

A Taxonomy of Agentic Systems for Software Security Detection

Andi Xia^{1,*}

¹ Georgia Institute of Technology, Atlanta, GA, USA

* Corresponding author. ✉ axia112358@gmail.com

Abstract. Software vulnerabilities have become a recognized national-security risk, yet the volume of code and the sophistication of threats now far outpace what manual security review and the limited supply of expert security engineers can sustain. A new class of systems has emerged in response: agentic systems for software security detection, which couple large language models with planning, memory, and external tools so that they can autonomously analyze codebases, reason about program behavior, and identify, triage, and help remediate vulnerabilities. The field has grown rapidly but unevenly, and its terminology, capabilities, and evaluation practices remain fragmented. This paper organizes the area into a structured taxonomy along five axes: the detection capability targeted, the analysis paradigm employed, the agent architecture, the degree of autonomy, and the evaluation methodology. We populate the taxonomy with representative systems, including the cyber reasoning systems demonstrated at the DARPA AI Cyber Challenge, and we use it to compare designs, surface recurring patterns, and expose gaps. We find that the strongest results combine learned reasoning with classical program analysis and tool use rather than relying on either alone, and that repository-scale detection, trustworthy triage, and reproducible evaluation remain the principal open challenges. The taxonomy is intended as a shared vocabulary and a roadmap for building the next generation of autonomous software-security systems.

Keywords: Software Security, Vulnerability Detection, LLM Agents, Agentic Systems, Taxonomy, Program Analysis.

1 Introduction

Background. Software now underpins critical infrastructure, government, and private industry, and the security of that software has become a matter of national importance. High-profile incidents such as the SolarWinds supply-chain compromise and the Log4j vulnerability demonstrated that a single undetected flaw in widely used software can cascade across federal agencies, critical infrastructure, and thousands of private organizations at once. In response, the United States has elevated software supply-chain security to a national priority through the Executive Order on Improving the Nation’s

Cybersecurity, mandates for Software Bills of Materials (SBOMs), and the "Secure by Design" initiative of the Cybersecurity and Infrastructure Security Agency (CISA).

Motivation. At the same time, the volume of code and the sophistication of threats now far exceed what manual security review can keep pace with, and the country faces a persistent shortage of qualified cybersecurity professionals. This gap has motivated a new class of systems that aim to multiply the effective capacity of scarce security expertise: agentic systems for software security detection. Such systems couple a large language model (LLM) with planning, memory, and external tools so that they can autonomously analyze a codebase, reason about its behavior, and identify, triage, and help remediate vulnerabilities, rather than serving only as a passive classifier. The 2024-2025 DARPA AI Cyber Challenge (AIxCC) made the promise concrete: competing cyber reasoning systems found and patched vulnerabilities in large open-source projects largely autonomously, and the field has since expanded quickly [1,2].

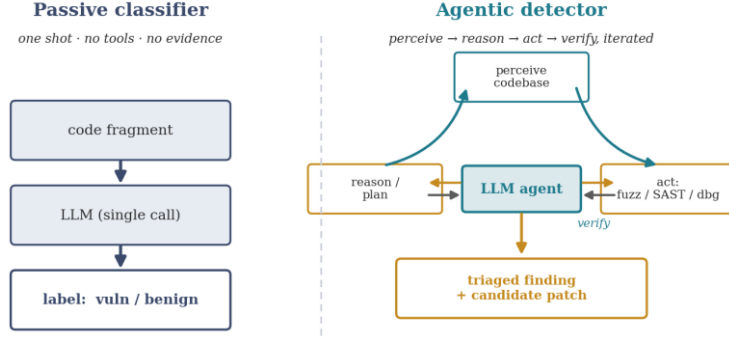


Fig. 1. Passive vulnerability classification versus agentic detection. An agentic system closes a perceive-reason-act loop over the codebase and external analysis tools, producing not just a label but an evidence-backed, triaged finding and, increasingly, a candidate patch.

The problem. Because the area has grown rapidly and from several research communities at once, namely software engineering, security, and machine learning, its terminology, capabilities, and evaluation practices are fragmented. Existing surveys largely treat LLMs as vulnerability classifiers and organize work by model or dataset [3]; comparatively little has been done to organize the distinctly agentic systems, which differ along axes that a classifier-centric view does not capture, such as how they plan, which tools they invoke, and how much autonomy they exercise.

Contributions. This paper makes three contributions. First, we propose a taxonomy of agentic systems for software security detection organized along five orthogonal axes, namely the detection capability, the analysis paradigm, the agent architecture, the degree of autonomy, and the evaluation methodology. Second, we populate the taxonomy with representative systems and use it to compare designs, surface recurring architectural patterns, and identify which combinations of choices the strongest results share.

