

and the ability to reason about execution under implicit constraints. The introduction of LLMs has altered this landscape. Pre-trained on extensive corpora of source code, documentation, and vulnerability reports, models such as ChatGPT, Claude, Gemini, and specialized code-oriented LLMs are increasingly incorporated into security workflows [9]. Their utility lies in interpreting code through a semantic lens, bridging natural language descriptions and program behavior.

In practice, LLMs are employed across semantic anomaly detection, vulnerability explanation, exploit sketching, and automated patch suggestion. Compared to SAST tools that depend on rule-based abstract syntax trees and taint propagation, LLMs operate at a higher level of abstraction, reasoning over code structure, naming conventions, and developer intent [22]. This enables them to surface patterns difficult to capture through purely syntactic analysis. At the same time, their outputs are derived from probabilistic inference over learned distributions rather than execution or formal verification. Their behavior is sensitive to prompt formulation, context availability, and token-level correlations. When reasoning extends beyond local patterns or requires maintaining consistency across multiple steps, these models lack mechanisms to enforce correctness.

### 2.2 The Vulnerability Reporting Ecosystem and Triage Bottlenecks

Modern cybersecurity practice depends on distributed vulnerability discovery and standardized reporting infrastructures. Central to this ecosystem are the CVE database maintained by MITRE and bug bounty programs hosted on platforms including HackerOne and Bugcrowd [8]. Within this pipeline, triage serves as the primary point of control. Human analysts evaluate incoming reports, verify exploitability, assess impact, and eliminate duplicates or spurious claims.

Historically, linguistic clarity and structural coherence functioned as useful heuristics. Well-written reports often correlated with careful analysis and reproducible findings. This implicit signal has been weakened by the widespread availability of LLMs. Both inexperienced contributors and adversarial actors can now generate reports conforming to established conventions, drawing on templates, common vulnerability patterns, and superficial indicators such as error logs or dependency warnings [7].

The problem is compounded by pre-existing gaps in the disclosure ecosystem. Dong et al. (2025) conducted a large-scale empirical study of "silent" vulnerability fixes in open-source software, security-relevant code changes that were never assigned a CVE, never disclosed through coordinated channels, and often lacked any accompanying security advisory [6]. Their analysis revealed that a substantial fraction of real vulnerability remediation occurs entirely outside the visible reporting infrastructure. When an LLM repackages one of these unreported fixes into a "novel" vulnerability report, there may be no public record to contradict it. The ground truth was never published, so the repackaged claim inherits an unearned veneer of plausibility. This fragmentation makes semantic repackaging harder to detect and gives AI-generated claims a structural advantage.

The bottleneck has shifted from identifying obviously flawed reports to determining whether a well-structured submission corresponds to an actual vulnerability. This increases the cognitive load on analysts who must reconstruct or refute the implicit reasoning behind each claim.

### 2.3 Formalizing "AI Slop": The Reasoning Gap

A conventional false positive typically arises from the conservative nature of deterministic tools, which may flag benign constructs due to incomplete context or overly broad rules. AI slop reflects a different failure mode. We define **AI Slop** in cybersecurity as *automatically generated security artifacts, such as reports, proofs-of-concept, or patches, that exhibit high linguistic fluency and plausible syntactic structure, but suffer from catastrophic semantic failures due to hallucinated technical facts or disconnected logical inferences.*

This distinction is rooted in how reasoning is carried out. A human security researcher evaluates a vulnerability through causal and multi-step deductive reasoning, constructing a working model of the system, tracing data flow across functions, and verifying that all conditions for exploitation can be satisfied [33]. AR LLMs generate text by estimating the likelihood of token sequences conditioned on prior context. When tasked with vulnerability analysis, they assemble outputs resembling known reporting patterns. If critical information is missing or exceeds the model's effective reasoning capacity, the gap is resolved through statistically plausible continuation rather than explicit verification [11]. The result is a systematic divergence between appearance and validity. Outputs may mirror the structure of rigorous analysis but lack a consistent execution path that supports their claims.

## 3 Literature Review: A Taxonomy of AI Slop

### 3.1 Literature Search Methodology

To ground our taxonomy, we conducted a systematic literature review across IEEE Xplore, ACM Digital Library, and arXiv, querying combinations of LLM and security-related keywords. Two independent reviewers screened the deduplicated results, retaining studies that provided empirical evidence of LLM failure modes in security contexts rather than relying solely on aggregate metrics. After full-text filtering for relevance and experimental rigor, we categorized the documented failures into three branches, supported by representative works in Table 1.

Figure 1 presents a structure the field has lacked. What appeared as scattered failure cases organizes itself into a pattern. AI slop is not a single defect. It is a family of behaviors that emerge when generation is mistaken for reasoning. The taxonomy exposes how they evolve, reinforce each other, and erode trust in vulnerability intelligence.

Most existing evaluations reward syntactic validity and functional success while overlooking artifacts that feel convincing but remain logically hollow. The

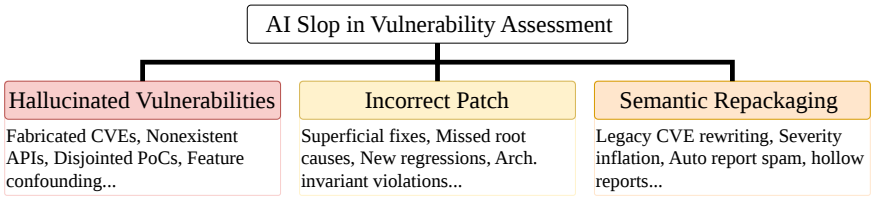


**Fig. 1.** Hierarchical classification of deceptive AI-generated artifacts in security triage. Dashed sub-branches indicate failure modes identified through the review but not yet extensively studied in isolation.

taxonomy reframes the problem by asking whether an answer carries verifiable substance. Three distinct branches emerge, each representing a different way the model drifts from grounded reasoning. Table 1 anchors these categories in empirical evidence.