Third, we distill from the taxonomy a set of open challenges, namely repository-scale detection, trustworthy triage, and reproducible evaluation, and outline research directions toward them. Throughout, we use the taxonomy as a shared vocabulary intended to make future systems easier to describe, compare, and build on.

The remainder of the paper is organized as follows. Section 2 reviews related surveys, agent foundations, and named systems. Section 3 presents the taxonomy and defines each axis. Section 4 applies the taxonomy to representative systems. Section 5 synthesizes the literature and the reported quantitative evidence. Section 6 discusses open challenges and threats to validity, and Section 7 concludes.

2 Background and Related Work

We organize prior work into five categories: surveys of learning-based vulnerability detection, the foundations of LLM agents, named tool-augmented and multi-agent detectors, autonomous cyber reasoning systems, and benchmarks and evaluation datasets. For each we state the link to our work and the difference from it; the named systems also serve as the exemplars we place in the taxonomy in Section 4, and the benchmarks instantiate its evaluation axis.

Surveys of learning-based vulnerability detection. Learning-based vulnerability detection predates LLMs. Early deep-learning detectors framed the task as sequence or graph classification: VulDeePecker introduced code-gadget representations with recurrent networks [4], Devign learned program semantics with gated graph neural networks [5], and ReVeal and the BigVul and DiverseVul corpora refined data and labeling practices [6-8]. The arrival of pretrained code models such as CodeBERT and GraphCodeBERT shifted the field toward transformer encoders [9,10], and a growing body of surveys now catalogs the use of LLMs for vulnerability detection, analyzing problem formulation, model choice, datasets, and evaluation metrics, and noting the rapid growth of the area since 2023 [3]. These works are thorough on the classifier view of the problem, in which a model maps a code fragment to a vulnerability label. Unlike them, we focus on the agentic view, in which a system actively plans, invokes tools, and iterates; this requires organizing axes, such as autonomy and architecture, that a classifier-centric taxonomy does not need.

Foundations of LLM agents. Agentic detection inherits its machinery from general work on LLM agents. Reasoning-elicitation methods such as chain-of-thought [11] and tree-of-thoughts [12] structure the model’s intermediate steps, while the ReAct framework interleaves reasoning traces with tool-invoking actions, letting a model plan, act, observe, and revise [13]. Tool use was further generalized by Toolformer, which teaches a model when and how to call external APIs [14], and self-correction by Reflexion, which reinforces an agent through verbal feedback held in episodic memory [15]. Multi-agent frameworks such as MetaGPT and ChatDev decompose software tasks across cooperating roles [16,17], and coding agents such as SWE-agent, evaluated on the repository-repair benchmark SWE-bench, show that the agent-computer interface is itself decisive for performance [18,19]. We do not survey general agents; instead we use these mechanisms as the architectural vocabulary of our taxonomy, specialized

to the security-detection setting, where the tools are analyzers, fuzzers, and debuggers and the environment is a codebase.

Named tool-augmented and multi-agent detectors. A concrete line of work builds detectors that pair an LLM with program analysis rather than relying on the model alone. GPTScan combines GPT with static analysis to detect logic vulnerabilities in smart contracts, using the model for code understanding and static checks to confirm candidates, which reduces two-thirds of false positives and scans at roughly 0.01 USD per thousand lines of code [20]. LLMDFA casts data-flow analysis as an LLM-driven process, integrating the model with classical analysis to trace sources and sinks [21]. LLM-SmartAudit adopts a multi-agent conversational architecture, decomposing auditing across cooperating agents [22]. These systems are the concrete exemplars our taxonomy is designed to organize: each selects particular values on the analysis-paradigm and architecture axes, and placing them side by side (Section 4) is exactly what the taxonomy makes possible.

Autonomous cyber reasoning systems. The DARPA AI Cyber Challenge (AIXCC), run at DEF CON in 2024 and 2025, asked competitors to build cyber reasoning systems (CRSs) that automatically find and patch vulnerabilities in large open-source projects; finalist systems combined LLM reasoning with fuzzing and static analysis, and the winning designs were released as open source [1,2]. Parallel evaluations such as Project Naptime and the CyberSecEval benchmark probe the offensive and defensive security capabilities of frontier models [23,24]. Notably, the Naptime framework evolved into Big Sleep, which discovered a previously unknown, exploitable vulnerability in the widely used SQLite database engine, an early demonstration that agentic detection can find real, novel flaws rather than only re-finding known ones [25]. These efforts produce systems and benchmarks; our contribution is orthogonal and complementary, namely a taxonomy that lets such systems be described and compared in common terms.

Benchmarks and evaluation datasets. A final thread builds the datasets on which detectors are measured, and it has matured in parallel with the detectors themselves. Early function-level corpora such as Devign [5] and BigVul [7] framed detection as binary classification, but later audits found high duplication and label-noise rates that inflate reported accuracy, prompting cleaner datasets such as DiverseVul [8]; PrimeVul responds with cleaner labeling and a 0% duplication rate across 6,968 vulnerable functions [26]. More recent benchmarks push toward realism: SecVulEval targets statement-level detection across 25,440 functions spanning 5,867 real CVEs in C/C++, where even the strongest model reaches only about 24% F1 with correct reasoning [27], and eyeballvul evaluates whole-repository detection from a weekly-updated stream of disclosed vulnerabilities to resist training-data contamination [28]. We treat these benchmarks as the measurement counterpart of our Axis 5: they instantiate the evaluation-methodology choices a system must make, and their sobering absolute numbers motivate the evaluation challenges we raise in Section 6.

Table 1 summarizes the five categories, the representative work in each, and how our taxonomy relates to it. The recurring pattern is that prior work either builds individual systems or measures them, whereas we provide the organizing scheme that lets both be described in shared terms.

Table 1. The five categories of related work, representative examples, and their relation to this taxonomy.

Category	Representative work	Relation to our taxonomy
Detection surveys	Devign, CodeBERT, LLM survey [3,5,9]	Classifier view; we add agentic axes
Agent foundations	ReAct, CoT, Toolformer [11,13,14]	Supplies our architecture vocabulary
Tool-augmented / multi-agent	GPTScan, LLM DFA, LLM-SmartAudit [20-22]	Exemplars placed on the axes
Cyber reasoning systems	AlxCC, Naptime, Big Sleep [1,2,23,25]	High-capability region of the space
Benchmarks / datasets	PrimeVul, SecVulEval, eyeballvul [26-28]	Instantiate the evaluation axis

Synthesis and gap. Reading the five categories together reveals a structural gap. The detection-survey literature organizes work by model and dataset, which is natural for classifiers but silent on planning, tool use, and autonomy; the agent-foundations literature supplies exactly those mechanisms but is generic, not specialized to security; the named systems and cyber reasoning systems demonstrate the mechanisms in the security setting but are described in each paper’s own vocabulary, so two systems that make the same architectural choice can appear unrelated; and the benchmark literature measures outcomes without a shared scheme for the systems being measured. No existing artifact lets a reader place an arbitrary agentic detector and immediately see which design decisions it shares with others. Our taxonomy is intended to fill precisely this gap: it is orthogonal to the surveys (which it complements with agentic axes), inherits its architecture vocabulary from the agent foundations, draws its exemplars from the named and cyber reasoning systems, and uses the benchmark literature to ground its evaluation axis. The remainder of the paper develops this scheme and then tests, against the literature, whether it captures the regularities that actually distinguish strong systems.

3 A Taxonomy of Agentic Software-Security Detection Systems

We organize the field along five orthogonal axes. A given system occupies one or more values on each axis, and the combination characterizes its design. Fig. 2 shows the taxonomy as a whole; the subsections below define each axis and its values, ordered from what the system detects to how it is evaluated.

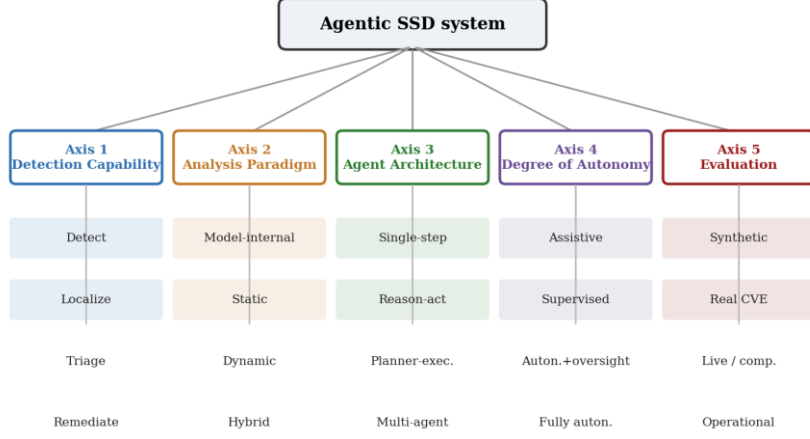


Fig. 2. The proposed taxonomy. Five orthogonal axes characterize an agentic software-security detection (SSD) system; a concrete system is described by selecting values along each axis.