### 3.2 Hallucinated Vulnerabilities and Feature Confounding

The first branch is the most direct. The model does not merely misinterpret a vulnerability. It invents one. Fabricated CVEs, nonexistent APIs, and disjointed exploit paths appear with confidence mirroring genuine analysis.

Ullah et al. (2024) showed that when presented with benign code, models still produce vulnerability reports, as if compelled to satisfy the expectation embedded in the prompt [31]. The model does not verify whether a flaw exists. It assumes one should exist and proceeds to construct it. Zhang et al. (2025) revealed how fragile this process is. By altering variable names and surface syntax while preserving semantics, the model's judgment shifts dramatically [38]. What changes is not the logic of the program but the statistical signals the model relies on. Chakraborty et al. (2022) traced this to its roots: neural models learn patterns correlating with vulnerabilities that reside in superficial syntax rather than execution semantics [4]. When scaled to modern language models, tokens such as `malloc` or `memcpy` act as triggers, nudging the model toward familiar vulnerability narratives even when the underlying code is safe.

Two additional failure modes in this branch deserve mention even though they lack dedicated studies in the security domain. First, prompt-injection hallucination occurs when adversarial or ambiguous prompt phrasing overrides the user's stated intent, causing the model to fabricate threat models or attack surfaces that no reasonable reading of the input would justify. This phenomenon has been documented in general code generation but not systematically studied under security-specific prompt conditions. Second, tool-use confabulation arises when autonomous security agents invoke external tools (compilers and analyzers) and then misinterpret or fabricate the tool's output, producing analyses that carry the *appearance* of tool verification without its substance. Pearce et al. observed this behavior in AI agents tasked with autonomous security auditing, where incorrect tool arguments and misinterpreted structured output produced

analyses that looked verified but were not [22]. These sub-branches, shown as dashed in Figure 1, represent known but under-studied failure modes that the taxonomy identifies as priorities for future work.

### 3.3 Plausible but Incorrect Patch Synthesis and Regression Introduction

The second branch shifts from detection to repair. The model attempts to solve the problem, yet the solution reveals a different fragility. Patches address what is visible while ignoring what is essential. They compile and pass superficial checks but fail to resolve the underlying issue or introduce new inconsistencies.

Pearce et al. (2023) showed that zero-shot repair frequently produces patches treating symptoms rather than causes [23]. A null check is added, an exception handled, yet the deeper flaw remains untouched. Xia et al. (2024) extended this into interactive settings, where iterative dialogue does not converge toward correctness but cycles through variations of the same flawed reasoning [36]. At the repository level, SWE-bench exposed the difficulty of maintaining consistency across multiple files and hidden invariants [12]. The model operates locally, making decisions valid in isolation that conflict within the broader architecture.

A further failure mode amplifies this problem. Pearce et al. (2023) demonstrated that LLM-generated code, even when functionally correct, introduces security weaknesses at a rate comparable to inexperienced developers [22]. Transposed to patch synthesis, a patch that compiles and passes a unit test may embed a new CWE, trading one vulnerability for another. This regression-introduction failure is invisible to functional benchmarks but directly relevant to security contexts. Table 1 reveals a common thread: the model struggles to preserve constraints not explicitly stated, producing patches that look correct because they align with familiar patterns rather than being verified against the system as a whole.

**Table 1.** Comparative analysis of representative works on AI slop and reasoning failures. **H** = Hallucination, **IP** = Incorrect Patch, **RS** = Report Spam.

| Reference | Cat. | Failure Subtype | Key Reasoning Failure | Grounding |
|---|---|---|---|---|
| Ullah et al. [31] | H | Sycophantic halluc. | Hallucinates vulns in safe code to satisfy prompts | None |
| Zhang et al. [38] | H | Token-bias confound. | Judgment shifts with surface renaming, not logic | None |
| Chakraborty et al. [4] | H | Spurious feature learn. | Learns syntactic artifacts, not exec. semantics | Partial |
| Pearce et al. [22] | H | Tool-use confab. | Agents misinterpret tool output in security tasks | Tool logs |
| Pearce et al. [23] | IP | Symptom-only repair | Fixes surface, misses root cause, adds regressions | Compile |
| Xia et al. [36] | IP | Repair loop drift | Plausible fixes silently break arch. state | Tests |
| Jimenez et al. [12] | IP | Cross-file invariant loss | Cannot maintain invariants across files | Tests |
| Pearce et al. [22] | IP | Security regress. intro. | Generated patches embed new CWEs | Static analysis |
| Kabir et al. [14] | RS | Fluency-as-credibility | Coherent structure masks technical errors | User study |
| Stenberg [29] | RS | Automated flooding | Plausible narratives without PoC validation | Triage logs |
| Dong et al. [6] | RS | Undisclosed-fix repackage | Silent fixes provide undetectable raw material | Commit diffs |

### 3.4 Semantic Repackaging and Report Spam

The third branch emerges from deliberate use. Language models can reshape existing information into forms that appear original and authoritative. Legacy vulnerabilities are rewritten, minor issues inflated, and large volumes of reports generated with minimal grounding.