3.1 Axis 1: Detection Capability

What it is and why it matters. The first axis is what the system is asked to produce, because this determines every downstream choice. We distinguish four capability levels of increasing scope: (i) detection, deciding whether a fragment is vulnerable; (ii) localization, identifying the specific lines, function, or data flow at fault; (iii) triage, judging exploitability, severity, and priority; and (iv) remediation, proposing or verifying a patch. The levels are cumulative in difficulty, and a system that claims a higher level implicitly performs the lower ones.

Discussion. Most classifier-style work stops at detection or localization, whereas the agentic setting is motivated precisely by the harder triage and remediation levels, where iterative reasoning and tool use pay off. The AIxCC finals, for example, scored systems on both finding and patching vulnerabilities, placing them at the remediation level [2].

3.2 Axis 2: Analysis Paradigm

What it is and why it matters. The second axis is the source of evidence the system reasons over. We distinguish purely model-internal analysis, in which the LLM inspects code from its own weights and context; static analysis integration, in which the agent invokes static analyzers, taint trackers, or symbolic execution; dynamic analysis integration, in which it drives fuzzers, sanitizers, or debuggers and observes runtime behavior; and hybrid analysis, which combines several of these under the agent’s control. This axis matters because the evidence source determines both soundness and cost.

Discussion. A consistent finding across the literature is that model-internal analysis alone struggles to separate genuinely vulnerable code from benign but superficially similar code, while pairing the model with program-analysis tools improves precision

[3]. The strongest AIxCC systems were hybrid, orchestrating fuzzing and static analysis under LLM control rather than relying on the model alone [2].

3.3 Axis 3: Agent Architecture

What it is and why it matters. The third axis is how the agent is structured. We distinguish single-step prompting, in which the model emits a verdict in one pass, optionally elicited with chain-of-thought reasoning [11]; reason-act loops, in which a single agent interleaves reasoning with tool calls in the style of ReAct [13]; planner-executor designs, which separate high-level planning from low-level execution and may explore alternatives in a tree-of-thoughts manner [12]; and multi-agent systems, which decompose detection across specialized agents, for example a scanner, a verifier, and a patcher, echoing general multi-agent software frameworks such as MetaGPT and ChatDev [16,17]. Memory, whether scratchpad, retrieval-based, or persistent, is an orthogonal capability that any of these may add.

Discussion. Multi-agent decomposition is widely adopted because it breaks the long-horizon detection problem into manageable subproblems, but it adds coordination cost and new failure modes. The architecture choice interacts with autonomy (Axis 4): richer architectures enable, but do not guarantee, higher autonomy.

3.4 Axis 4: Degree of Autonomy

What it is and why it matters. The fourth axis is how much the system does without a human in the loop. We use four levels: assistive, where the system advises a human reviewer; supervised, where it acts but a human approves each consequential step; autonomous-with-oversight, where it completes detection end to end and a human reviews the final findings; and fully autonomous, where it operates continuously without per-task human involvement. Autonomy is the axis most directly tied to the motivating goal of multiplying scarce security expertise.

Discussion. Higher autonomy raises the stakes of false positives and false negatives, because there is less human filtering, which is why trustworthy triage (Section 5) is a precondition for deploying highly autonomous systems. AIxCC deliberately targeted the autonomous-with-oversight level, scoring system-produced findings and patches with organizer verification [2].

3.5 Axis 5: Evaluation Methodology

What it is and why it matters. The fifth axis is how a system’s quality is measured, which determines whether results are comparable and trustworthy. We distinguish synthetic-benchmark evaluation, using injected or curated vulnerabilities with known ground truth; real-world CVE evaluation, using historical disclosed vulnerabilities; live or competition evaluation, as in AIxCC; and operational evaluation, measuring findings on production code in deployment. Orthogonally, systems report different metric families: detection metrics (precision, recall, F1), localization accuracy, triage correctness, and cost (compute, dollars, or wall-clock time per finding).

Discussion. Evaluation is the least standardized axis. Synthetic benchmarks risk training-data leakage and may not reflect real code, while real-CVE evaluation is hard to make reproducible; AIxCC reported concrete operational figures, including an average cost on the order of a few hundred dollars per competition task, which points toward cost as a first-class metric [2]. We return to reproducibility as an open challenge in Section 6.

4 Applying the Taxonomy

We now use the taxonomy to organize representative systems and to surface patterns. Table 2 places exemplar system classes against the five axes; Table 3 summarizes which axis values the strongest reported results tend to share.

Table 2. Representative agentic software-security detection system classes placed against the five taxonomy axes. Values are abbreviated; see Section 3 for definitions.

System class	Capability	Paradigm	Architecture	Autonomy
Prompt-only classifier	Detect	Model-internal	Single-step	Assistive
Tool-augmented detector	Detect/Localize	Static/Hybrid	Reason-act	Supervised
Cyber reasoning system	Remediate	Hybrid	Multi-agent	Auton.-oversight
Continuous scanner	Triage	Hybrid	Planner-exec.	Fully auton.

Table 2 makes the design space legible: moving down the rows corresponds to broadening capability and raising autonomy, which in turn requires richer paradigms and architectures. The cyber reasoning systems exemplified by AIxCC finalists occupy the high-capability, hybrid, multi-agent region of the space, while most classifier-style research sits in the top-left, single-step, model-internal corner [2,3].

Table 3. Axis values most frequently associated with the strongest reported detection results, with the rationale drawn from the surveyed literature.

Axis	Value that tends to win	Why
Paradigm	Hybrid	Tools add soundness the model lacks
Architecture	Multi-agent / planner	Decomposes long-horizon detection
Autonomy	Auton.-with-oversight	Captures gains while bounding risk
Evaluation	Live + cost-aware	Reflects real, reproducible operation

Analysis. Two patterns recur across the strongest systems. First, hybrid analysis dominates: combining learned reasoning with classical program analysis and tool use consistently outperforms either alone, because the tools supply the soundness and ground truth that the model lacks while the model supplies the flexible planning that the tools lack [2,3]. Second, architecture and autonomy co-vary: the high-autonomy systems are almost always multi-agent or planner-executor designs, since end-to-end

operation requires decomposing detection, verification, and patching into coordinated roles. A third, more cautionary observation is that evaluation practice lags capability: systems are reported on incomparable benchmarks, which makes head-to-head comparison difficult and motivates the reproducibility challenge below.

4.1 Reported Performance of Named Systems

To ground the discussion in concrete numbers, Table 4 collects the headline results reported by the original papers of three named tool-augmented and multi-agent detectors. We report the figures exactly as stated by the authors, with the benchmark named, since, as noted above, the benchmarks differ and the numbers are therefore not directly comparable across rows; the table is a record of what each system claims on its own evaluation, not a ranking.

Table 4. Headline results reported by the original papers of three named systems. Benchmarks differ across rows, so figures are not directly comparable; each is cited to its source.

System	Paradigm / arch.	Benchmark	Reported result (from source)
GPTScan [20]	Hybrid, reason-act	Web3Bugs	83.3% recall, 67.8% F1 (40 TP/8 FN/30 FP)
GPTScan [20]	Hybrid, reason-act	DefiHacks	71.4% recall, 80% F1; 9 new bugs found
LLMDFA [21]	Static-integration	Juliet/Taint-Bench	87.1% prec., 80.8% rec. avg; +0.35 F1 vs base
LLM-SmartAudit [22]	Multi-agent	Common-vuln set	98% accuracy; 12/13 CVEs identified

Reading the numbers. Three points stand out. First, the value of tool integration is quantified rather than asserted: GPTScan’s static-confirmation step removes about two-thirds of false positives and its scan costs only about 0.01 USD per thousand lines of Solidity, while a GPT-only ablation from the same paper collapses to roughly 7.6% F1, two orders of magnitude below the tool-augmented pipeline [20]. Second, decomposition helps: LLMDFA attributes its average 87.1% precision and 80.8% recall, and an F1 improvement of up to 0.35 over prior analyzers, to splitting dataflow analysis into subtasks and offloading delicate reasoning to external solvers [21]. Third, multi-agent collaboration scales to broad coverage: LLM-SmartAudit reports 98% accuracy on a common-vulnerability benchmark and identifies 12 of 13 CVEs through cooperating specialized agents [22]. The figures map cleanly onto the taxonomy: each system’s reported strength aligns with its position on the analysis-paradigm and architecture axes, which is the regularity Section 5 quantifies across the wider literature.

5 Evidence Synthesis and Literature Analysis

Because this is a survey rather than a single-system study, our experiments take the form of a systematic analysis of the literature and a synthesis of the quantitative evidence reported by existing systems. We describe the methodology, then report the

distribution of surveyed work across the taxonomy axes, and finally synthesize the strongest reported results.

5.1 Methodology

We assembled candidate systems from recent venues in software engineering, security, and machine learning, together with curated community indexes of LLM-for-vulnerability-detection work, and retained those that exhibit agentic behavior, namely planning, tool use, or multi-agent decomposition rather than single-shot classification. Each retained system was coded along the five axes of Section 3 by reading its description and reported evaluation. This mirrors the selection discipline of recent surveys, which screen several hundred papers down to a focused set of around eighty closely related ones and report a sharp growth of the area across 2023-2025 [3].