Kabir et al. (2024) showed that readers consistently prefer AI-generated explanations due to clarity and structure, even when those explanations are incorrect [14]. Fluency becomes a proxy for credibility, and what once served as a useful heuristic in triage becomes a point of exploitation. Stenberg documented the impact on *cURL*, where automated submissions flood the pipeline with well-structured but unverifiable reports lacking functional proofs of concept [29].

The silent vulnerability fixes documented by Dong et al. (2025) deepen this picture [6]. A substantial portion of real remediation never enters the public record. An LLM that has ingested commit diffs from open-source repositories may "discover" a vulnerability that was silently fixed years ago, package it as a novel finding, and submit it to a bug bounty platform. Because no CVE was assigned, there is no public contradiction. The slop is undetectable by any lookup-based validation. Autonomous agent frameworks compound this by generating reports at scale. When pointed at static analyzer output (which itself has high false-positive rates), the result is a cascade: low-confidence alerts are laundered through fluent language into reports that appear analytically rigorous despite the underlying signal being noise.

### 3.5 Summary and Comparative Analysis

Figure 1 and Table 1 converge on a single conclusion. The manifestations differ, but the origin remains the same. Hallucinated vulnerabilities, incorrect patches, and semantic repackaging all arise when probabilistic generation stands in for causal verification. The "Grounding" column is particularly telling: not a single study provides execution-level validation of claims in LLM outputs. The strongest grounding is compile checking or static analysis, both operating far below what security reasoning demands. Until generation is anchored to verification, these categories will continue to evolve, becoming more fluent and more difficult to distinguish from genuine analysis.

## 4 The Empirical Divide: Human Deductive Reasoning versus AI Probabilistic Generation

Figure 2 captures a tension at the heart of modern vulnerability analysis. Two systems receive the same input and walk in fundamentally different directions. One builds, tests, and verifies. The other predicts, connects, and completes. The gap is not a matter of performance. It is a difference in how each system understands what it means to be correct.

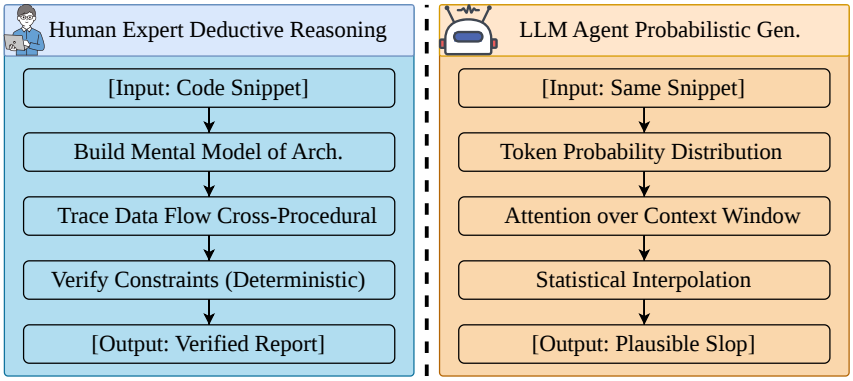


**Fig. 2.** Epistemic gap between human analysts and AR LLMs. Dashed arrows on the LLM side indicate optional external interactions (tool use, retrieval) that partially mitigate but do not close the gap. The human side includes heuristic shortcuts alongside deductive steps.

### 4.1 Human Cognitive Models in Vulnerability Assessment

On the left side of Figure 2, the human analyst constructs a mental model of the system. This model evolves as the analyst moves through the code, tracing data across functions, resolving dependencies, and identifying constraints governing execution. Votipka et al. observed that this process is a constant negotiation between architectural understanding and instruction-level detail [32]. The analyst forms a hypothesis, follows it through the program, and abandons it when it conflicts with observed behavior. Reachability is not assumed. It is proven.

This mode of reasoning depends on a persistent internal state carrying forward invariants never explicitly written yet always enforced. When a potential vulnerability appears, it is treated with skepticism. The question is not whether it looks like a known pattern but whether it survives the full chain of execution without contradiction.

It would be a mistake to idealize this process. Human analysts are not uniformly deductive. Real-world triage does not operate under ideal conditions. Experienced researchers rely heavily on pattern recognition and heuristic pruning, recognizing that a code structure "looks like" a known vulnerability class without fully tracing every path. They operate under time pressure, incomplete information, and cognitive biases such as anchoring on the first plausible hypothesis. Votipka et al. note that analysts frequently skip verification steps when a finding "feels right," a behavior that produces its own false positives. The difference is one of degree rather than kind. Human heuristics are grounded in experience with real execution failures. Model heuristics are grounded in token co-occurrence statistics with no causal relationship to program behavior. When humans err, they err within a framework that admits the possibility of error. When models err, they lack that meta-cognitive capacity.

### 4.2 Architectural Constraints of AR Generation

On the right side of Figure 2, LLMs follow a different path. They do not construct an internal execution trace. They process input as a sequence of tokens, estimating the probability of what should come next based on patterns learned during training. The notion of likelihood replaces the notion of correctness.

This distinction becomes pronounced as task complexity increases. Liu et al. (2024) showed that when context grows, attention becomes unevenly distributed, leading to a "lost in the middle" effect where critical information fades [19]. For vulnerability analysis, long-range dependencies and cross-procedural relationships are not consistently preserved. The model samples fragments of context and assembles them into a coherent narrative. What appears as reasoning is often reconstruction. Variable names, API calls, and security patterns are stitched together because they co-occur in similar contexts, not because they have been verified within a single execution path. When gaps emerge, they are filled with statistically plausible continuations. A legitimate data flow can quietly transition into an invented function call, not as an explicit error but as a natural extension of the learned distribution.