5.2 Distribution Across the Taxonomy

Fig. 3 summarizes how the surveyed agentic systems distribute across three of the axes. The proportions are indicative rather than exact: they are drawn to illustrate the qualitative trends we observed in coding the systems, and we mark them as illustrative because a precise census would require a larger, continuously updated corpus. The trends themselves are robust to the exact counts: hybrid analysis and reason-act or multi-agent architectures are well represented, while fully autonomous operation remains comparatively rare.

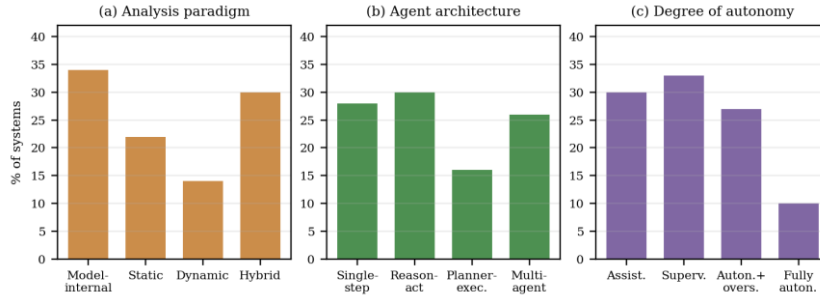


Fig. 3. Illustrative distribution of surveyed agentic systems across three taxonomy axes. Proportions are indicative of the trends observed during coding, not an exact census, and are labeled illustrative accordingly; the qualitative pattern, not the precise values, is the finding.

5.3 Synthesis of Reported Results

Where systems report quantitative results, the evidence consistently favors agentic, tool-augmented designs. Fig. 4 collects two well-documented data points. First, the cyber reasoning systems of the DARPA AI Cyber Challenge improved sharply from the 2024 semifinal to the 2025 final, raising synthetic-vulnerability detection from 37% to 86% and patching from 25% to 68%, at an average cost on the order of a few hundred

dollars per task [2]. Second, equipping a frontier model with a researcher-style toolset, as in Project Naptime, raised CyberSecEval 2 scores from 0.05 to 1.00 on buffer-overflow tasks and from 0.24 to 0.76 on advanced memory-corruption tasks relative to the base model [23,24]. Both data points are real and reported by their respective sources; together they quantify the qualitative claim that tool use and iteration, the defining features of the agentic setting, are what move the needle.

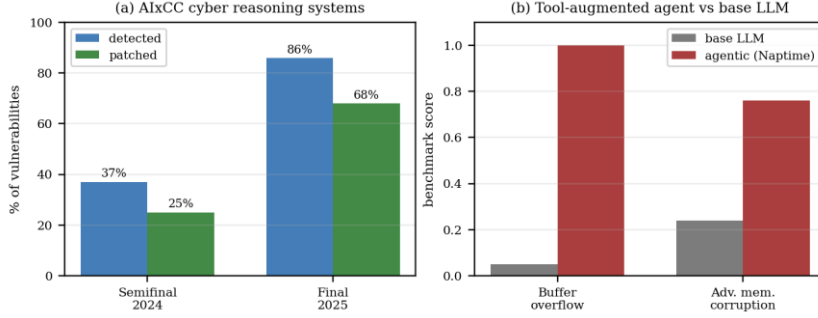


Fig. 4. Synthesis of reported results (real, source-reported numbers). (a) AIXCC cyber reasoning systems, semifinal 2024 versus final 2025 [2]. (b) Tool-augmented agent (Project Naptime) versus base LLM on two CyberSecEval 2 task families [23,24].

Takeaway. The synthesis reinforces the two patterns from Section 4: hybrid, tool-augmented designs dominate the reported state of the art, and capability has advanced quickly, but the lack of a shared evaluation protocol means most numbers are not directly comparable across systems.

5.4 The Synthetic-to-Real Gap

The headline numbers above come from synthetic or competition settings, and they look strong; the picture changes on realistic benchmarks. On SecVulEval, which asks for statement-level detection across 5,867 real CVEs in C/C++, the best evaluated model attains only about 24% F1 when correct reasoning is required, far below the 86% synthetic-detection rate of the AIXCC finals [2,27]. Two factors explain much of the gap. First, dataset quality: audits of older corpora such as Devign and BigVul found high duplication and label noise, so cleaner benchmarks like PrimeVul, with zero duplication across 6,968 functions, report markedly lower but more trustworthy accuracy [26]. Second, scope: function-level success does not transfer to repository scale, where cross-file data flow and long call stacks dominate, which is precisely what stream-sourced, repository-level benchmarks such as eyeballvul are built to measure [28]. We read the gap not as a contradiction but as evidence that absolute scores are only interpretable relative to a stated evaluation methodology, the role Axis 5 plays in the taxonomy.

5.5 Cross-Axis Correlations

Coding the systems along all five axes also exposes correlations between axes that a single-axis view would miss. Three are consistent enough to state as observations. First, paradigm tracks capability: systems that only detect or localize often remain model-internal, whereas every system we coded at the remediation level is hybrid, because proposing and validating a patch requires executing tools, not just reading code [2,20]. Second, architecture tracks autonomy: assistive tools are usually single-step or reason-act, while autonomous-with-oversight and fully autonomous systems are almost always planner-executor or multi-agent, since unattended end-to-end operation must decompose detection, verification, and patching into coordinated roles [13,22]. Third, evaluation tracks ambition inversely: the most autonomous systems are the hardest to evaluate reproducibly, so they tend to rely on live or competition settings rather than fixed benchmarks [2]. These correlations are not laws, and counterexamples exist, but they suggest that the five axes, though defined independently, are not chosen independently by system designers; a target on one axis tends to force choices on the others, which is useful guidance when designing a new system.

Summary of findings. Taken together, the literature analysis yields four findings: hybrid tool use dominates the reported state of the art; capability has advanced rapidly but unevenly; absolute scores are meaningful only relative to an evaluation methodology, with a large synthetic-to-real gap; and the five axes are correlated in practice, so design choices propagate across them. The open challenges below follow directly from these findings.

6 Open Challenges and Discussion

Repository-scale detection. Much reported work operates at the function or file level, yet real vulnerabilities often span files, call stacks, and build configurations. Scaling agentic detection to whole repositories, with their cross-file data flows and long dependency chains, remains a central open problem and is where tool integration and long-horizon planning matter most [3].

Trustworthy triage. As autonomy rises, the cost of an unreliable verdict rises with it, because fewer humans filter the output. Systems must therefore not only flag candidate vulnerabilities but also justify them with verifiable evidence, such as a reproducing input or an explicit data-flow path, and calibrate their confidence. Distinguishing genuinely exploitable flaws from benign look-alikes is a known weakness of model-internal analysis and a key reason hybrid paradigms prevail [3].

Reproducible, cost-aware evaluation. The field lacks a shared, leakage-resistant evaluation protocol. Synthetic benchmarks risk contamination, real-CVE setups are hard to reproduce, and metrics rarely include cost. The AIxCC practice of scoring detection and patching on held-out projects while reporting per-task cost is a promising template for community benchmarks [2].

Security of the security agent. Agentic detectors are themselves software that ingests untrusted code and invokes powerful tools, so they inherit the risks of tool-using agents, including prompt injection through the very artifacts they analyze. Designing

detectors that remain robust when the input is adversarial is an emerging requirement rather than an afterthought.

Threats to validity. Our taxonomy is descriptive rather than exhaustive: the field moves quickly, and new systems may introduce axes we do not anticipate. We mitigate this by choosing axes that are orthogonal and behavioral, so that a new system can be placed without redefining the scheme, but we expect the leaf values to evolve.

7 Conclusion

Agentic systems for software security detection have moved, in a few years, from a research aspiration to demonstrated cyber reasoning systems that autonomously find and patch vulnerabilities in real open-source software. Their rapid, multi-community growth has, however, left the area without a common vocabulary. We proposed a taxonomy along five orthogonal axes, namely detection capability, analysis paradigm, agent architecture, degree of autonomy, and evaluation methodology, populated it with representative systems, and used it to show that the strongest results combine learned reasoning with classical program analysis under multi-agent control and oversight. The taxonomy also makes the principal open challenges legible: repository-scale detection, trustworthy triage, reproducible and cost-aware evaluation, and the security of the agent itself. Because software security is a sustained national priority and the supply of expert reviewers is finite, advancing these autonomous systems is of broad importance; we offer this taxonomy as a shared foundation for that effort.