Increasing model scale does not resolve this tension. It often amplifies it, allowing the system to generate more coherent and persuasive narratives while remaining anchored to the same probabilistic foundation.

### 4.3 Partial Bridges: Chain-of-Thought, Tool Use, and Their Limits

The picture in Figure 2 would be incomplete without acknowledging architectural modifications designed to narrow this gap. Chain-of-thought (CoT) prompting, tool-augmented generation, and retrieval-grounded approaches each inject structure into otherwise unconstrained token prediction. The question is whether they help enough.

CoT encourages the model to externalize intermediate reasoning steps. In vulnerability analysis, this can manifest as explicit "trace data from source to sink" instructions. The improvement is concentrated in simple, single-function vulnerabilities where the reasoning chain is short. On cross-procedural vulnerabilities requiring three or more hops, CoT's benefit diminishes sharply. In several documented cases the model produced longer but equally incorrect reasoning chains, more words with no more verification. The model narrates its way toward an answer it has already probabilistically committed to.

Tool-using agents introduce a more promising mechanism. Frameworks such as ReAct allow models to invoke external utilities during generation. In practice, the results are mixed. The agent's ability to correctly formulate the query remains subject to the same token-level biases that produce slop in standalone generation. Pearce et al. observed that AI agents tasked with autonomous security auditing frequently invoke tools incorrectly, passing wrong arguments, querying the wrong functions, or misinterpreting structured output [22]. The tool call becomes a prop in a narrative rather than a genuine constraint on reasoning.

Retrieval-augmented generation (RAG) anchors the model's output to retrieved documents. This can prevent fabrication of nonexistent CVEs. The limitation is that retrieval operates at the document level, not the execution level. A retrieved CVE description confirms that a vulnerability class exists. It does not confirm that the specific code under analysis instantiates that class. RAG reduces factual hallucination but does not address logical hallucination, the case where the model draws a valid-sounding but causally incorrect inference from accurately retrieved facts.

These modifications narrow the gap but do not close it. They add friction to the generation process, which helps, but they do not substitute for execution-grounded verification.

### 4.4 Operationalizing the Gap: The Deductive Coverage Score

To move beyond schematic arguments, the reasoning gap must be measurable. We propose the **Deductive Coverage Score (DCS)** as a proxy for the degree to which a generated vulnerability claim is grounded in explicit, verifiable evidence rather than statistical interpolation.

Given a vulnerability claim $C$, a domain expert (or deterministic verification tool) decomposes $C$ into $k$ atomic constraints $\{c_1, c_2, \ldots, c_k\}$ that must each hold for the claim to be valid. For a typical memory-corruption claim, these might include: $c_1$ (tainted source reachable from a public entry point), $c_2$ (data flows to a dangerous sink without effective sanitization), $c_3$ (sink operation triggerable under realistic preconditions), $c_4$ (target buffer size insufficient for expected input). The DCS is:

$$\mathrm{DCS}(C) = \frac{1}{k} \sum_{i=1}^{k} \mathbf{1}[\text{claim } c_i \text{ is explicitly grounded in the output}] \tag{1}$$

"Explicitly grounded" means the output contains a concrete evidence anchor for $c_i$: a specific code reference, execution trace, concrete input specification, or formal proof fragment. Vague appeals to "the data flows through several functions" do not count.

## 5 Mitigating AI Slop: Content Provenance and Verification Strategies

Before surveying mitigation approaches, it is necessary to clarify what we are trying to mitigate. The core failure of AI slop is not that it was produced by a machine. A human analyst working from incomplete information under time pressure can produce an equally hollow report. An LLM-assisted analysis that has passed formal verification is useful regardless of its origin. The variable that matters is not provenance but correctness: whether the artifact carries verifiable substance that survives independent validation.

Much of the initial mitigation effort targeted provenance rather than correctness. This is understandable as a first response, but as we show, it encounters fundamental limits in security domains. We organize the landscape into three phases: historically attempted but misaligned approaches, ecosystem-level attestation, and the verification-first paradigm.

### 5.1 The Misalignment of Passive Statistical Detection

The initial wave of AI mitigation relied on statistical analysis of generated text to identify machine provenance. DetectGPT by Mitchell et al. (2023) identifies machine-generated content by evaluating probability curvature of a text sequence [21]. The premise is that LLMs generate text in negative curvature regions of the log-probability function, producing sequences with lower perplexity and burstiness than human writing. Supervised classifiers fine-tuned on architectures such as RoBERTa were deployed to detect semantic anomalies.

These approaches degrade significantly when applied to vulnerability reports, and the failure mode reveals why provenance is the wrong target. Security artifacts are inherently formulaic with low lexical diversity. Human-written bug bounty reports rely on rigid templates, structured stack traces, and standardized formatting that naturally produce the low perplexity scores statistical detectors flag as artificial. The detector distinguishes fluent-from-template from fluent-from-model, a distinction with no bearing on whether the underlying claim is true.

Sadasivan et al. (2025) formalized this fragility, proving that as generative model output distributions approximate human writing distributions, the ROC AUC for any statistical detector converges to random chance [24]. Applying these detectors to triage platforms produces unacceptably high false-positive rates, penalizing legitimate researchers while remaining vulnerable to adversarially engineered burstiness. Passive detection asks "was this written by a human?" when it should ask "is this claim correct?"