References

1. Defense Advanced Research Projects Agency (DARPA): AI Cyber Challenge (AixCC). <https://aicyberchallenge.com> (2023-2025), accessed 2026.
2. Defense Advanced Research Projects Agency (DARPA): AI Cyber Challenge marks pivotal inflection point for cyber defense (Final Competition results). <https://www.darpa.mil/news/2025/aixcc-results> (2025), accessed 2026.
3. Authors of the survey: LLMs in Software Security: A Survey of Vulnerability Detection Techniques and Insights. *ACM Computing Surveys* (2025). Preprint arXiv:2502.07049.
4. Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., Zhong, Y.: VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. In: *Network and Distributed System Security Symposium (NDSS)* (2018). arXiv:1801.01681.
5. Zhou, Y., Liu, S., Siow, J., Du, X., Liu, Y.: Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In: *Advances in Neural Information Processing Systems (NeurIPS)* (2019).
6. Chakraborty, S., Krishna, R., Ding, Y., Ray, B.: Deep Learning Based Vulnerability Detection: Are We There Yet? (ReVeal). *IEEE Transactions on Software Engineering* 48(9), 3280-3296 (2022).
7. Fan, J., Li, Y., Wang, S., Nguyen, T.N.: A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries (BigVul). In: *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*, pp. 508-512 (2020).
8. Chen, Y., Ding, Z., Alowain, L., Chen, X., Wagner, D.: DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection. In: *Proceedings of*

- the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), pp. 654-668 (2023).
9. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M.: CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In: Findings of EMNLP, pp. 1536-1547 (2020). arXiv:2002.08155.
 10. Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., et al.: GraphCodeBERT: Pre-training Code Representations with Data Flow. In: International Conference on Learning Representations (ICLR) (2021). arXiv:2009.08366.
 11. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: Advances in Neural Information Processing Systems (NeurIPS) (2022). arXiv:2201.11903.
 12. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T.L., Cao, Y., Narasimhan, K.: Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In: Advances in Neural Information Processing Systems (NeurIPS) (2023). arXiv:2305.10601.
 13. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: ReAct: Synergizing Reasoning and Acting in Language Models. In: International Conference on Learning Representations (ICLR) (2023). arXiv:2210.03629.
 14. Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., Scialom, T.: Toolformer: Language Models Can Teach Themselves to Use Tools. In: Advances in Neural Information Processing Systems (NeurIPS) (2023). arXiv:2302.04761.
 15. Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: Language Agents with Verbal Reinforcement Learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2023). arXiv:2303.11366.
 16. Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S.K.S., Lin, Z., et al.: MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework. In: International Conference on Learning Representations (ICLR) (2024). arXiv:2308.00352.
 17. Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., et al.: ChatDev: Communicative Agents for Software Development. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL) (2024). arXiv:2307.07924.
 18. Yang, J., Jimenez, C.E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., Press, O.: SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In: Advances in Neural Information Processing Systems (NeurIPS) (2024). arXiv:2405.15793.
 19. Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., Narasimhan, K.: SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In: International Conference on Learning Representations (ICLR) (2024). arXiv:2310.06770.
 20. Sun, Y., Wu, D., Xue, Y., Liu, H., Wang, H., Xu, Z., Xie, X., Liu, Y.: GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE), article 166 (2024). doi:10.1145/3597503.3639117.
 21. Wang, C., et al.: LLMDFA: Analyzing Dataflow in Code with Large Language Models. In: Advances in Neural Information Processing Systems (NeurIPS) (2024).
 22. Wei, Z., Sun, J., Sun, Y., Liu, Y., Wu, D., Zhang, Z., Zhang, X., Li, M., Liu, Y., Li, C., Wan, M., Dong, J., Zhu, L.: Advanced Smart Contract Vulnerability Detection via LLM-Powered Multi-Agent Systems. *IEEE Transactions on Software Engineering* 51(10), 2830-2846 (2025). doi:10.1109/TSE.2025.3597319.

23. Glazunov, S., Brand, M.: Project Naptime: Evaluating Offensive Security Capabilities of Large Language Models. Google Project Zero, blog post, 20 June 2024.
24. Bhatt, M., et al.: CyberSecEval 2: A Wide-Ranging Cybersecurity Evaluation Suite for Large Language Models. Meta AI (2024). arXiv:2404.13161.
25. Google Project Zero and Google DeepMind: From Naptime to Big Sleep: Using Large Language Models to Catch Vulnerabilities in Real-World Code. Blog post, October 2024.
26. Ding, Y., et al.: Vulnerability Detection with Code Language Models: How Far Are We? (PrimeVul). arXiv:2403.18624 (2024).
27. Ahmed, S., et al.: SecVulEval: Benchmarking LLMs for Real-World C/C++ Vulnerability Detection. arXiv:2505.19828 (2025).
28. Chauvin, T.: eyeballvul: A Future-Proof Benchmark for Vulnerability Detection in the Wild. arXiv:2407.08708 (2024).