### 5.2 Cryptographic Watermarking and Ecosystem Attestation

Kirchenbauer et al. (2023) proposed soft watermarking that alters vocabulary distribution during token selection, dividing the token space into permissible and restricted lists to verify provenance [15]. In code generation and security patching, the vocabulary space is extremely constrained. Forcing a model to select a watermarked token in a deterministic logic chain frequently produces an incorrect API call, an invalid variable name, or a broken payload. Watermarking does not just fail to detect slop. It can actively produce slop by corrupting the logic it is supposed to authenticate.

Recognizing this, the community is pivoting toward ecosystem-level attestation. Inspired by supply chain frameworks such as *in-toto* [30], platforms are moving toward proving human provenance rather than machine provenance. Requiring researchers to cryptographically sign proofs-of-concept and execution

traces using authenticated public key infrastructure shifts the verification burden. Attestation is a pragmatic infrastructure defense, but it remains a provenance mechanism. A cryptographically signed report can still describe a phantom exploit path. The signature proves who submitted it, not whether it is true.

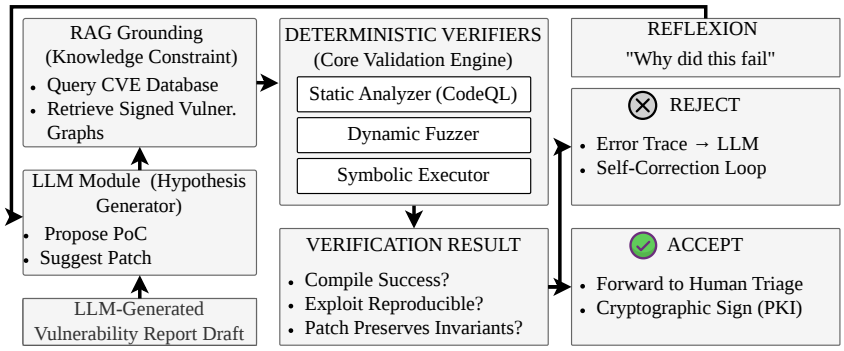


**Fig. 3.** Neuro-symbolic verification architecture for trustworthy triage. Each component maps to prior systems and documented limits discussed in Section 5.3.

### 5.3 Active Verification and Neuro-Symbolic Architectures

Given the limitations of passive detection and watermarking, the most resilient strategy shifts the target from provenance to correctness. The architecture in Figure 3 embodies a different premise: the language model generates hypotheses, and a pipeline subjects each hypothesis to deterministic verification before it reaches a human analyst. Below we map each component to concrete prior systems and their documented limits on security inputs.

**Retrieval-Augmented Grounding.** The model anchors claims in externally verified sources such as CVE databases [16]. RAG operates at the document level rather than the execution level. A retrieved CVE entry confirms a vulnerability class exists but does not confirm that the specific code under analysis instantiates it. Lewis et al. note that retrieval noise can degrade generation quality below the parameter-only baseline [16]. RAG reduces factual hallucination but does not address logical hallucination.

**Static Analysis Verification (CodeQL).** Generated claims are handed to an engine operating on the actual AST and control-flow graph. CodeQL is widely deployed and integrated into GitHub's infrastructure [1]. Its coverage on security-critical bug classes is uneven. Memory-safety vulnerabilities involving complex heap layouts require custom queries, and its path-sensitive analysis does not model heap state by default. Template-heavy C++ and indirect calls

through vtables produce false negatives that undermine verification [18]. CodeQL can confirm certain claims but cannot refute complex ones with reliable completeness.

**Dynamic Verification (Fuzzing).** For claims involving exploitability, the generated proof-of-concept is compiled and executed under a coverage-guided fuzzer such as AFL or libFuzzer [20]. Coverage-guided fuzzers struggle with magic-byte comparisons, deep conditional branches, and path constraints requiring specific heap layouts [17]. Furthermore, the verification quality depends entirely on the LLM-generated harness. As documented in Section 3.3, LLM-generated code frequently contains errors that prevent valid harnesses from compiling or exercising the intended path [37].

**Symbolic Execution Verification.** A symbolic executor such as KLEE [3] or angr [26] explores the path space to determine whether claimed constraints are satisfiable. This approach provides strong theoretical guarantees but faces path explosion as feasible paths grow exponentially with branch count. Constraint solvers time out on non-linear arithmetic, string operations, and floating-point comparisons. These limitations restrict practical coverage on the complex programs where slop is most likely to appear.

**Iterative Self-Correction (Reflexion).** When a claim fails verification, the error trace is fed back to the model. Shinn et al. (2025) introduced Reflexion as a framework for verbal reinforcement learning through iterative external feedback [25]. Effectiveness depends heavily on the granularity of this feedback. There is no theoretical guarantee of convergence. Huang et al. provide evidence that without access to external ground truth, models struggle to correct their own reasoning even when explicitly presented with their errors [10]. In security contexts, where error traces from fuzzers or symbolic executors can be voluminous and partially irrelevant, this limitation poses a tangible risk of reinforcing rather than resolving invalid generation.

**Synthesis.** Each component has blind spots. The pipeline raises the floor by requiring a hallucinated claim to survive at least two independent verification stages before reaching a human. Most slop fails at the first hurdle. The architecture shifts the burden from the human analyst to the system and ensures human judgment is exercised only on claims that have survived deterministic tests.

## 6 Open Challenges and Future Directions

Active verification architectures provide a foundational defense, but the rapid evolution of foundation models continuously shifts the threat landscape. We outline three trajectories, with concrete specifications for the evaluation instruments the field currently lacks.

### 6.1 Standardizing Benchmarks for Deceptive Generation

Existing benchmarks reward models for appearing correct rather than being verifiably right. HumanEval and MBPP reduce evaluation to binary functional

**Table 2.** Existing benchmarks vs. proposed evaluation instruments.

| Benchmark | Focus | Security Gap for AI Slop | |
|---|---|---|---|
| **HumanEval / MBPP** | Standalone code synthesis | None | Binary pass/fail; no security context |
| **SWE-bench** [12] | Repo-level patching | Partial | Evaluates regression, ignores deceptive fluency |
| **CyberSecEval** [2] | Compliance & insecure coding | High | Tests jailbreaks, not benign-code hallucination |
| **CVE-Bench** | Phantom exploit detection | High | Isolates deceptive generation with execution-validated ground truth |
| **Slop-Score** | Fluency vs. substance | High | Continuous metric penalizing the fluency-substance gap |

outcomes. SWE-bench considers repository context but overlooks whether the reasoning behind a patch is sound [12]. CyberSecEval asks whether a model will comply with malicious intent but not what happens when the intent is benign and the model invents a vulnerability anyway [2]. We specify two instruments to address this gap.

**CVE-Bench** CVE-Bench tests whether triage systems can separate grounded vulnerability reports from fluent outputs containing phantom exploit paths. Positive samples consist of verified recent CVEs paired with reproducible proofs-of-concept. Negative samples are LLM-generated reports on the same codebases, filtered to retain only those citing nonexistent CVEs or fabricating paths while scoring high on human fluency ratings.

The benchmark covers three tasks: binary classification (F1 on phantoms), ungrounded claim localization, and triage ranking (nDCG). To prevent the evaluation from collapsing into simple retrieval, the design uses temporal holdouts, a no-identifier variant forcing technical-content judgment, and mechanically paraphrased positives. Expected baselines span raw LLMs, chain-of-thought prompting, retrieval-augmented generation, and the full neuro-symbolic pipeline in Section 5.3.

**Slop-Score** Slop-Score provides a continuous metric quantifying the gap between linguistic polish and verifiable substance in a security artifact $O$:

$$S(O) = \frac{F(O)}{\alpha \cdot D(O) + \beta \cdot C(O) + \epsilon} \tag{2}$$

$F(O)$ is normalized fluency, computed via reference-model perplexity against a calibration corpus of human reports. $D(O)$ is evidence density, the fraction of technical claims in $O$ backed by explicit anchors such as code references, line numbers, or execution traces. $C(O)$ is the constraint satisfaction ratio, the fraction of required exploit-path constraints for the claimed CWE class that are explicitly verified in the output. $\alpha$ and $\beta$ are tunable hyperparameters prioritizing constraint verification over mere evidence citation, and $\epsilon$ prevents division by zero. Higher $S$ indicates fluent but hollow output. The metric is decomposable: a high score directs attention to whether the failure is driven by missing evidence, unverified constraints, or unusually polished language. Validation requires

computing the score on a held-out set of expert-written versus LLM-generated phantom reports, measuring discrimination via AUC-ROC, and ablating each component to confirm its contribution.

### 6.2 Defending Against the Adversarial Weaponization of Slop

Any filtering mechanism invites adversaries to study its boundaries. Open-weight models lower the barrier to fine-tuning specialized generators designed to evade detection. Zou et al. (2023) demonstrated how fragile model alignment is under targeted pressure, where constructed inputs force specific harmful outputs [39]. In vulnerability triage, this fragility takes a new form. Adversarial slop can be engineered to exploit assumptions of verification pipelines, trigger parsing edge cases, or consume disproportionate computational resources.

A particularly insidious variant exploits the component-level gaps documented in Section 5.3. An adversary aware that CodeQL has poor heap-coverage, that symbolic execution times out on complex loops, and that fuzzers struggle with magic-byte comparisons can craft a phantom report designed to pass through these blind spots. The report would describe a use-after-free with plausible but unverifiable heap assumptions, knowing no single component can conclusively refute it. Defending against this requires pipelines that anticipate manipulation. Triage architectures must account for computational cost, identifying inputs designed to exhaust resources and deprioritizing them before the verification budget is spent.

### 6.3 Verifiable Reasoning and Human-Centric Triage

Current generative models lack an internal sense of contradiction. When they fail, they fail with confidence, and when asked to reflect, they tend to produce explanations reinforcing the original error [10]. A hallucinated exploit path does not disappear when questioned. It evolves into a more elaborate narrative defending its own existence.

A reliable path forward places verification at the center and redefines the human role. Systems should require models to expose the structure of their reasoning. Intermediate representations, execution traces, or formal proofs of exploitability become the primary artifacts, with natural language serving only as a secondary explanation layer. The model is judged not by how convincingly it speaks but by whether its claims can be independently validated. The Deductive Coverage Score proposed in Section 4.4 provides a concrete mechanism for this interaction. A triage interface displaying the DCS breakdown shows which constraints are grounded and which are asserted without evidence, giving the analyst a direct signal about where to direct attention. Trust is no longer inferred from fluency but earned through verifiability.

## 7 Conclusion

The integration of LLMs into cybersecurity is reshaping vulnerability assessment. What depended on careful expert analysis is increasingly delegated to automated systems promising scale and efficiency. Beneath fluent reports and confident outputs lies a widening trust gap. Hallucinated vulnerabilities, incorrect patches, and recycled bug reports reflect a mismatch between how human experts reason and how models generate.

Human analysts build conclusions through constraint, verification, and deductive discipline, though they too rely on heuristics that can fail. Language models operate by extending patterns, predicting what is likely rather than what is true. Chain-of-thought prompting, tool augmentation, and retrieval grounding narrow this gap for simpler cases but do not close it for the multi-step reasoning that security demands. Statistical filtering and watermarking target provenance rather than correctness and face hard limits in logic-critical domains. Cryptographic attestation provides pragmatic infrastructure defense but cannot verify the substance of what it signs.

The path forward requires treating generation as the starting point of a verification pipeline, pairing generative models with deterministic evaluators while remaining explicit about where each evaluator's blind spots lie. This survey has moved the conversation past anecdote toward structure. The taxonomy, the measurable reasoning gap, the component-level analysis of verification architectures, and the concrete benchmark specifications all treat AI slop as a structural failure to be engineered out of the system. Trustworthy AI in cybersecurity will not emerge from better phrasing. It will come from systems that justify what they produce in terms that are checkable, reproducible, and grounded in formal reasoning.

## References

1. Avgustinov, P., de Moor, O., Jones, M.P., Schäfer, M.: QL: Object-oriented Queries on Relational Data. In: Krishnamurthi, S., Lerner, B.S. (eds.) 30th European Conference on Object-Oriented Programming (ECOOP 2016). Leibniz International Proceedings in Informatics (LIPIcs), vol. 56, pp. 2:1–2:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2016). https://doi.org/10.4230/LIPIcs.ECOOP.2016.2, `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2016.2`
2. Bhatt, M., Chennabasappa, S., Nikolaidis, C., Wan, S., Evtimov, I., Gabi, D., Song, D., Ahmad, F., Aschermann, C., Fontana, L., Frolov, S., Giri, R.P., Kapil, D., Kozyrakis, Y., LeBlanc, D., Milazzo, J., Straumann, A., Synnaeve, G., Vontimitta, V., Whitman, S., Saxe, J.: Purple llama cyberseceval: A secure coding benchmark for language models (2023), `https://arxiv.org/abs/2312.04724`
3. Cadar, C., Dunbar, D., Engler, D.: Klee: unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation. p. 209–224. OSDI'08, USENIX Association, USA (2008)

4. Chakraborty, S., Krishna, R., Ding, Y., Ray, B.: Deep learning based vulnerability detection: Are we there yet? IEEE Transactions on Software Engineering **48**(9), 3280–3296 (2022). https://doi.org/10.1109/TSE.2021.3087402
5. Chi, J., Qu, Y., Liu, T., Zheng, Q., Yin, H.: Seqtrans: Automatic vulnerability fix via sequence to sequence learning. IEEE Transactions on Software Engineering **49**(2), 564–585 (2023). https://doi.org/10.1109/TSE.2022.3156637
6. Dong, J., Chen, X., Susilo, W., Sun, N., Shaghaghi, A., Ma, S.: What Lies Beneath: An Empirical Study of Silent Vulnerability Fixes in Open-Source Software . In: 2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 345–357. IEEE Computer Society, Los Alamitos, CA, USA (Jun 2025). https://doi.org/10.1109/DSN64029.2025.00043, https://doi.ieeecomputersociety.org/10.1109/DSN64029.2025.00043
7. Ferrara, E.: Genai against humanity: Nefarious applications of generative artificial intelligence and large language models. Journal of Computational Social Science **7**(1), 549–569 (2024). https://doi.org/https://doi.org/10.1007/s42001-024-00250-1
8. Finifter, M., Akhawe, D., Wagner, D.: An empirical study of vulnerability rewards programs. In: 22nd USENIX Security Symposium (USENIX Security 13). pp. 273–288. USENIX Association, Washington, D.C. (Aug 2013), https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/finifter
9. Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang, H.: Large language models for software engineering: A systematic literature review. ACM Trans. Softw. Eng. Methodol. **33**(8) (Dec 2024). https://doi.org/10.1145/3695988, https://doi.org/10.1145/3695988
10. Huang, J., Chen, X., Mishra, S., Zheng, H.S., Yu, A.W., Song, X., Zhou, D.: Large language models cannot self-correct reasoning yet. In: The Twelfth International Conference on Learning Representations (2024), https://openreview.net/forum?id=IkmD3fKBPQ
11. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12) (Mar 2023). https://doi.org/10.1145/3571730, https://doi.org/10.1145/3571730
12. Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.R.: SWE-bench: Can language models resolve real-world github issues? In: The Twelfth International Conference on Learning Representations (2024), https://openreview.net/forum?id=VTF8yNQM66
13. Jin, D., Fu, Q., Li, Y.: Good News for Script Kiddies? Evaluating Large Language Models for Automated Exploit Generation . In: 2025 IEEE Security and Privacy Workshops (SPW). pp. 278–282. IEEE Computer Society, Los Alamitos, CA, USA (May 2025). https://doi.org/10.1109/SPW67851.2025.00039, https://doi.ieeecomputersociety.org/10.1109/SPW67851.2025.00039
14. Kabir, S., Udo-Imeh, D.N., Kou, B., Zhang, T.: Is stack overflow obsolete? an empirical study of the characteristics of chatgpt answers to stack overflow questions. In: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems. CHI '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3613904.3642596
15. Kirchenbauer, J., Geiping, J., Wen, Y., Katz, J., Miers, I., Goldstein, T.: A watermark for large language models. In: Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., Scarlett, J. (eds.) Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 202, pp.

17061–17084. PMLR (23–29 Jul 2023), https://proceedings.mlr.press/v202/kirchenbauer23a.html
16. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. NIPS '20, Curran Associates Inc., Red Hook, NY, USA (2020)
17. Li, Y., Chen, B., Chandramohan, M., Lin, S.W., Liu, Y., Tiu, A.: Steelix: program-state based binary fuzzing. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. p. 627–637. ESEC/FSE 2017, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3106237.3106295, https://doi.org/10.1145/3106237.3106295
18. Lipp, S., Banescu, S., Pretschner, A.: An empirical study on the effectiveness of static c code analyzers for vulnerability detection. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. p. 544–555. ISSTA 2022, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3533767.3534380, https://doi.org/10.1145/3533767.3534380
19. Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts. Transactions of the Association for Computational Linguistics **12**, 157–173 (2024). https://doi.org/10.1162/tacl_a_00638, https://aclanthology.org/2024.tacl-1.9/
20. Manès, V.J., Han, H., Han, C., Cha, S.K., Egele, M., Schwartz, E.J., Woo, M.: The art, science, and engineering of fuzzing: A survey. IEEE Transactions on Software Engineering **47**(11), 2312–2331 (2021). https://doi.org/10.1109/TSE.2019.2946563
21. Mitchell, E., Lee, Y., Khazatsky, A., Manning, C.D., Finn, C.: Detectgpt: zero-shot machine-generated text detection using probability curvature. In: Proceedings of the 40th International Conference on Machine Learning. ICML'23, JMLR.org (2023)
22. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R.: Asleep at the keyboard? assessing the security of github copilot's code contributions. Commun. ACM **68**(2), 96–105 (Jan 2025). https://doi.org/10.1145/3610721, https://doi.org/10.1145/3610721
23. Pearce, H., Tan, B., Ahmad, B., Karri, R., Dolan-Gavitt, B.: Examining Zero-Shot Vulnerability Repair with Large Language Models . In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 2339–2356. IEEE Computer Society, Los Alamitos, CA, USA (May 2023). https://doi.org/10.1109/SP46215.2023.10179420
24. Sadasivan, V.S., Kumar, A., Balasubramanian, S., Wang, W., Feizi, S.: Can AI-generated text be reliably detected? stress testing AI text detectors under various attacks. Transactions on Machine Learning Research (2025), https://openreview.net/forum?id=00gsAZdF0t
25. Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: language agents with verbal reinforcement learning. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. NIPS '23, Curran Associates Inc., Red Hook, NY, USA (2023)
26. Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., Feng, S., Hauser, C., Kruegel, C., Vigna, G.: Sok: (state of) the art of war: Offensive techniques in binary analysis. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 138–157 (2016). https://doi.org/10.1109/SP.2016.17

27. Stenberg, D.: Death by a thousand slops (2025), https://daniel.haxx.se/blog/2025/07/14/death-by-a-thousand-slops/, last accessed 2026/03/02
28. Sun, Y., Wu, D., Xue, Y., Liu, H., Ma, W., Zhang, L., Liu, Y., Li, Y.: Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning (2025), https://arxiv.org/abs/2401.16185
29. The Register: Curl shutters bug bounty program to stop ai slop (2026), https://www.theregister.com/2026/01/21/curl_ends_bug_bounty/, last accessed 2026/03/11
30. Torres-Arias, S., Afzali, H., Kuppusamy, T.K., Curtmola, R., Cappos, J.: in-toto: Providing farm-to-table guarantees for bits and bytes. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 1393–1410. USENIX Association, Santa Clara, CA (Aug 2019), https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias
31. Ullah, S., Han, M., Pujar, S., Pearce, H., Coskun, A., Stringhini, G.: LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks . In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 862–880. IEEE Computer Society, Los Alamitos, CA, USA (May 2024). https://doi.org/10.1109/SP54263.2024.00210
32. Votipka, D., Rabin, S.M., Micinski, K., Foster, J.S., Mazurek, M.M.: An observational investigation of reverse engineers' processes. In: Proceedings of the 29th USENIX Conference on Security Symposium. SEC'20, USENIX Association, USA (2020)
33. Votipka, D., Stevens, R., Redmiles, E., Hu, J., Mazurek, M.: Hackers vs. testers: A comparison of software vulnerability discovery processes. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 374–391 (2018). https://doi.org/10.1109/SP.2018.00003
34. Wikipedia contributors: curl (mar 2026), https://en.wikipedia.org/wiki/CURL, last accessed 2026/03/13
35. Wikipedia contributors: Daniel stenberg (mar 2026), https://en.wikipedia.org/wiki/Daniel_Stenberg, last accessed 2026/03/13
36. Xia, C.S., Zhang, L.: Automated program repair via conversation: Fixing 162 out of 337 bugs for $0.42 each using chatgpt. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. p. 819–831. ISSTA 2024, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3650212.3680323, https://doi.org/10.1145/3650212.3680323
37. Zhang, C., Zheng, Y., Bai, M., Li, Y., Ma, W., Xie, X., Li, Y., Sun, L., Liu, Y.: How effective are they? exploring large language model based fuzz driver generation. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. p. 1223–1235. ISSTA 2024, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3650212.3680355, https://doi.org/10.1145/3650212.3680355
38. Zhang, Z., Wang, C., Wang, Y., Shi, E., Ma, Y., Zhong, W., Chen, J., Mao, M., Zheng, Z.: Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation. Proc. ACM Softw. Eng. **2**(ISSTA) (Jun 2025). https://doi.org/10.1145/3728894
39. Zou, A., Wang, Z., Carlini, N., Nasr, M., Kolter, J.Z., Fredrikson, M.: Universal and transferable adversarial attacks on aligned language models (2023), https://arxiv.org/abs/2307.15043